Multi-Objectivization in Genetic Algorithms

A dissertation submitted in partial fulfillment
of the requirements for the degree
of Doctor of Philosophy

By

DARRELL F. LOCHTEFELD
B.S., Wright State University, 1999
M.B.A., The Ohio State University, 2004

2011
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

MAY 26, 2011

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Darrell F. Lochtefeld ENTITLED Multi-Objectivization in Genetic Algorithms BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

_____

Frank W. Ciarallo, Ph.D.
Dissertation Advisor

_____

Ramana V. Grandhi, Ph.D.
Director, Ph.D. in
Engineering Program

Committee on
Final Examination

_____

Andrew Hsu, Ph.D.
Dean, School of Graduate Studies

_____

Frank W. Ciarallo, Ph.D.

_____

Raymond R. Hill, Ph.D.

_____

Yan Liu, Ph.D.

_____

W. Paul Murdock, Ph.D.

_____

Mateen Rizki, Ph.D.

ABSTRACT

Lochtefeld, Darrell F., Ph.D. Dissertation, Department of Biomedical, Industrial, and Human Factors Engineering, Engineering Ph.D. Program, Wright State University, 2011. Multi-Objectivization in Genetic Algorithms

Multi-objectivization is the process of reformulating a single-objective problem into a multi-objective problem and solving it with a multi-objective method in order to provide a solution to the original single-objective problem. Multi-objectivization differs from other traditional divide-and-conquer techniques – the method splits the objective function rather than the search space. Prior to recent evidence, such reformulations were thought to make a problem more complex. However, more recent research suggests that decomposition in the objective space can lead to useful optimization techniques when coupled with a population-based search.

Machine based optimization techniques are varied but often based upon an analogy to real-world phenomena. A family and sub-family of algorithms called evolutionary and genetic algorithms respectively are inspired by Darwin's survival-of-the-fittest theory and the concept of modeling evolutionary process. These algorithms manage a population of solutions in a global search process that recombines and creates new solutions in order to generate useful solutions to hard problems. A sub-type of Evolutionary Algorithms (EAs) called Multiple Objective Evolutionary Algorithms (MOEAs) are designed to search for solutions to problems formulated with multiple-objectives. Multiple-objective problems have two or more partially competing objectives such as minimization of cost and maximization of safety. The application of MOEAs to solve problems that in their most natural formulation have a single objective is relatively new.

This work investigates Genetic Algorithms (GAs) and their close relatives MOEAs in both a general categorical sense and as they are applied to multi-objectivization. A diversity

classification framework for GAs is proposed. Furthermore, multi-objectivization techniques are examined. Through study of an abstract problem, job-shop scheduling problems, and the Traveling Salesman Problem, principles governing the design decisions for multi-objectivization are identified. Two ways in which multi-objectivization creates beneficial search results are theorized: a theory of how multi-objectivization effectively pairs fitness improvements with fitness decrements is developed and a theory for how multi-objectivization circumvents non-linear interactions is proposed. Evidence for both of these beneficial effects is provided through a series of computational experiments. Two prevalent multi-objectivization techniques are compared both analytically and through these experiments. A third, more general version of the studied techniques is proposed with results showing robust performance across a variety of computational budgets.

Table of Contents

Chapter 1 Introduction 1
1.1 An analogy toward multi-objectivization 1
1.2 Research Overview 3
1.3 Contributions to date 11
1.4 Publications 11
1.4.1 Peer review journal articles 11
1.4.2 Peer review conference proceedings 12
Chapter 2 Diversity in Genetic Algorithms (GAs) 13
2.1 Select GA Concepts 15
2.1.1 Phenotype space, genotype space and the concept of a problem landscape 15
2.1.2 The schema theorem 17
2.1.3 The building block hypothesis 18
2.1.4 Exploration / exploitation balance 19
2.1.5 Neighborhoods and the measure of distance 20
2.1.6 Selection pressure 21
2.1.7 Properties / processes that can degrade GA performance 22
2.2 Diversity management choices 31
2.2.1 Problem formulation 33
2.2.2 Solution representation and the fitness function 34
2.2.3 Constraint handling and repair operators 41
2.2.4 Initial population generation & restarts 41
2.2.5 Survivor selection strategies 42
2.2.6 Parent selection strategies 45
2.2.7 Recombination operator strategies 46
2.2.8 Mutation operator strategies 48
2.2.9 Parameter values & parameter selection strategies 49
2.3 Past diversity-based classifications 52
2.4 Classification of solution representations 55
2.5 Classification of constraint handling, fitness evaluation, and population structure 57
2.6 Classification of Population Structure 57
2.7 Classification of GA Operators 58
2.7.1 Complexity 59
2.7.2 Randomness 59
2.7.3 Occurrence Pattern 59
2.7.4 Diversity Preservation vs. Promotion 60
2.7.5 Directness 60
2.8 Example classifications of algorithms based on diversity 62
Chapter 3 Multi-Objective Optimization (MOO) and MOEAs 67
3.1 Multi-Objective Evolutionary Algorithms (MOEAs) 68
3.2 Solving single objective problems using MOO methods (multi-objectivization) 72
3.2.1 An example of multi-objectivization 78
3.3 Foundational development of helpers 80
Chapter 4 Helper methods applied to the JSSP 82
4.1 Specifics of the JSSP 82
4.1.1 JSSP solution methods 86
4.1.2 GA details for the JSSP 87
4.2 Helpers applied to the JSSP 89
4.2.1 Shortest Job First (SJF) helper sequencing 93

4.2.2 Investigating how helpers accelerate search 97
4.2.3 Exploring helper size 104
4.2.4 Helper effectiveness and non-dominated front size 108
4.3 Chapter Conclusion 113
Chapter 5 Multi-objectivization with the Tunable Objectives Problem (TOP) 115
5.1 Background 116
5.1.1 Multi-objectivization 117
5.1.2 Basic related work 119
5.2 The Weise model 120
5.2.1 Genotype creation 121
5.2.2 Redundancy reduction 121
5.2.3 Epistasis transformation 122
5.2.1 Decoding of objectives 123
5.2.2 Overfitting and oversimplification 124
5.2.3 Ruggedness transformation 125
5.3 The Tunable Objectives Problem (TOP) 125
5.3.1 Modeling problems with layered and summed objectives 128
5.3.2 Modified ruggedness transformation 131
5.3.3 Epistasis and objective-convolution 136
5.4 Experiments 137
5.4.1 Experimental settings 137
5.4.2 Experiments and results 140
5.4.3 Experiment 1 – Basic problem hardness with local optima 140
5.4.4 Experiment 2 – Multi-objectivization on the basic problem with local optima 150
5.4.5 Experiment 3 – The effect of objective-convolution on multi-objectivization 154
5.4.6 Experiment 4 – Breakage of multi-level epistasis 158
5.5 Chapter Conclusion 161
Chapter 6 Multi-Objectivization via Segmentation on the TSP 165
6.1 Introduction 165
6.2 Background 166
6.2.1 Multi-objectivization studies with the TSP 167
6.2.2 Principles of multi-objectivization 170
6.3 Experiments 173
6.3.1 Algorithms, settings, and operators 174
6.3.2 Experiment 1 180
6.3.3 Experiment 2 190
6.3.4 Experiment 3 199
6.4 Chapter Conclusion 208
Chapter 7 A comparison of helpers and complete decomposition 210
7.1 Related background review 210
7.2 Relating Helpers and Complete Decomposition 212
7.2.1 Definitions: 213
7.3 The Job Shop Scheduling Problem (JSSP) 221
7.4 Experiments 221
7.4.1 Experiment 1 – Helpers versus Complete Decomposition 224
7.4.2 Experiment 2 – Exploring heuristic strength and decomposition size 227
7.5 Chapter Conclusion 244
Chapter 8 Overall Summary and Conclusions 246
8.1 Summary of Work 246
8.2 Contributions 248

8.3 Further Research 248
References 251
Appendix A – Acronyms 261
Appendix B – JMP ANOVA model supporting experiment 2 of Chapter 7 263

List of Figures

Figure 1 – Decomposition swapping trend 8
Figure 2 – The Tunable Objectives Problem (TOP) model transformation process 9
Figure 3 – The bcl380 TSP instance city configuration 10
Figure 4 – A flowchart of a typical GA optimization process 14
Figure 5 – Genotype, Phenotype, and Objective Space Relationships 17
Figure 6 – Diversity Hierarchy 33
Figure 7 – Plot of F2 37
Figure 8 – Implied Fitness Landscape for Population P1 38
Figure 9 – Implied Fitness Landscape for Population P2 38
Figure 10 – Diversity in Single Point Crossover 47
Figure 11 – A Taxonomy for Parameter Settings (Eiben et al., 1999) 51
Figure 12 – Operator directness taxonomy 61
Figure 13 – A Sample SPEA2 Solution Fitness (Zitzler et al., 2001) 70
Figure 14 – A Multi-objectivization Example 80
Figure 15 – JSSP Schedule Representations 84
Figure 16 – JSSP With All Possible Machine Links 84
Figure 17 – A sample active schedule for the swv12 instance 85
Figure 18 – GT Schedule Builder Pseudo-code 88
Figure 19 – Average Best Fitness by Generation for la01 and la06 Problems 99
Figure 20 – Frequency and magnitude of improvement in best current fitness by generation 103
Figure 21 – Problems la06 and swv07 Average Best Solution by Generation 108
Figure 22 – Average front size by generation for problems la01 and la06 109
Figure 23 – The Weise model landscape transformation process 121
Figure 24 – Epistasis transformation pseudo-code for a gene 124
Figure 25 – The TOP landscape transformation process 127
Figure 26 – An example breakdown of a 20 bit optimal solution into the SO and SOG layers 129
Figure 27 – Example objective and fitness evaluation(s) for the SO and SOG layers 131
Figure 28 – Modified ruggedness transformation pseudo-code 132
Figure 29 – An illustration of the creation of various ruggedness transformations 133
Figure 30 – Number of generations to find the optimal solution 135
Figure 31 – Ruggedness transformation for various m 142
Figure 32 – Average best fitness by number of SOGs 144
Figure 33 – Average number of generations to find the optimal solution. 145
Figure 34 – Percentage of local optima overcome by number of SOGs 147
Figure 35 – Effect of UX only and UX with mutation. 148
Figure 36 – Percentage of solutions created by crossover 150
Figure 37 – Comparison of multi-objectivization on SOG local optima landscapes. 152
Figure 38 – Objective-convolution hardness for TOP with $n = 30$, $m = 30$ 155
Figure 39 – Effectiveness of various methods against several different multi-level landscapes. 158
Figure 40 – Radial segmentation of cities in a TSP 179
Figure 41 – City configuration for the pr124 instance 186
Figure 42 – City configuration for the bcl380 instance 188
Figure 43 – City configuration for the pr107 instance 190
Figure 44 – Vertical segmentation of cities in a TSP 195
Figure 45 – Average best fitness by generation and level of K for portgen-100-1 201
Figure 46 – Average best fitness by generation and level of K for portgen-200-1 201
Figure 47 – Average best fitness by generation and level of K for portgen-400-1 202
Figure 48 – Best average fitness by generation for the bcl380 instance 208

Figure 49 – Pseudo code for instance transformation to uniform job process times 230
Figure 50 – Average percentage from the best found solution for all cases 235

List of Tables

Table 1 – Drift example population 26
Table 2 – I_1 survival probabilities 27
Table 3 – CHC operators in Eshelman (1991) 63
Table 4 – CHC parameters in Eshelman (1991) 63
Table 5 – DCGA with CPSS operators in Shimodaira (1997) 64
Table 6 – DCGA with CPSS parameters in Shimodaira (1997) 64
Table 7 – DGEA operators in Ursem (2003) 65
Table 8 – DGEA parameters in Ursem (2003) 66
Table 9 – JSSP problems analyzed 91
Table 10 – Average best found solution as a percentage from the best known solution 96
Table 11 – Helper selection variability as a percentage of standard deviation 101
Table 12 – Average best found solution as percentage from the best known solution 107
Table 13 – Correlation between average front size and quality of best found solution 113
Table 14 – Average best fitness for n = 30, m = 30, $\eta = 5$, $\gamma 2 = 0$ and various levels of τ 156
Table 15 – Average best fitness for n = 30, m = 30, $\eta = 5$, $\gamma 2 = 58$ and various levels of τ 157
Table 16 – Experiment 1 - Average percentage from the optimal solution 183
Table 17 – Experiment 2 - Average percentage from the optimal solution 197
Table 18 – Parameters of MOPS 203
Table 19 – Experiment 3 - Average percentage from the optimal solution for various methods 206
Table 20 – Comparison of complete decomposition and helpers. 225
Table 21 – Standard deviation of total job process times 231
Table 22 – Standard deviation of operation process times 232
Table 23 – Factors and Levels for Experiment 2 233
Table 24 – Best Values Found in All Experimental Replications 234
Table 25 – Effects tests for the ANOVA model 236
Table 26 – Pairwise t-tests (base instance). 237
Table 27 – Pairwise t-tests (PA). 238
Table 28 – Pairwise t-tests (DM). 238
Table 29 – Pairwise t-tests (DB). 238
Table 30 – Pairwise t-tests for the interaction (DM*PA). 242
Table 31 – Pairwise t-tests for the interaction (DM*DB). 243

PREFACE

This document has been approved for public release by the United States Air Force on June 20, 2011. Case Number: 88ABW-2011-3524.

ACKNOWLEDGEMENTS

The list of friends, family, co-workers, fellow students, and professional faculty that I would like to thank for their support is enormous. Unfortunately a comprehensive list would be impossible so for those reading these acknowledgements that have not been called out specifically, know that I appreciate the efforts you have put forth on my behalf.

First and foremost my utmost gratitude is for my Lord and Savior Jesus Christ whom, without question and in more ways than I can count, has brought me through this test-of-fire by research. No words can express my profound gratitude for His death made in recompense for our sins. Thank you Mark Fagan, Dr. Cheryl Levine, Renee Cox, those volunteers at Beginning Experiences, and Rev. Lou Izor. Your gentle guidance and tough love has helped to bring me to true faith and salvation.

One person deserves a paragraph unto himself: my mentor, friend, and advisor - Dr. Frank Ciarallo. He has been an unbelievable source of inspiration and professional growth. Dr. Ciarallo has helped motivate, inspire, and unlock potential. His patience, wise direction, attention to detail, ability to meet me where I was at, and ability to focus a discussion on the core issues played critical roles in all aspects of the research. Without his assistance, a similar quality of dissertation and completion time would not have been possible. Here is my gratitude to you, Dr. Ciarallo.

I thank my family and friends that provided support through a trying time of personal issues. Your amazing love and support helped heal my heart and head. Mom and Dad – Thank you for believing in me. My sister– I appreciate your words of wisdom regarding life's troubles. Mark Fagan: thank you for your mentorship, friendship, and concern in a time of need. Chips deserves a big thank-you for providing me with an inspiration to continue on through life.

Lastly, I thank the United States Air Force for giving me a truly unique opportunity. Integral to that organization I thank those airmen (civilian, military and contractor) that have enabled this

quest for knowledge, especially committee members Dr. Raymond Hill and Dr. W. Paul Murdock.

Chapter 1 Introduction

In 2001 an optimization technique called multi-objectivization was introduced. Only a small segment of research has explored the inner workings of this new technique in the ten years since its introduction. This research provides additional understanding of multi-objectivization and its associated algorithms. The term *multi-objectivization* refers to the process of reformulating a single-objective problem into a multi-objective problem and solving it with a multi-objective method in order to provide a solution to the original single-objective problem (Knowles et al., 2001).

There are more than a handful of optimization techniques inspired by natural phenomena. A subset of these optimization techniques uses a group, or population of solutions, and a number of operators to guide the optimization process. Multi-objectivization utilizes these types of algorithms in its process. Before providing the many technical details that constitute multi-objectivization, this discussion begins with an analogy to provide a general context of how multi-objectivization works. An example of human breeders optimizing a population for specific purposes can be found at home – in man's best friend, the dog.

1.1 An analogy toward multi-objectivization

An analogy of dog breeding for hunting rabbits is used here to describe multi-objectivization. Let us assume that a rabbit hunter desires a dog that will significantly increase his effectiveness at hunting rabbits. A gun-carrying, on-foot hunter may desire certain traits in his dog. Desirable traits in such a hunting dog may be: a good sense of smell, a slow speed of travel, a high visibility profile and a clear and loud bay or bark. A good sense of smell helps the dog to track scented trails to a den or resting place. A low speed of travel helps in maintaining proper

distance between the hunter and his dog. A dog can be effectively followed by the hunter through both visual and audio cues. An audible bay or bark helps hunters if line of sight to their companion is blocked and an easily visible tail may help hunters follow their dog through thick grass or heavy underbrush.

Suppose the ideal rabbit hunting dog does not exist in a currently living animal – how do we eventually create such a dog through breeding from our current population of canines? Traits are bred in to new offspring by mating dogs with desirable features repeatedly until a great hunter is born. If breeders, for example, have a dog that tracked scents well, but the dog was too fast, the breeders would breed in shorter legs. When the legs of offspring were short enough, the breeders may then attempt to integrate the white tipped tail. This simple description underestimates some of the complexity – as a breeder brings in one trait through one or many generations another trait may suffer. As a result it can take many generations to create a completely new breed.

A similar breeding process, over many generations, has given us the modern day basset hound. The basset hound was bred for the purpose of hunting rabbits with a gun-wielding hunter following on foot. The hound has short legs for slow ground speed, a large nose for a great sense of smell, a set of large ears and loose skin that collect odors to enhance scent, a white tipped tail for visibility, and a loud, low frequency bay that can be easily heard from a great distance. Hunters may have not envisioned what the breed would look like. Instead, they focused on the traits of interest in order to select which dogs would be bred to hopefully generate desirable offspring.

Previous evolutionary optimization techniques applied to single-objective problems have not closely followed the analogy of utilizing sub-fitnesses (traits) of a solution (dog) to help guide the search. Before 2001 and the introduction of multi-objectivization, evolutionary optimization methods were only focused on the overall effectiveness of solutions, analogous to the effectiveness of a dog at catching rabbits. Unfortunately this can lead to difficulties in some

cases. For instance, back to the analogy, suppose that a population of dogs contains greyhounds and those dogs are much better than the others dogs in the population at catching rabbits. A greyhound is an effective hunter of rabbits in large open spaces due to their powerful legs and large lungs which allows them to chase a rabbit until it tires and slows. If a breeding process focused only on total hunting effectiveness in choosing dogs to mate, it would put heavy emphasis on mating greyhounds. More often than not, such a process would perfect the greyhound breed rather than generate an entirely new hunting companion. Still, occasionally, a greyhound might be mated with a shorter legged dog, leading to something closer to the basset hound. Breeding a greyhound with a shorter legged dog likely results in an ineffective hunter. The offspring of such a combination has lost the speed advantage of the greyhound but is still too fast for a hunter to follow on foot with ease. One-dimensional algorithms that only focus on overall hunting effectiveness might get 'caught' in the greyhound local optima. It's the breeder's knowledge of what sub-traits are important and compatible that guides them to breeding non-greyhounds so that a shorter and slower companion is eventually born.

1.2 Research Overview

This dissertation studies multi-objectivization and a family of genetically inspired optimization techniques called Evolutionary Algorithms (EAs). Solution diversity plays a key role in evolutionary algorithms as well as in many other optimization processes. Multi-objectivization encourages diversity by rewarding subcomponents of fitness in individuals, but it does more than this: multi-objectivization can guide the algorithm into areas of the search space that are more productive and that might otherwise be ignored. Chapter 2 summarizes a detailed investigation of the role diversity plays in evolutionary algorithms. From that study of the existing literature a new classification scheme for GAs emerges that focuses on how the components and operators in a GA effect diversity based on whether they preserve vs. promote diversity, how directly they are intended to modify diversity, in what occurrence pattern they are

used in the GA, how they use randomness, and their time order. Although this dissertation is not focused on studying new diversity-oriented techniques, the development of this classification framework serves as a useful structure to summarize many threads in the existing literature on standard GAs. Chapter 2 also reviews fundamental understandings regarding EAs including applicable theories, practices, and known ways in which evolutionary search can fail to identify good results.

Implementation of multi-objectivization can be tricky business. It is important to understand the fundamentals and techniques that underlie the processes of multi-objective optimization and multi-objectivization. Multiple-Objective Evolutionary Algorithms (MOEAs) have been used to address a significant number of problems that are naturally multi-objective (Coello Coello et al., 2007). These multi-objective problems have competing and incompatible objectives such as minimizing cost while maximizing safety in a factory line. MOEAs, unlike traditional EAs, attempt to find multiple, non-dominated solutions that approximate a Pareto frontier by using various methods of determining relative fitness of solutions. In MOEAs a solution is considered dominated by another solution if it is worse in at least one objective and no better in all other objectives. Non-dominated solutions are those solutions that are at least as good as all other solutions in all objectives. Pareto comparisons can be made between solutions with incompatible objectives to determine the set of decisions that a rational decision maker would select from (the Pareto frontier). MOEAs like the Non-dominated Sorting Genetic Algorithm version II (NSGA-II) described in Deb et al. (2002) reward solutions partially based upon Pareto comparisons.

Multi-objectivization techniques are varied and divided into two major categories: some techniques add novel objectives while others decompose the objective function itself. This dissertation focuses on the latter techniques. Multi-objectivization Via Decomposition (MVD) has been accomplished in two basic ways. In *complete decomposition* through algorithms such as Multi-Objectivization via Segmentation (MOS) the main objective is not explicitly represented.

Other methods called *helper-objectives* use decomposed parts simultaneously with the main objective in the evolutionary optimization process. A complete description of these concepts is included in Chapter 3.

The research presented in this dissertation is based on a careful study of select problems through a sequence of planned experiments. In total, two fundamental ways in which multi-objectivization improves genetically inspired search were uncovered:

The first way multi-objectivization improves the search is through identification of fitness improvements that are otherwise ignored. Suppose we ignored the possibility that a basset hound was possible and instead wanted a more effective greyhound that would hunt rabbits on its own. With no hunter-dog interaction, longer legs may be a desirable trait for a greyhound. If we simply focused on the effectiveness of the greyhounds in catching rabbits, we may not reward newborn greyhounds that have longer legs if they happen to also have an accompanying worse sense of smell than their counterparts. If a greyhound offspring gains a useful attribute, like longer legs, we may not breed this offspring if its overall hunting effectiveness went down. Focusing on good parts of a solution (legs) separately from other important attributes (nose) allows us to keep and breed the longer-legged dog in hopes that we can shift focus back to improving the sense of smell in its offspring. By isolating a specific attribute in the solution (legs) we can reward its fitness at the expense of other attributes. The improvement in fitness of a dog's legs can be thought of as a useful signal, while concurrent fitness decrements can be thought of as detrimental noise. Multi-objectivization allows us to improve the effectiveness of search through the isolation of useful signals. Improving *signal to noise ratios* is the first fundamental way in which multi-objectivization improves the search process. This concept is identified in Chapter 5 through the creation and study of a class of new problems generated from the Tunable Objectives Problem (TOP) model. In Chapter 6 we used the signal to noise principle to generate decompositions with

improved performance. In Chapter 7 we discuss the trade-offs between improving signal to noise ratios and providing explicit focus on the main objective of a problem.

The second way in which multi-objectivization improves the search process is by not allowing the search to be confused by confounding interactions in solutions. Suppose that there were no breeds similar to the basset hound in our current population but that the basset hound is a possible outcome of many iterations of breeding. Generating such an animal, therefore, requires multiple generations. Say the shorter legged dogs in your population have relatively weak noses and small lungs but they can be easily followed. Unfortunately their weak nose makes them poor trackers. If the larger dogs in your population have strong lungs, a great sense of smell but long legs they are also poor hunting partners – they must hunt independently to be effective. First order combinations that "average" the two polar opposites in these traits result in dogs that are poor in all categories. Medium legs are too short to run-down a rabbit, but too long to be easily followed by a hunter. A moderate sense of smell makes a dog a moderate tracker. A medium lung capacity makes the dog tire more quickly than the larger dogs. However, suppose we focus on finding good solutions with various land speeds while still examining hunting effectiveness. The trade-offs between these two attributes would be important, but we place value in short legs. After some amount of breeding, we gain a broader variety in the population by rewarding shorter legs. This makes it easier to overcome the local optima that prevent short legs to be paired effectively with a good nose and large lungs. Over the course of generations, by focusing for some number of generations on key attributes in addition to total hunting effectiveness, a breeder can avoid (break) the impact of interactions that would otherwise 'hold-back' the breeding process. We call this process of avoiding problem interactions as the *breakage of epistasis*. This concept is explored and discussed in Chapters 4 and 5. The act of switching decompositions appears to break epistasis as indicated in Chapter 4 and previewed below. In Chapter 5 objective conflict is directly controlled at multiple layers in the problem in order to determine the

relationships between objective conflict and the breakage of epistasis through multi-objectivization. An experiment clearly shows how various layered levels of epistasis can be avoided through multi-objectivization which results in improved search performance.

Knowledge of the two ways in which multi-objectivization improves the search process is insufficient to implement multi-objectivization in practice. Large combinatorial optimization problems are unfortunately much more complex than the dog breeding example. It becomes difficult to discern what is a "good trait" and as a result the accompanying study of multi-objectivization is more complex. There are many decisions to make in designing and implementing the optimization process, but a few guiding principles have been established in the literature prior to this work. Instead most prior research focuses on empirical and analytic evidence that multi-objectivization can provide better results on select problems. In Chapters 4 – 7 we explore and identify principles governing choices about multi-objectivization. Several of these principles are related to the definition of decompositions. From the research in this dissertation we learn that decompositions:

- should be sequenced according to their contribution to fitness (importance) (Chapter 4),
- should be selected by their likelihood to be unhampered by local optima that are present in the main objective (Chapter 5),
- should be selected based on improving SNRs that are internal to the problem (Chapter 5),
- should be selected such that concurrent helpers have complementary properties (Chapter 5),
- should be selected for their alignment with the main objective (Chapter 5), and
- should be sized according to conflict levels, strength of heuristics, and critical minimum size (Chapter 7).

Furthermore, evidence uncovered here indicates that switching decompositions within one run of an algorithm is an important remedy to the negative effects of non-linear interactions within a problem's fitness. For example in the work in Chapter 4 on the classic ft10 scheduling problem instance using helper-objectives, it was observed that at every 20 generations in Figure 1 where a decomposition switch takes place, the frequency with which a GA finds better solutions increases. This result is described in more detail in Chapter 4. In addition to this evidence that

decomposition swapping is important, it may be difficult to identify good decompositions in NP hard problems, thus multi-objectivization often uses different decompositions as the algorithm progresses.

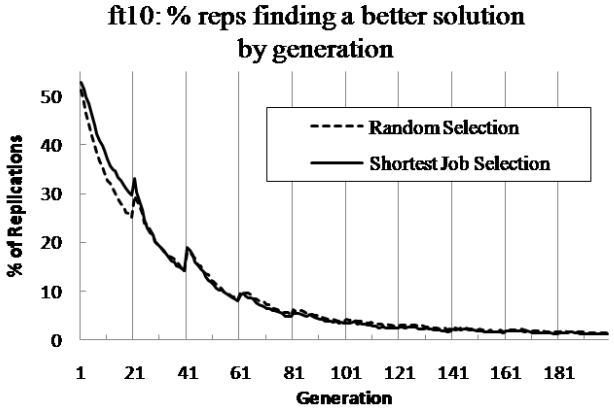


Figure 1 – Decomposition swapping trend

Because NP hard problems are complex, the study of methods in that arena can be difficult. As a result in addition to study on NP hard problems we extended, defined, and studied an abstract problem called the Tunable Objectives Problem (TOP) in Chapter 5. TOP allows a researcher to introduce various effects into a problem in order to study how algorithms resolve such effects. This abstract and general problem can be studied by the broad category of integer programming techniques including but not limited to evolutionary approaches. TOP introduces individual objectives in a layered approach, includes local optima, and objective conflict. Figure 2 shows the TOP transformation process. Processes in grey are inherited from a previous model. Traditional single-objective optimization and multi-objectivization techniques were studied using the TOP model in order to generate direct comparisons of the methods. From the study of the TOP model, many of the aforementioned principles regarding decomposition approaches were revealed.

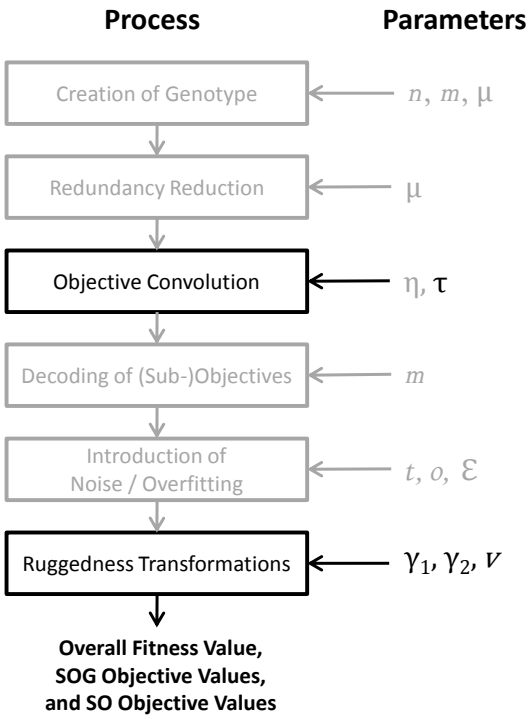


Figure 2 – The Tunable Objectives Problem (TOP) model transformation process

Armed with many principles for how to generate effective decompositions, we revisit the Traveling Salesman Problem (TSP) in Chapter 6. The TSP was studied previously by three different multi-objectivization research efforts using three different methods. Multi-Objectivization via Segmentation (MOS) was the most recently proposed algorithm. When MOS was introduced it was compared with helper-objectives and shown to give better results, on average, for the problems studied but the difference between the methods was relatively small and there were more than a few cases where helpers were better than MOS. Some of the insight for MOS's improved performance, provided in Chapter 6, was not revealed in the original introduction of MOS.

In the context of MOS, we study various decompositions for the TSP in Chapter 6. New and previous decompositions are studied and divided into two major categories: those decompositions that attempt to distinguish between cities that are within neighborhoods and cities that are between neighborhoods, and those decompositions that attempt to divide a number of self-

contained neighborhoods. For instance, a new decomposition called Nearest Neighbor Ratio of Distances (NNROD) distinguishes between cities that are within- and cities that are between-neighborhoods by using a density estimate to define a neighborhood. This method is shown to perform better than the previous method proposed with MOS. Additionally, spatial decompositions are prescribed for problems with strong spatial orientations. Radial and vertical segmentations were studied. Figure 3 shows a TSP instance in which vertical segmentation was particularly effective due to the isolation of signal (fitness improvements). Finally in Chapter 6 we also identify how signal and noise relate over time in a typical evolutionary optimization process. It was noted that signal generally gets scarcer as the optimization process continues and noise rises proportionately. From this knowledge we develop a more general method called Multi-Objectivization via Progressive Segmentation (MOPS). A single objective genetic algorithm and MOS are both special cases of MOPS. MOPS is shown to provide robust performance across a variety of experiments.

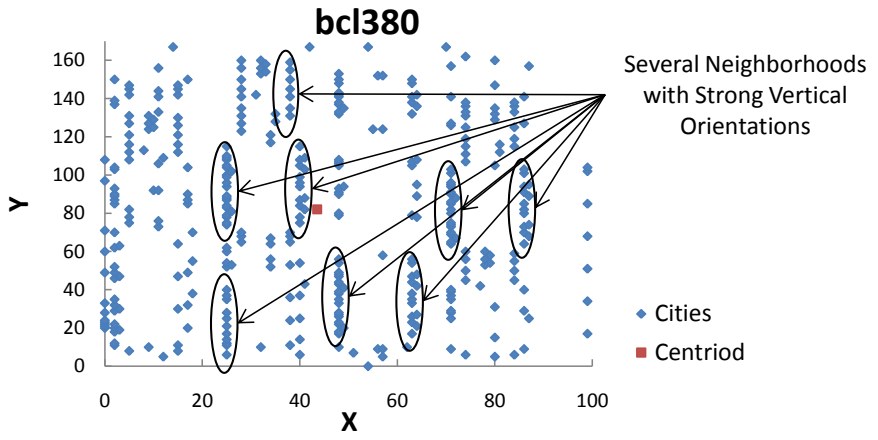


Figure 3 – The bcl380 TSP instance city configuration

Lastly in Chapter 7 we delve into the difference between helper-objectives and MOS type decompositions (complete decompositions). Through a formal analysis it is shown how solutions in one method map to relative fitness of solutions in another method. Postulates about relative solution fitness are derived from a careful study of Pareto dominance relationships. These

postulates are used to support a theorem that establishes that solutions in helper decompositions that are non-dominated are also non-dominated in complete decompositions. In addition to analysis comparing helper-objectives and complete decomposition, the chapter studies these methods empirically in the context of job-shop scheduling. We conclude that complete decomposition is generally better for balanced decompositions and no worse than helpers for imbalanced decompositions. Also in this chapter we explore the relationship between heuristic strength, minimal decomposition size, and the focus of the decomposition. From this examination three principles governing decomposition choice are uncovered: focus through balance, focus by order of appearance, and adequate coverage.

1.3 Contributions to date

The following are the major research contributions of this effort:

- Identified numerous principles of multi-objectivization
- Revealed two major methods in which multi-objectivization gains improved results
- Extended and developed an abstract problem instance for testing of optimization techniques
- Developed a new method for generating instances of the JSSP
- Identified better decomposition techniques for use on the JSSP and TSP
- Developed a robust multi-objectivization technique
- Developed a diversity based classification scheme for GAs

1.4 Publications

1.4.1 Peer review journal articles

Lochtefeld D.F., F.W. Ciarallo. *Helper-Objective Optimization Strategies for the Job-Shop Scheduling Problem.* Journal of Applied Soft Computing. September 2011. 11 (6): 4161-4174

Lochtefeld D.F., F.W. Ciarallo. *Multi-objectivization via Helper-objectives with the Tunable Objectives Problem (TOP).* IEEE Transactions on Evolutionary Computation, To Appear. 2011

Lochtefeld D.F., F.W. Ciarallo. *An Analysis of Decomposition Approaches in Multi-objectivization via Segmentation*. Journal of Applied Soft Computing. In review. Submitted April 2011

Lochtefeld D.F., F.W. Ciarallo. *Multi-objectivization via Decomposition: A Comparison of Helper-Objectives to Complete Decomposition*. Journal of Soft Computing. In review. Submitted May 2011

1.4.2 Peer review conference proceedings

Lochtefeld D.F., F.W. Ciarallo. “Deterministic Helper-Objective Sequences Applied to Job-Shop Scheduling”. *GECCO '10: Proceedings of the 12th Annual conference on genetic and evolutionary computation.* Portland, OR. July 2010

Lochtefeld D.F., F.W. Ciarallo. “A Diversity Classification Scheme for Genetic Algorithms”. *IERC 2011: Proceedings of the 61st Industrial Engineering Research Conference.* Reno, NV. May 2011

Chapter 2 Diversity in Genetic Algorithms (GAs)

Genetic Algorithms (GAs) are one of several major types of meta-heuristics that are used to address hard optimization problems (De Jong and Spears, 1989). As reflected by a large volume of literature, GAs have been applied to a significant number of real world problems. The GA approach has a number of strengths as well designed GAs are robust and efficient at searching large problem spaces. GAs are based on the idea that the *recombination* of solutions (*individuals*) can potentially find new and better solutions. GAs are global search techniques as recombination of solutions allows the algorithm to visit any area in a solution landscape without local traversal. These algorithms are stochastic and use random processes as integral parts to their search. Random draws are often used in the initial population generation, in the selection of parent solutions, in the mating of selected solutions, and in the mutation of solutions. GAs use the concept of a collection of individual solutions, or a *population*. From the population some solutions are selected to create offspring solutions. In GAs, not all offspring survive, thus the algorithm uses the concept of survival-of-the-fittest to help determine the portions of the problem space that the GA searches most intensively. In most implementations, a GA adapts from a diverse search method into one that intensifies search within permutations of the current population. GAs can elegantly transition from global search (*exploration*) to more local searching (*exploitation*). Since GAs are stochastic, they are often re-initialized multiple times with new starting conditions in an attempt to search a different part of the solution space. A flowchart of a typical GA optimization process is shown in Figure 4. For a complete description of genetic algorithm structure and properties see Chapter 3 of Eiben and Smith (2003).

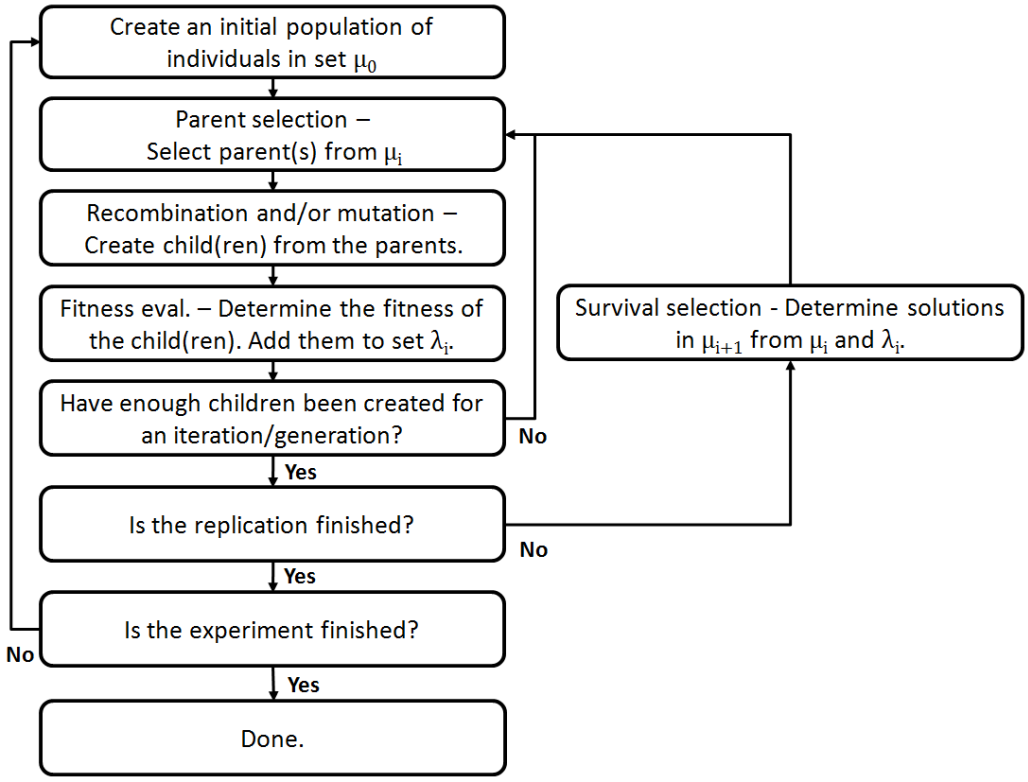


Figure 4 – A flowchart of a typical GA optimization process

Since GAs operate by maintaining a population of multiple solutions, it is important to have solution diversity in order to search varied portions of the problem landscape. Much of past work on GAs has been on defining new operators and control mechanisms. Two important questions exist for nearly any GA method. First, how does the method influence the nature of the fitness landscape or the way in which the method traverses the landscape? And second, how does the method utilize, create, and/or preserve diversity? Understanding and predicting the answer to the first question is difficult as the analyst does not know the true nature of the problem. If everything about the problem was known, the analyst could select appropriate operators to traverse it. However, if everything in the problem is known, there is no need to use a GA to find a good solution. The uncertainty of the characteristics of the problem causes difficulty in answering the first question and as a result, much focus has been placed on the diversity question. The answer to the diversity question is also elusive because there are many definitions to diversity and the design of GAs is highly non-linear – a design change in one area may have significant impacts to

the diversity implications of other operators in the GA. However, different GAs exhibit varying ranges of diversity and therefore it can still be useful to discuss the level of diversity that they use and maintain. The following sections describe properties and fundamentals regarding GAs in general. Later sections in this chapter specifically discuss how these characteristics relate to the population diversity in a particular method. Lastly a diversity classification framework is proposed to provide clarity for how different GAs manage diversity.

2.1 Select GA Concepts

Several select GA concepts follow. These concepts are by no means an exhaustive and complete examination of GAs. However, the particular concepts are reviewed to provide some foundation for further discussion.

2.1.1 Phenotype space, genotype space and the concept of a problem landscape

Individuals in GAs contain a string of values that map-to or directly-represent the decision variables of the problem. Decision variables are frequently encoded in some fashion in order to be represented in the model. The origins of GAs used a binary representation; a set of 1s and 0s. The encoded representation of a solution is the *chromosome* of an individual. All feasible values for the chromosome make up the *genotype space*.

Decision variables are the *genes* of a solution. *Alleles* are the encoded components of genes. For example a binary solution of 1,0,0,1,0,0,1,0 could map to two decision variables x_1 and x_2 where x_1 is the first 4 bits of the solution and x_2 is the last 4 bits. In this example, x_1 is represented by the bits 1,0,0,1 and could have a value of 9 while x_2 is represented by the bits 0,0,1,0 and could have a value of 2. The alleles of the solution are the individual 1 or 0 values while the genes of the solution are *x1* and *x2*. Early GA research encoded solutions into a set of binary strings representing each decision variable (Eiben and Smith 2003). To perform an optimization of two real valued decision variables, a binary string would be generated for each of the two decision

variables. Since early work encoded solutions to a binary string, problem details were frequently not embedded in the GA. Contrary to their origins, modern GAs use a wide variety of encodings. Many of these encodings are not binary strings and the chosen operators are typically specific to the encoding of the solution and the problem details (Eiben and Smith 2003).

Unlike the genotype space, the *phenotype space* is not solution-form dependent. The phenotype space contains the possible solutions to the problem. In the above example, all values of x_1 and x_2 make up the phenotype space. Depending on encoding, a given genotype may or may not represent all solutions that fully cover the phenotype space.

Individual solutions have a single objective associated in deterministic single objective optimization. The *objective space* contains the overall fitness values for each possible solution. With each solution having a single objective value, the concept of a *problem or objective landscape* can be useful when describing search strategies. Problem landscapes provide a mapping of the decision variables to the objective space. Problem landscapes can be thought of as having a variety of features such as high peaks and valleys, broad flatter areas, areas of gradual increase or decrease, and plateaus. Most problems have more than two variables and cannot be visually examined with ease. The concept of a problem landscape can still be useful in understanding discussion of neighborhoods and global versus local search. For each solution there is an associated objective value but this value may not be the value that the GA uses to determine solution fitness. This objective value can be subsequently modified for handling constraints, for promotion of diversity, or for other purposes. The value the algorithm uses to determine solution fitness is the *fitness value*. Fitness value may or may not directly correspond to the objective value.

This mapping of genotype space to phenotype space to objective space can be observed in Figure 5 which was inspired from a similar figure presented by Weise et al. (2008).

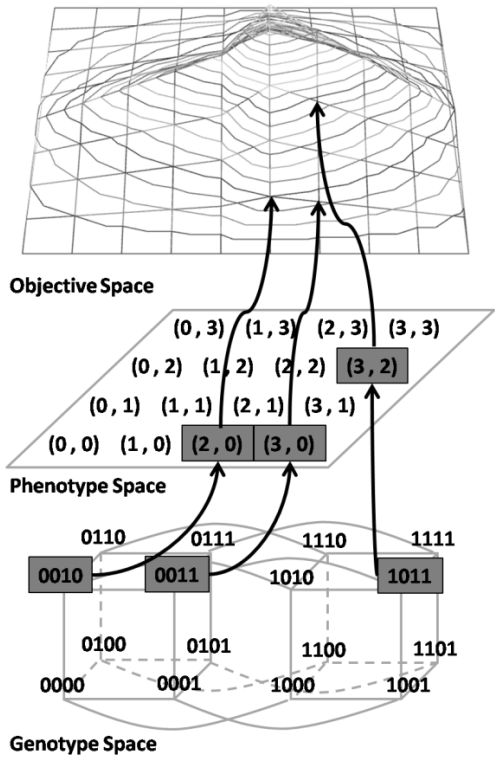


Figure 5 – Genotype, Phenotype, and Objective Space Relationships

2.1.2 The schema theorem

John Holland's schema theorem was published in the first textbook on GAs in 1975 (Holland, 1992). The *schema theorem* described a GA as a method that samples large portions of the problem space called *hyperplanes*. Large hyperplanes are large subsets of the set of all solutions. *Schemata* describe the hyperplanes by enumerating which variables are converged (fixed) in the subset and which variables are not yet converged. Schemata are similar to binary encoded solutions but also introduce the concept of a wildcard in order to describe a set of solutions rather than a single solution. An individual encoded as a binary string is specified by a string of 1's and 0's describing each allele. In contrast each allele in a schema has the option of being a 0, 1, or wildcard (*). If *all* solutions in a set contain a single value for an allele, the schema for that set contains that value. Conversely, if *any* solutions in the set contain different values for an allele, the schema contains a *. Thus the schema **0***1* describes a solution set

with eight bits in each solution. The third bit in all of the solutions in the set has a value of 0 and the seventh bit in all solutions has a value of 1; this schema is the hyperplane in the third and seventh bit. No information about the relative density of 0's and 1's in the * positions is given other than stating that there is a mix 0's and 1's in each of the * alleles in the set. The number of fixed numerical positions in a schema determines the order of the schema. Thus an eight bit schema represented as *001**** is an order-3 schema. Holland's schema theorem essentially states that GAs sample hyperplanes (schema) in proportion to the representation of that schema in the population and the average fitness of the solutions sampled in that schema (Goldberg, 1989c). The schema theorem is proven only for a single generational change.

2.1.3 The building block hypothesis

The *building block hypothesis* states that "better and better strings [are constructed from] the best partial solutions of past samples" (Goldberg, 1989c) . In other words, lower order schemata are combined together to form intermediate solutions with better component parts. These intermediate solutions are combined over and over to created better and better intermediate solutions until the algorithm eventually converges to a final solution that is represented by each individual in the population. The building block hypothesis implies that for GAs to be effective, solutions must have some complementary components (synergies) in order to be combined into better solutions. Most traditional GA approaches to single optimization problems use overall fitness in survival and parent selection. Solutions are not are not selected based upon the fitness of their component parts leading to the question: when is it appropriate to select solutions based on their parts rather than the sum of the parts? The building block hypothesis places emphasis on the notion that GAs work best by combining solutions together. This emphasizes a difference between GAs and Evolutionary Strategies (ES). GAs generally use crossover as the primary method of creating new solutions while ES generally places more emphasis on mutation (Bäck, 1996).

2.1.4 Exploration / exploitation balance

Exploration/Exploitation Balance (EEB) is the degree to which the GA searches in areas far from current solutions or close to current solutions (Maturana and Saubion, 2008). Exploitation can be viewed as local search around a given solution. Exploitation of solutions generally quickens convergence as it produces solutions that are very alike existing solutions. These new solutions compete with the old solutions and as a result exploitation generally pressures the population toward homogeneity by driving out worse than average solutions quickly. Exploration can be viewed as more global search and can help maintain diversity by including very different solutions into the search. However, pure exploration selection pressure boils down to completely random search. It is important to strike a balance in the search profile to assure both sufficient searching of the problem landscape and sufficient searching of the promising areas in that landscape. Generally exploitation happens after exploration in a GA because the areas of the problem space that appear promising must be found before the GA can attempt to exploit them. Both processes of exploration and exploitation happen concurrently for most GAs. It is important to match the specific problem with the strategy and parameters used to solve it. The concept of an *EEB profile* can be helpful since there are always limited amounts of time and computing resources to answer a problem.

Definition of the terms exploitation and exploration are linked directly to the definition of a neighborhood. When a new search point is within an already identified neighborhood, the algorithm is exploiting (solutions). If the new search point is outside of the known neighborhoods, the algorithm is exploring. According to (Random House, 1997), the word explore is defined as: “to traverse … over (a region, area, etc.) for the purpose of discovery.” Discovery is a key word: something new must be found for exploration to happen. Since most GAs do not keep a history of places visited (Michalewicz and Fogel, 2000), exploration is often inferred as a direct function of the current population rather than a function of all the places

visited. However, to be truly discovering, the GA must visit new areas of the search space not points that were already searched. Focusing entirely on aspects of exploration and exploitation can be misleading (Eiben and Schippers, 1998). This is due to two factors: the EEB focuses on only a crisp definition of neighborhoods. Many definitions of exploitation and exploration are possible due to the ambiguous definition of neighborhoods. Focusing on optimization of a single EEB can be misleading since many definitions of a neighborhood implies that there are more than one possible EEB to a problem. A single EEB only accounts for one viewpoint on the way a GA searches. It is more empowering to focus on those design decisions that result in robust search rather than those that optimize a single EEB.

2.1.5 Neighborhoods and the measure of distance

Maintaining diversity in a GA seems like a noble cause, but diversity of what? How diversity is defined is based heavily on how neighborhoods are defined within the solution landscape. There are many different ways in which we can define neighborhoods (Michalewicz and Fogel, 2000). One could define diversity based on a set of distances in the search landscape (phenotypic or genotypic) or as a set of distances in terms of objective values. Friedrich et al. (2007) compared these two metrics and concluded that both have implied impacts on runtime performance. Most diversity literature focuses on diversity measured in the phenotypic search space. This is likely due to natural definitions for distance in that space and our intuitive thinking of the search space as a landscape. Since the definition of a neighborhood is problem dependent, this document does not focus on the definition of what is inside- or outside-of neighborhoods. Emphasis will not be placed on an explicit definition for the terms of exploration, exploitation diversification, and intensification since these terms depend on the definition of a neighborhood. However, since there are number of properties that degrade GA performance, it is still useful to utilize these terms when attempting to understand of how different algorithms perform.

2.1.6 Selection pressure

Parent selection pressure involves one of two elements that determine overall *selection pressure*. A GA needs a way to determine which solutions should pass genetic material onto children. The operator that decides which parents to mate is commonly referred to as *parent selection*. Three methods of parent selection are most common: tournament selection, roulette wheel selection, and rank-based selection. Tournament selection competes two or more random individuals and the fittest of those individuals goes on to be a parent. Tournament selection is likely the simplest method because it only requires local decisions regarding fitness. In roulette wheel selection all potential parents compete with each other based upon their relative fitness proportions. In rank-based selection, individuals are ranked from best to worst in fitness and then compete based upon their rank in the population. Each method has different amounts of pressure for the top solutions to pass on genetic material. In Roulette wheel selection the amount of parent selection pressure is based upon the current population's fitnesses. Tournament parent selection pressure differs based on the tournament size. Parent selection pressure for rank-based methods depends on the proportion of times higher rank individuals are selected as parents and can be scaled up or down based on a parameter.

The second element that determines selection pressure is survival selection. *Survival selection* determines which individuals survive and which do not. Two major types of survival selection strategies are generational and steady state. GAs using steady state survival selection are also commonly referred to as Steady State Genetic Algorithms (SSGAs). Steady state survival selection makes a decision about how to survive individuals in the population as each new individual is created. In contrast, generational survival selection creates a number of individuals in a batch or *generation* and then determines survival on the individuals in the parent generation and in the newly created children generation after the set of children have been created. Conventionally, the set of parents is denoted by λ and the set of children is denoted by μ. A $(\lambda+\mu)$

survival strategy considers both the parents and children when determining survival. On the other hand a (λ, μ) strategy replaces the parent set λ with the child set μ. When using the (λ, μ) survival strategy, GAs often employ *elitism* to survive one or more of the best solutions from the parent set into the child set. This ensures that the best found solution is not 'forgotten' by the optimization. A (λ+1) strategy describes the SSGA and a (1+1) strategy describes an evolutionary hill climber. Different survival selection strategies have differing survival selection pressure. Generally steady state approaches have higher survival selection pressure than generational methods. Together with parent selection pressure, survival selection pressure determines overall selection pressure for a GA.

The combination of parent selection and survival selection determines *selection pressure*. The selection pressure concept is one of several aspects that can make a GA converge fast or slow. Before going further into many of the interactions that can affect diversity and convergence, it is useful to discuss the effects that can degrade performance of a GA.

2.1.7 Properties / processes that can degrade GA performance

There are several properties and processes that can degrade GA performance. Many places in the literature describe how premature convergence is a problem for genetic algorithms (Maturana and Saubion, 2008, Mahfoud, 1995, Bhattacharya, 2004, Shimodaira, 1997, Oppacher and Wineberg, 1999, Smith et al., 1993). Premature convergence is the degeneration of the GA search where the population becomes dominated by one or more suboptimal solutions. Premature convergence happens as a result of a variety of negative effects. Since the GA is stochastic, *genetic drift*, processes due to random sampling, can cause early convergence. *Operator bias* may also cause early convergence by forcing the search into confined or similar areas. *Deception* in a GA can cause high selection pressure that may force the GA to converge on a sub-optimal solution. *Selection pressure* can cause the phenomena of *hitchhiking* whereby poor alleles are replicated in the population because they are attached to a solution with above average fitness.

Interactions between genes or alleles known as *epistasis* can cause the GA to make sub-optimal decisions. Lastly, duplication can be a negative force on GAs as multiple copies of a solution tend to 'push' the algorithm toward the solution with the higher frequency. The concepts of deception, epistasis, drift, duplication, hitchhiking, and operator bias are discussed in the following sections.

2.1.7.1 Deception

Loosely defined, deception is a property of a problem landscape that leads the GA to proceed as if a given area for searching is bad when in fact that area contains good solutions. Deception is one way in which GAs can be fooled into convergence to suboptimal solutions. Deception occurs when selection pressure leads the GA away from optimal solutions (lower order schema do not lead the algorithm to optimal points but instead lead away from them) (Goldberg, 1989a, Goldberg, 1989b). In Goldberg (1989a) the concept of deception was introduced and the definition of a complete deceptive problem was demonstrated using Walsh polynomials. Complete deceptive problems contain schemata that always lead away from optimal points. Whitley (1991) proved that deceptive problems must be multimodal and theorized that only hard problems were deceptive. However, Grefenstette (1993) concluded that deception is not the only condition that makes a problem hard for GAs. Grefenstette (1993) also demonstrates that deception can be viewed as a dynamic phenomenon. An example shows how a GA can be deceived (for a while) and still reliably find the optimal solution.

Diversity management has the potential to naturally combat the effects of deception. The longer 'bad' alleles survive, the more likely the GA will 'stumble' into the solution. Thus, GAs with increased diversity are generally less 'fooled' by deception. Diversity management, by itself, is not a complete solution to deceptive problems but it can help combat the forces of deception by slowing convergence.

2.1.7.2 Epistasis

Epistasis is the interaction between genes or alleles of a solution. Epistasis has been proposed as an alternate viewpoint to deception for what makes problems hard for a GA. However, the viewpoints between deception and epistasis are not entirely distinct as functions that are deceptive by definition must have some component interactions. Unlike the viewpoint of deception, epistasis does not require that lower order schemata lead away from the optimal solution. A needle-in-a-haystack problem where all but one point in the landscape have sub-optimal values can be difficult for a GA even though lower order schemata do not lead away from the optimal as the deception argument requires. Instead in this case, lower order schemata lead nowhere since lower order schemata do not indicate a direction of increasing fitness. Problems that have the highest levels of epistasis do not contain regularities in the search space and as a result heuristics are no better (and often worse) than non-repeating random search and problems with little epistasis are generally simple making them solvable with a hill-climber (Davidor, 1991). Deception is a special case of epistasis (Beasley et al., 1993) and epistasis is known to be sensitive to the solution representation. A problem studied with a GA may be difficult in one representation and considerably easier in another.

Several methods of problem sampling have been proposed to date to determine levels of epistasis in a problem through problem landscape analysis. Davidor's measure of epistasis variance was the first attempt (Davidor, 1991) but the measure did not account for possible scaling in fitness and the number of negative and positive interactions (Reeves and Wright, 1995). Measures of normalized epistasis were proposed that addressed possible bias from scaling the magnitude of fitness values (Naudts et al., 2000,Van Hove and Verschoren, 1994,Suys and Verschoren, 1996). Reeves and Wright (1995) proposed a Design of Experiments (DOE) approach using contrasts. This approach breaks the interactions into different groups that alias a number of effects together. The DOE approach is attractive because both magnitude of change

and direction of change can be accounted for in the analysis. However, like all the other methods to date, the DOE approach requires assumptions about the significance of different interactions in order to provide meaningful insight since effects must be aliased together in a incomplete sampling of the problem space. The problem of measures of epistasis based upon sampling the problem space is that if we knew what makes the problem hard, we could isolate it and thus make the problem easy. However, most GAs work on extremely large problems where a significant sampling of the space is time prohibitive. Determination of when a problem is well suited for a GA is likely better done based upon our past experience with similar problems and any knowledge that can be gleaned from input data than it is from objective space sampling.

2.1.7.3 Genetic drift

Genetic drift is the process of accumulating stochastic 'errors' that result in premature convergence on a suboptimal solution. Eiben and Smith (2003) describes a simple example to demonstrate drift: a GA with population of individuals evenly split in half between two solutions. Both of the two solutions have equal fitness. Without considering the effects of mutation and crossover, the example shows how the population must converge to either solution due to drift error through an argument based on probability. Genetic drift can cause a GA to 'melt-down' to suboptimal solutions as shown in the following example.

Suppose there was a population of individuals shown in Table 1 managed by a GA. For this example the GA uses uniform crossover (UX), roulette wheel selection, and no mutation. UX does a uniform random draw for each allele, selecting the appropriate bit from each parent with a fifty percent probability. Define PA_i as a variable holding the individual that is the i^{th} parent where $i \in 1, 2$. Since the total fitness of the population is 910, the probability of selecting I_1 as the first parent can be calculated: $P(PA_1 = I_1) = 100/910$ or ~11.0%.

The probability of selecting I_1 as the second parent (PA_2) given that I_1 was not selected as PA_1 is 100/820 or ~12.2%. The probability of selecting I_1 for PA_2 is higher because the

previously selected parent, one of the fitness 90 solutions, is not considered for the PA_2 roulette wheel selection. The overall odds of I_1 being selected as either parent is: $P(PA_1 = I_1) + P(PA_1 != I_1)*P(PA_2 = I_1 \mid PA_1 != I_1)$.

Therefore, the probability of selecting I_1 as either parent in a given mating is 11.0 %+ (100%-11.0%)*12.2% = 21.8%.

Table 1 – Drift example population

Individual (I)	Genotype	Fitness
1	0 1 0 1	100
2	0 1 1 1	90
3	0 1 1 1	90
4	0 1 1 1	90
5	0 1 1 1	90
6	0 1 1 1	90
7	0 1 1 1	90
8	0 1 1 1	90
9	0 1 1 1	90
10	0 1 1 1	90

Given the generational approach (λ,μ) where ten new offspring are created and replace the previous generation, we can calculate the overall odds of I_1 passing genetic material to the next generation. The probability for each number of matings (*ma*) for I_1 is calculated using the binomial distribution where *p* equals 21.8%, m equals 10 and n equals *ma*. Each mating is independent of prior matings. The chance of passing I_1's genetic material to the next generation can also be calculated. In a given mating with I_1 and any other population member there is a 50% chance that any allele is selected from I_1; but since there is only one allele distinguishing the difference between I_1 and the rest of the population in a given mating, there is a 50% chance I_1 will pass on its critical genetic material. The chance of I_1 passing on no genetic material for multiple matings is one minus the value from the binomial distribution where *p* equals 0.5, *n* equals 0, and *m* equals *ma.* These values are depicted in Table 2.

Table 2 – I_1 survival probabilities

Number of I_1 Matings (*ma*)	P(*ma*)	Probability of passing no genetic material from I_1 given *ma* matings (A)
0	0.08504	1
1	0.23768	0.5
2	0.298934	0.25
3	0.2228	0.125
4	0.108974	0.0625
5	0.036549	0.03125
6	0.008513	0.015625
7	0.00136	0.007813
8	0.000142	0.003906
9	8.85E-06	0.001953
10	2.47E-07	0.000977

Summing the product of P(*ma*) and A for all *ma* gives the total probability of I_1 not surviving. This probability is .315 for the above example. This example shows that even when there is a significant difference in fitness between the best solution and the population, there may be a fair probability that a converged population causes the algorithm to 'melt down' the hill. As the population size gets larger and fitness values more similar, the force of drift can be even more significant than shown here. This example is one argument for using an elitist survival strategy. However, drift effects can happen on sub-elite solutions as well so drift is not prevented simply by using an elitist strategy. This example shows that the population can be thought of as having properties of inertia where a converged ('at rest') population can tend to stay in a converged state. Populations having properties of inertia implies early influence on the search can be important as it gets more difficult to affect bias later in the search.

Genetic drift can be combated by managing the diversity of a population. Diversity in a population slows genetic drift rates because drift is partially a function of population similarity. It takes more steps for a GA to drift to a single solution if there is great diversity since one solution must force out all of the other solutions from the population to cause complete convergence. If the population is very similar, the GA may need to replicate only a few solutions to completely

converge. Drift rates are a function of population size since it takes more steps to cause convergence in a larger population. GAs need to have more selection pressure than drift pressure in order to prevent convergence to an arbitrary solution. Gibbs et al. (2008) proposed a way to calculate population size for real-based problems in order to assure selection pressure is greater than genetic drift given a set of assumptions and operators. However, this method is highly restricted to the problem representation (real values) and associated operators. Much discussion has been given to genetic drift although little research has been done to characterize drift rates when combined with selection pressure in a GA. It is hard to distinguish which convergence in a particular run or set of runs is due to drift and which convergence is due to selection pressure making online measurement of drift rates difficult.

2.1.7.4 Duplication

Because most genetic algorithms do not check that solutions are unique, duplication of solutions within a population occurs. Population convergence is the extreme case of duplication where every individual in a population is identical. Lesser degrees of duplication happen prior to convergence. Duplication can be costly because it increases drift pressure and because repeated function evaluations occur for copies of the same individual. Solution duplication can waste a significant amount of computing resources. However, ensuring solution uniqueness in a GA is also costly. Evaluating a sample of ten million unique solutions takes ten-million function evaluations, but checking that those ten-million solutions are, in fact, collectively unique takes at least seventy-million comparisons (10,000,000*log(10,000,000)). Since the problem landscape is normally much larger than the total number of sampled points, most GAs do not check for complete diversity of all individuals within a run. In a problem space with 2^{50} possible solutions, the sample of ten million points corresponds to only one-millionth-of-one-percent of the solution space. The rate of duplication of individuals for a GA is a function of the diversity properties of the GA, the population size, and the proportion of sampled points compared to the solution

landscape size. A typical GA does not explicitly ensure that diversity is maintained within the population (Michalewicz and Fogel, 2000). One might conjecture that with a solution landscape that is large, GAs would not commonly repeat solutions, but this view does not consider the practical effects of drift and selection pressure which can quickly eliminate significant diversity from a population. Furthermore, preventing duplication due to lowering drift and selection pressure by using a very large population size leads to significant negative effects; very large population size means to fewer generations are created within a constrained computational budget and thus less survival-of-the-fittest gets modeled by the GA.

2.1.7.5 Hitchhiking

In an attempt to show how GAs use building blocks to generate better solutions, Forrest and Mitchell (1993) created a problem set called the *Royal Road* where GAs were hypothesized to outperform hill climbers on a simple structured problem. The results were counter-intuitive to their hypothesis and showed that the GA performed significantly worse than the hill climber on the *Royal Road* problem. The article provided evidence that high fitness solutions (although not optimal) quickly dominated the results of the runs despite the fact that parent selection pressures were reduced. Generally, more fit solutions get a higher chance of being parents so reduction in parent selection pressure causes less bias toward selecting parents with high fitness. This quick dominance of single solutions caused problems for the GA in constructing the overall optimal solution as diversity was driven out of the population. The single solution that dominated the population carried with it a series of suboptimal bits. Because a single solution was able to take over the population, the suboptimal bits associated with that solution also became predominate even though the bits did not contribute positively to the fitness of the solution. The article referred to this phenomenon as *hitchhiking*. Formally hitchhiking is the process of poor alleles being represented in many individuals in a population because they are associated with a highly fit solution. No amount of hitchhiking is useful, but some amount is unavoidable as, by definition,

sub-optimal solutions must always contain a component part that is sub-optimal. Hitchhiking can be reduced by diversity methods because when similar solutions are not allowed to dominate the population, hitchhiking is reduced.

2.1.7.6 Operator bias

Selected parents must be combined in some way in order to create offspring. Ideally GAs could determine the 'best' traits of the parents and give them to children. However, this is rarely the actual case as there are a vast number of ways to combine two solutions. This has lead to a large body of literature studying various recombination operators for different representations and problems. In GAs, recombination is also commonly described as *crossover*.

Recombination operators are known to have bias related to how they select bits from different parents (Eshelman et al., 1989). Positional bias relates to how bits that are close together in the genotype of a given solution are likely to stay close together (Eiben and Smith, 2003). One point crossover is known to have high positional bias as it only chooses one splice point between parents in a given crossover. Choosing one splice point means that the first part of the child comes from one parent and the second part of the child must come from the other parent. A parent with a genotype of "0 0 0 0 0" bred with a parent with a genotype of "1 1 1 1 1" using single point crossover can never produce a child with a genotype where 0s and 1s are interlaced since passed bits must come from single consecutive strings from each of the parents. In positional bias the position of the bits have an effect on what solutions be created through crossover. Distributional bias, on the other hand, is associated with the number of bits parents are expected to transmit to a given child (Eshelman et al., 1989). For instance, UX transmits on average fifty percent of its bits from one parent and fifty percent from the other - the same expected value as single point crossover. However, for UX as the number of bits that differ increases, the probability of selecting only a few bits from a single parent decreases. Because of distributional bias, for highly different solutions UX tends to search farther away from individual

parents than single point crossover. UX does more exploration than exploitation when parents are highly different. Single point crossover, on the other hand, does a more even balance of exploration and exploration when combining two highly different solutions.

Some diversity mechanisms combat operator bias through random introduction of genetic material. Mutation, for instance, can combat operator bias at the cost of potentially making the search too random. Other diversity mechanisms can combat operator bias by ensuring an appropriate level of search in various areas. Fitness sharing, for instance, often prevents the GA from convergence onto single solutions and distributes solutions based on their relative fitness.

2.2 Diversity management choices

The negative forces described above can be combated through a broad range of diversity management choices. Diversity strategies, in a variety of different ways, affect the rate of convergence and the targets for convergence. Duplication, deception, drift, operator bias, epistasis, and hitchhiking are combated by diversity management because diversity management can slow the progress to convergence thus reducing these negative forces.

Many diversity choices take an indirect approach at diversity maintenance. A large number of choices made during the course of setting up a GA determine how much exploration versus exploitation is done while searching. The convergence profile of a GA is a function of a large number of things including the implied problem landscape. The initial population generation, parent selection strategy, recombination operator strategy, mutation operator strategy, survivor selection strategy, population size strategy, and repair operator strategy (where used) can all affect a search profile. A search profile is also sensitive to individual parameters that are used to implement each of these strategies. Additionally many of these choices include a stochastic process making the affect on the convergence profile sensitive to actual random draws. Convergence and diversity are directly related. The faster a GA converges, the faster diversity

leaves the population. It is important to make robust choices in strategies and parameters in order to search the landscape efficiently.

Most typical evolutionary algorithms have no direct method of guaranteeing population diversity (Michalewicz and Fogel, 2000). The concepts of using direct methods to maintain diversity were first explored quantitatively by Mauldin (1984). Mauldin's approach is discussed below in the recombination operator section. Direct methods use a measure of the diversity between solutions (past and / or present) in order to ensure the population remains diverse. Because of the large number of solutions visited, most direct methods only focus on the current population diversity and rely on reintroduction of genetic material to ensure that the algorithm sufficiently searches global sections of the problem landscape.

Figure 6 details a high level view of where diversity can be influenced in GAs. It can be seen that the diversity hierarchy mirrors the process of solving a problem using a GA. An implied landscape is defined by the problem formulation, individual representation, fitness function, and penalty or repair function. These decisions determine the dimensions, size, and surface characteristics of the search landscape. Specific choices for operators in the genetic algorithm (initial population, survival selection, parent selection, recombination method, and mutation method), are more- or less-effective depending on the landscape searched. Any of these operators can contribute to a quick or slow convergence rate depending on how they are setup and the problem landscape being searched. For example if survival selection eliminates duplicate individuals directly, as in Shimodaira (1997), it may slow convergence compared to selection by a method that does not remove duplicate individuals.

Layers of the diversity hierarchy in Figure 6 contain other layers. The inner layers are generally affected by their outer containing layers such that the inner layers can generally preserve no more diversity than was present in the outer containing layers. For instance, since recombination is contained within parent selection, recombination generates individuals that are

no more diverse than the series of midpoints between the parents. Parent selection cannot select parents that are more diverse than those that survived the previous survival selection. This hierarchy implies that decisions at the highest levels of the GA hierarchy have more widespread influences on diversity preservation.

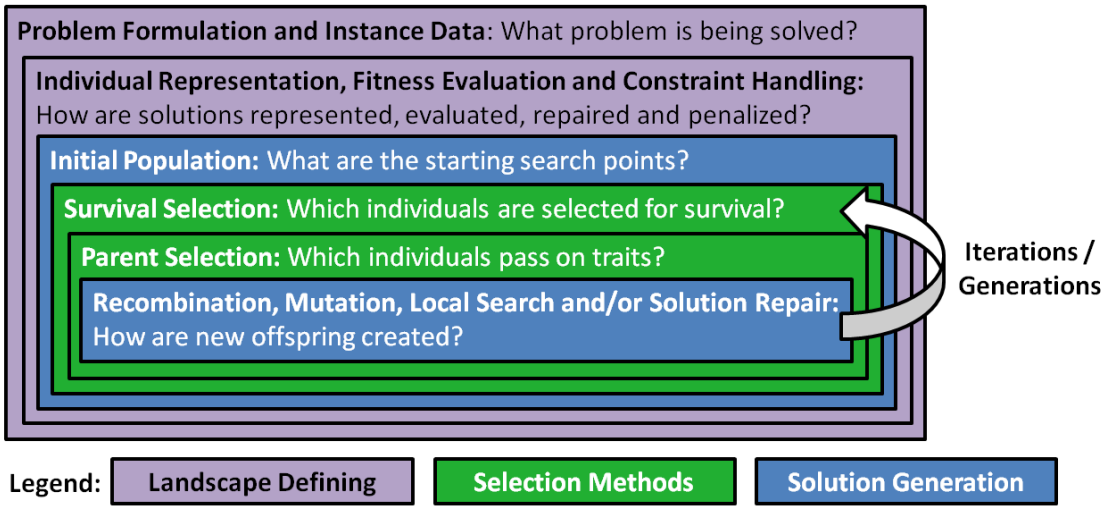


Figure 6 – Diversity Hierarchy In Figure 6 there are three major areas where diversity is controlled in a GA. The first grouping, landscape defining, shows all the elements that can transform the search landscape. Some landscapes may require more diversity for an effective search and differing landscapes can make a problem easy or hard. The second grouping, selection methods, encompasses mechanisms determines how much selection pressure is present. The third grouping, solution generation, covers how solutions are created, usually by recombination and mutation, after parent solutions are selected.

The different layers in the diversity hierarchy are discussed below. None of these sections are collectively exhaustive as there are a vast number of references on GA operators and strategies. However, it is believed that each section sufficiently discusses a sample of the operators and strategies used. The focus is placed on understanding how the choices made in each area affects GA searching as it relates to diversity.

2.2.1 Problem formulation

Many textbooks on optimization recognize that problem formulation is an important first step in optimization. Good problem formulations capture important details while leaving out extraneous details. Good formulations have a structure that makes the solution process easier. Problem formulation involves a problem statement, data identification / collection, identification of design variables, identification of criterion to be optimized, and identification of constraints (Arora, 2004). The decisions made in problem formulation often imply techniques for solving the problem. For instance, a problem formulated with multiple competing objectives is likely to be solved with multi-objective methods. Problem formulation has a large impact on the size, shape, and nature of the solution landscape and helps determine the difficulty of the problem.

2.2.2 Solution representation and the fitness function

The combination of the problem formulation, solution representation, the problem data, and the fitness function imply the actual problem landscape that the GA searches. Some landscapes are easier to search than others because, for example, some landscapes may have higher levels of deception thus confusing the GA more frequently. The impact of solution representation and fitness functions on the problem landscape are discussed in the sections below.

2.2.2.1 Solution representation

Solution representation has important impacts on the difficulty of the search landscape and encoded solutions can make a search space easy or difficult depending on the encoding technique and the problem formulation. Solution representation can have an impact on problem diversity. For instance the messy GA solves difficult problems using redundant and underspecified solution representations. These representations can be used to maintain diversity in the search process (Goldberg et al., 1989).

Messy GAs uses a problem encoding that allows some bits of a solution to be underspecified (absent) and over specified (appearing more than once) (Goldberg et al., 1989). The messy GA encodes solutions into a set of bit position and bit value tuples. For example the solution {1 0},{2

1},{3 0},{4,0} encodes the solution 0100. An under-specified solution of {1 0},{2 1},{4 1} encodes the solution 01*1 where * is an unknown value. An over specified solution of {1 0},{2 1},{1 1},{3 0},{4 1} encodes the solutions 0101 and 1101. The messy GA uses two phases called the primordial and juxtapositional phases. The primordial phase happens first and uses partial enumeration to build 'good' building blocks. In the juxtapositional phase cut and splice operators are used to combine the building blocks. The messy GA assumes that good building blocks can be identified and that local optima can be combined in order to build a globally optimal solution. In addition to the redundancy of the encoding, the type and size of encoding used can affect GA performance.

Binary encoded solutions for problems with real-valued solutions can have varying levels of precision in their phenotypic representations depending on the number of bits in the encoded solutions. Binary encoding of real variables divides a continuous real landscape into a discrete set of points. Schraudolph and Belew (1992) proposed that, for encoded solutions requiring high levels of precision a solution could start with a low number of bits. When the solution with a given precision converges, bits are added to the solution representation with a zoom operator. This action increases the precision of the search since the solution representation becomes larger. The algorithm for changing the solution representation through zooming was named Dynamic Parameter Encoding (DPE). It can be demonstrated that adding bits to the solution representation makes the problem landscape larger. However, since DPE waits for convergence before increasing the precision of the solution, it is, in effect, searching several smaller landscapes. While the name of DPE implies that it modifies parameters, it actually is working on the solution encoding, not the GA variables. DPE is theorized to reduce levels of genetic hitchhiking because a low number of bits are able to hitchhike with an early good solution.

Genotypic and phenotypic representations can have a large impact on the implied landscape. For instance, a problem encoded in a standard binary form can introduce *Hamming cliffs* into a

landscape. Hamming cliffs occur where solutions that are close to one another in the phenotype space are far apart in the genotype space. Consider the phenotype numbers 7 and 8. Encoded in standard binary using four bits they are 0111 and 1000 respectively. While 7 and 8 are as close as possible in the phenotype space, they are as far apart as possible in the genotype space. Hamming cliff problems can be avoided by instead using gray coding which ensures that numbers next to one another only differ by a single bit. Textbooks on genetic algorithms frequently provide a mapping of gray code to standard binary. See for instance, Appendix A in Eiben and Smith, (2003). It has been argued that because of problems with encoding solutions, such as Hamming cliffs, it is easier to represent real-based problems with real-based phenotypic representations (Gibbs et al., 2008).

2.2.2.2 Fitness functions

Fitness functions can take many forms and can be used directly or indirectly in the maintenance of diversity. *Fitness sharing*, used in *niching* or *speciation* strategies, is an example of how the manipulation of a fitness function can shape the landscape and thus help manage diversity. In addition to fitness sharing, other fitness function strategies have been used to manage diversity. For instance, Wong et al. (2003) proposed two repelling algorithms to maintain diversity in the population without using fitness sharing. The repelling approach adds a bonus to the fitness of solutions that have rare alleles thereby promoting diversity.

Fitness function strategies for managing diversity can modify the perceived or implied problem landscape based on the current population. For such strategies, the implied fitness landscape can be viewed as being dynamic. How a fitness function can be used to modify the implied fitness landscape is easily demonstrated by an example. Examine the proposed function (F2) and sharing method in Deb and Goldberg (1989). The F2 function is valid for the real variable x in the range of 0 to 1 and has five peaks. The F2 function is defined in (1)

$$F2(x) = e^{-2(\ln(2))\left(\frac{x-0.1}{.8}\right)^2} \sin^6(5\pi x) \quad (1)$$

In the F2 function each peak has a different height. See Figure 7 for a graph of F2 as a function of *x*.

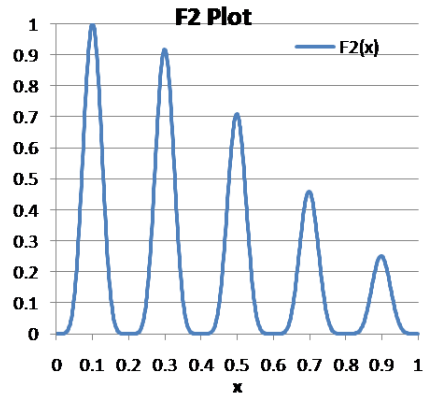


Figure 7 – Plot of F2

For an example of how fitness sharing as proposed in Deb and Goldberg (1989) changes the implied fitness landscape, two different notional populations consisting of ten individuals were created. Their respective fitness values and implied fitness landscapes are shown in Figure 8 and Figure 9. These graphs assume a sharing distance (σ) value of 0.2. It is easily observable that the implied fitness landscape is dependent on the population and that the implied landscape may vary significantly based on the current population. In each case, the implied fitness landscape is the GAs current view of the fitness landscape, based on the current population. In a diversity management strategy such as fitness sharing, solution fitness for each solution in a niche is the best fitness value in the niche divided by the number of solutions in the niche. Therefore, generally having more solutions in a niche causes lower fitness for each member of the niche.

Population P1	
x	F2(x)
0.110	0.928
0.240	0.040
0.260	0.265
0.280	0.690
0.310	0.844
0.380	0.001
0.420	0.001
0.500	0.707
0.720	0.322
0.920	0.172

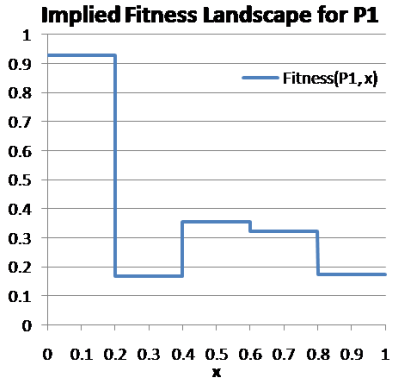


Figure 8 – Implied Fitness Landscape for Population P1

Population P2	
x	F2(x)
0.040	0.041
0.090	0.928
0.100	1.000
0.180	0.001
0.310	0.844
0.380	0.001
0.490	0.668
0.500	0.707
0.720	0.322
0.850	0.037

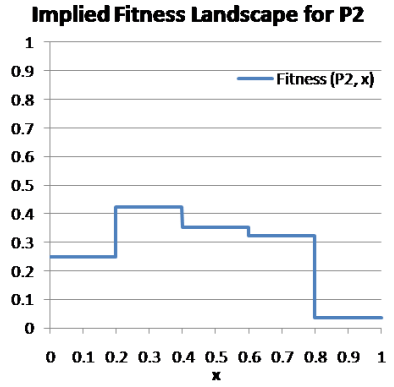


Figure 9 – Implied Fitness Landscape for Population P2

2.2.2.2.1 Speciation by fitness sharing

Niching or *speciation* is a process by which a population maintains diversity by causing individuals to populate different local areas of the landscape. *Crowding* and *fitness sharing* are the two most common methods of speciation. Crowding, is based on a survivor selection strategy and is discussed later in the survivor selection section. Fitness sharing came about as a natural extension of the concept that species have limited resources they must share. Fitness sharing causes individuals to compete for local resources based upon their location in the genotype space and relative fitness values. Fitness sharing generally causes solutions to spread out across the landscape based upon the relative height and width of peaks in the landscape. Speciation has been primarily used in attempting to solve multimodal problems where multiple optima can be found in a single instance of the GA.

Fitness sharing penalizes solutions if too many individuals are in the same region of the solution space (Della Cioppa et al., 2004). Fitness sharing distributes a raw fitness score across multiple individuals based on the number of individuals in the immediate area and the maximum fitness value of other solutions in that area (Deb and Goldberg, 1989). In fitness sharing individual fitness values become a function of the number of individuals in a single local. Thus the problem landscape can be thought of as transforming based upon the current population. Deb and Goldberg (1989) use fitness sharing with the assumptions that niches were evenly spaced σ distance apart and that the number of the niches, q, is known *a priori* and q is much less than the population size n. The algorithm is $O(n^2)$ in complexity for all cases. Miller and Shaw (1995) use a greedy algorithm called Dynamic Niche Sharing to increase the efficiency of finding the different peaks during a GA run. The procedure has $O(n^2)$ complexity early and approaches $O(nq)$ as a run progresses. The paper also introduces Dynamic Inbreeding which is discussed in the parent selection section below. Della Cioppa et al. (2004) propose a method to calculate the optimal values for q and σ. The method does not require *a priori* knowledge about the problem landscape but since it requires a design of experiments process to determine values of n, q, and σ it can be restrictive since experiments may take some time. Shir and Back (2006) propose an algorithm called Covariance Matrix Adaptation for Evolutionary Strategies (CMA-ES). CMA-ES does not require *a priori* knowledge about the number of peaks q and the peak width σ. Instead, a parameter λ specifies the number of individuals per niche and the individual niche radii are calculated based on a learning algorithm and a learning rate parameter α. Unlike previous work, CMA-ES can find peaks with variable width. Peaks are punished for fitness if they contain more than λ individuals thus the implied problem landscape changes with λ and the maximum number of peaks that can be tracked. Della Cioppa et al. (2007) propose a method called Dynamic Fitness Sharing that does not require the number of peaks q to be estimated beforehand. However, this

method was not compared with CMA-ES. The article also introduces an elitist survival strategy for species that is discussed in the survival selection section below.

(Smith et al., 1993) propose a fitness sharing method based on the biological analogy of antibody to antigen matching. This method uses a fitness bidding scheme to modify the fitness of solutions. Since partial matches can bid on solutions, generalist solutions can be maintained when the population is too small to maintain a set of all specialist solutions. Like other fitness sharing methods this method transforms the fitness landscape but, unlike other sharing methods there is a stochastic component to the landscape transformation. This stochastic component is a result of selecting antigens using a random process. Fitness is awarded based on the match of the individual population items to the selected antigen.

In general, fitness function methods that maintain diversity are powerful in that they affect diversity without changing GA operators. However, they can also be somewhat unpredictable because they are based upon the contents of the current population and the problem landscape, both of which are unknown *a priori*.

2.2.2.2.2 Multiple Objective Evolutionary Algorithms (MOEA) and the Pareto frontier

Research in niching led to a logical extension of genetic algorithms for solving Multiple Objective Optimization (MOO) problems. Like niching, MOEAs tend to preserve some amount of diversity because they have mechanisms to find more than one diverse but good solution. MOEAs are discussed further in Chapter 3.

2.2.2.2.3 Penalty functions for constraints.

In lieu of implementing a repair operator, constraints can be managed by application of a penalty on the solution fitness. Application of a penalty component to the fitness function for a constrained optimization problem also changes the problem landscape in that invalid regions are now searchable but generally have poor fitness. Typically, penalty functions are applied based on

some measure of distance such that the further a solution is away from a feasible region, the more it is penalized. Constraint handling through penalty functions has the liability of increasing the size of the landscape but also has the benefit of not introducing bias from a repair operator.

2.2.3 Constraint handling and repair operators

Problems can be formulated so that not all combinations of allele values are possible. For these problems recombination operations are often specialized as they require integral 'repair' operators since not all places in the genotype space are valid. Repair operators may also be required after mutation if mutations produce invalid solutions. Depending on how the repair operator works, the individual presented to the operator, and the closeness of a feasible solution, repair operators may 'fix' solutions to be very alike or dissimilar to the original solution. Often repair operations are implicit to the recombination and mutation operator and as such they may be indistinguishable.

Repair mechanisms have the possibility of changing the problem landscape. Good repair operators maintain the essential properties of a solution. However, even a well designed repair operator may make it more difficult to find certain solutions. Consider for instance when the GA generates an invalid solution close to an optimal solution. The repaired solution could be 'fixed' to be the optimal solution, but it also could be repaired away from the optimal solution. Thus, the repair function impacts how easy it is for a GA to search certain areas of the landscape. The impact of various repair operators on problem hardness is most often studied by comparison of crossover or mutation methods that integrate these operators. Since repair operators are problem-formulation-specific, research into the effect of those operators must be explored on an operator-by-operator and problem-by-problem basis.

2.2.4 Initial population generation & restarts

The initial population and restarted populations have the potential to bias a GA managed search. If certain alleles are not represented in the initial population, they may only be evaluated

by the GA by having mutation ‘blunder’ into them. In most GA implementations, bias from the initial population is combated by starting with a population of pseudo-random solutions. Since even a random population can be biased due to stochastic error on a given trial, many GAs employ a restart function to prevent a single initial population from skewing the entire search. Space filling methods or minimum distance between the initial individuals as suggested in Michalewicz and Fogel (2000) can help prevent random errors from causing the initial population to be skewed to a subset of the search space.

(Michalewicz and Fogel, 2000) and others have suggested that the initial population can include a heuristic for finding ‘good’ initial starting solutions by using heuristics. For example Hiremath and Hill (2005) shows how smart population seeding can speed up GA convergence. However, Eiben and Smith (2003) and others have argued that heuristic initialization does not significantly improve GA performance as ‘good’ solutions are generally found quickly by the GA. The argument points out that the focus of the GA is frequently on finding the smaller improvements in the already ‘good’ solutions found since little resources are wasted on finding the ‘good’ initial solutions.

2.2.5 Survivor selection strategies

Survivor selection affects diversity because selection strategies can survive populations with different levels of homogeneity. One can see this easily by comparing the Steady State Genetic Algorithm (SSGA) with their generational counterparts.

Steady State Genetic Algorithms (SSGAs) often exhibit stronger convergence properties than their generational counterparts. The GENtic ImplemenTOR (GENITOR) algorithm (Whitley, 1989) is a SSGA that replaces the worst solution with a better one when it is found. SSGAs can have the property of very rapid convergence in comparison to generational GAs). SSGAs often require either “large populations and/or a ‘no duplicates’ policy” (Eiben and Smith, 2003) to work well on hard problems.

Crowding by De Jong (1975) and deterministic crowding by Mahfoud (1992) are niching methods for multimodal optimization that use a SSGA with a replacement method based on distance from current members in the population. DeJong's original crowding method randomly selects a number of individuals in a parent population and then replaces the random individual if the created offspring has a higher fitness. This method attempts to maintain diversity over time, but can result in convergence to a single local optimum as the random choice of replacement candidates has an element of stochastic error that can accumulate over time (drift). To address the convergence issue, Mahfoud (1992) did several experiments with different crowding methods and noted that generally the offspring closely resembled one of the two parents. This observation led to an algorithm where only the parents were considered when eliminating the closest solution to the offspring (deterministic crowding). Several individuals can converge to the same point in both crowding methods. In a highly multimodal and large problem space, large population sizes are required for GAs to find many or all optimal solutions.

One fitness sharing survival strategy maintains species in a population by using elitism on one individual from every niche (Della Cioppa et al., 2007). This method was able to solve difficult multimodal problem with smaller population sizes as the preservation of niches was explicit and thus large populations were not required to ensure the survival of each niche. Ensuring survival of diverse solutions slows convergence as mating of diverse solutions commonly causes more exploration than exploitation.

The Diversity Control oriented Genetic Algorithm (DCGA) in Shimodaira (1997) eliminates duplicate solutions at each generation and then uses one of two elite strategies for survival. In cases where elimination of duplicate result in less survival candidates than the population size, the DCGA generates new random individuals that are placed in the population until there are enough individuals to fill the population. In both survival selection strategies an elite is implemented. Then, for Cross-generational Deterministic Survival Selection (CDSS), the

algorithm selects individuals based upon the highest Hamming distance from the best individual until sufficient individuals are selected for survival. For Cross-generational Probabilistic Survival Selection (CPSS), individuals are selected using a probability based on how different they are from the fittest individual and based on two shaping parameters. Some problems are shown to be sensitive to the two parameters implying that survival should not be a deterministic function of Hamming distance from the best individual.

There are classes of GAs that divide the population into subset populations sometimes called *spatially structured EAs* (Tomassini, 2005). By dividing the population, these models affect both parent selection and survivor selection. Two common terms for spatially structured EAs are *multiple-deme models* and *island models*. These models maintain different demes (or islands) and occasionally the demes exchange individuals (Cantu-Paz, 1998). Solutions are passed between the different islands after a number of generations have passed (an epoch's worth). Because these exchanges occur rather infrequently, spatially structured EAs are conducive to parallel computing efforts. Typically there is a structure that indicates which islands exchange individuals based upon a neighborhood definition for the islands (Cantu-Paz, 1998). Spatially structured models are sometimes also referred to as *diffusion models* because it takes a longer period of time for a good solution to propagate itself to all of the populations. Diffusion models can help combat premature convergence because local convergence rarely equates to global convergence in a distributed model. Since the exchange of individuals is relatively small compared to the number of individuals generated, a single population can converge and yet the overall GA can have significant diversity in the other populations. Different populations can use different operators so that if a given operator is causing quick convergence, it will not bias all islands in the model (Eiben and Smith, 2003).

2.2.6 Parent selection strategies

Parent selection has a direct impact on recombination diversity and convergence rate. Mating parents that are very similar generally results in more exploitation while mating parents that are very different results in more exploration. Three common parent selection strategies in use are roulette wheel selection, rank-based selection, and tournament selection.

For roulette wheel selection, if fitness variance is low, it is more likely that parents will be selected from diverse parts of the population. Conversely, more variance in solution fitness indicates that there is a higher probability that the selected parents are more fit. Thus, in populations that have high fitness variance, roulette wheel selection is more likely to breed only the top individuals. This causes faster convergence than a roulette wheel selection on a population with low variance. Parent selection can and does affect the diversity of the search. Tournament parent selection has more or less selection pressure than the roulette wheel selection based on how much variation exists in the current population fitness. For larger tournaments (more individuals in a single tournament) there is a higher pressure to select the most fit individuals. Rank-based selection methods can dial the selection pressure up or down dependent on a parameter that adjusts the proportion of the more fit to less fit solutions selected. Eiben and Smith (2003) describes parent selection and convergence properties of the different operators in greater detail.

Mating restriction schemes have been used to change the level of exploitation versus exploration. Some of these schemes select parents that are a threshold different in order to promote exploration. This is the case with incest control as proposed in Eshelman (1991)'s CHC algorithm. CHC stands for "Cross-generational elitist selection (by incest prevention), Heterogeneous recombination and Cataclysmic mutation" (Whitley, 2001). Incest control, also called incest prevention, ensures that parents are different by at least a designated threshold before they are selected to breed. Other mating restrictions are used to exploit the properties of

certain solutions. Two examples are dynamic (niche) inbreeding or dynamic line breeding (Miller and Shaw, 1995). Dynamic inbreeding and dynamic line breeding attempt to find local optima between a series of similar parents. For example dynamic inbreeding selects individuals from within a niche to handle search that is often contained within the niche.

2.2.7 Recombination operator strategies

The recombination operator can affect convergence rate because recombination can create individuals that are either very alike, or very different from the parents. The dissimilitude of children to parents for different recombination operators is based partly on random choice, and partly on the recombination operator itself. The point of mating solutions is to convey properties from differing parents in creating new offspring. According to the building block hypothesis, recombination attempts to create a child that is more fit than either of the two or more parents by trying different permutations. The terms *recombination* and *crossover* are often used interchangeably to describe the process of using two or more parents to create one or more children. Eiben and Smith (2003) provide more formal definitions for crossover and recombination and a good overview of a spectrum of recombination operators.

For recombination diversity, we first review the one-point crossover function. A typical one-point crossover function takes the first part of one parent and splices it with the later part of a second parent. The point in which the crossover operator ‘cuts’ between the two parents is determined by a random draw. If the random draw happens to be near the beginning or end of a solution, the created children will highly resemble one parent or another. Figure 10 demonstrates how one-point crossover can generate two children that are very similar to their parents. Because the crossover point is near the beginning of the solutions, child C1 only differs from parent P1 by one bit and similarly child C2 only differs from parent P2 by one bit. The chance of creating offspring that are identical or nearly identical to the parents using one point crossover depends on the number of alleles that the parents differ by in total and how identical the parents are in the

first and last portions of their genotypic representation. Goldberg (1989c) presents a generalized version of single point crossover called multi-point crossover.

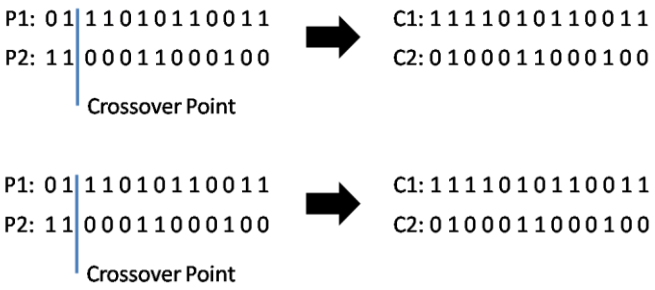


Figure 10 – Diversity in Single Point Crossover

UX picks which properties come from the different parents based on individual random draws for each non-unique allele. UX selects on average half of the genes from one parent. The likelihood that parents will generate identical offspring (to the parents) is a function of the number of different alleles between the parents. For a genotype represented as a string of binary digits the likelihood of generating exact replicates of the parents can be approximated by the binomial distribution where *n* is the number of bits differing between the parents, *p* is the probability of selecting a 0, and *m* is the exact number of successes to check. The binomial distribution is represented with the probability mass function in (2).

$$P(M = m) = \frac{n!}{m!\,(n-m)!} p^m (1-p)^{n-m} \tag{2}$$

For the typical UX operator, *p* equals 0.5 which makes the distribution symmetric about *n*/2. For parents that produce exact copies of children, *m* must be 0 or *n*. For instance, to calculate the probability that parents that differ by five bits will generate exact replicates, *n* is 5, *p* is 0.5, and *m* is 0 and 5. For this example, the probability of creating an exact copy of either parent is 6.25 percent. As *n* increases, there is a lower chance of obtaining an exact copy of either parent via UX.

The half uniform crossover (HUX) operator proposed by Eshelman (1991) is a variation on the UX operator. The HUX ensures that exactly half of the bits that are different between parents convey from one parent and the other half convey from the second parent. This operator ensures maximum diversity of the produced offspring. A child will have equal properties of both parents but can never be an exact copy of either parent except when the parents differ by no more than one bit.

Mauldin (1984) proposed a crossover operator that would check the offspring against the current population. After normal crossover, the new offspring was checked against all individuals in the population. If the offspring differed by less than *k* bits from any member in the population, bits were flipped at random until the offspring differed at least by *k* from each member. This method has the potential to introduce significant amounts of new genetic material into the gene pool. Like simulated annealing concepts, *k* was 'cooled' over time to focus effort on solutions that were closer together as the algorithm progressed. This approach was the first attempt by GA researchers to explicitly maintain diversity in the current population.

There many recombination operators, most of which are problem specific. The point of describing the above operators is to demonstrate that recombination operators can create solutions that are either close to parents (exploitation) or solutions that are farther from parents (exploration). Varying the degree to which the GA produces similar offspring can help manage diversity in the search process. Selecting recombination operators that work in harmony with the overall GA strategy is important.

2.2.8 Mutation operator strategies

Mutation is generally considered an exploration concept, but most implemented mutation strategies display both a component of exploration and a component of exploitation. As the population converges, standard mutation provides more exploration and less exploitation since the operator is more likely to mutate converged genes. For this reason, many references refer to

mutation as, primarily, an exploration operator. Pure random non-repeating search is often not the best search strategy on a constrained budget as random search does not use known problem information or structure. The No Free Lunch (NFL) theorem (Wolpert and Macready, 1997) states that a black-box optimization approach cannot beat non-repeating random search. However, incorporation of problem specific knowledge allows us to beat random search. Most mutation operators are purely random and thus very high mutation rates can force the GA to become no better than random search. Normally mutation rates are kept low so that the GA heuristic can work. Variable mutation rates often give better results than a static mutation rates. GA based search often benefits from a adaptive or self-adaptive approach controlling mutation rate (Thierens, 2002). Adaptive and self-adaptive parameter control are discussed in the following section.

Unlike standard mutation, cataclysmic mutation, or mass extinction, is a widespread changing of many alleles in multiple solutions or the killing-of many solutions in a single phase of an algorithm. This type of mutation can generally can be thought of as being a reset or restart of the search process. Some mass extinction methods, like in CHC, survive one or more representative individuals into the reset population (Eshelman, 1991).

2.2.9 Parameter values & parameter selection strategies

Picking operators for diversity management is only part of the diversity maintenance battle. Once problem formulation is complete, a fitness function is selected, and solution generation and survival selection mechanisms are determined, one still has to pick values for the parameters of the GA. The parameters can have a significant impact on diversity and search profile. The most common parameters, number of restarts (M), population size (N), probability of crossover (p_c), and probability of mutation (p_m) all have an impact on the diversity of the search. M affects the amount of initial randomness is present since most GAs start with random populations. Since a given run of a GA can be prone to drift, higher values of M are preferred. However, high values

of *M* must be balanced with the number of generations per run so that sufficient survival is modeled. *N* can be a critical variable as the larger values of *N* slow genetic drift and convergence as shown in the example in the section 'Genetic drift'. Parameter p_m determines the level of disruption in a search and can delay convergence and negative effects such as search bias. High values for p_m are counterproductive as it can cause the GA to simulate a purely random search and thereby hampers survival-of-the-fittest methods from finding an acceptable solution. Parameter p_c can affect diversity as it determines the percentage time spent examining solutions that are a combination of other solutions. Setting a high p_c may cause little local searching while setting a low value of p_c restricts the amount of global searching. A high or low setting for p_c can change diversity since it partially determines the number of solution combinations considered. One can see from these three parameters that setting 'good' values for parameters is important. Other more algorithm-specific parameters can also become important depending on how they affect the search. Significant research has been accomplished in the area of parameter optimization and parameter control.

The well accepted taxonomy in Figure 11 from (Eiben et al., 1999) can be used to distinguish different approaches to parameter control. The first major division is determined by when the parameter is set. Setting parameters prior to a run is considered parameter tuning while setting the value of the parameter as the GA runs is considered parameter control. Parameter control is further subdivided into three major categories: deterministic, adaptive, and self adaptive. Deterministic parameter control sets the parameter values of the GA without monitoring the actual performance of the GA. Adaptive parameter control changes the parameters according to how the GA is performing. Often this is done with a set of heuristic rules. Self-adaptive parameter control allows the GA to use the built-in survival and mating concepts to control the parameters. Each of the parameter control types is discussed below.

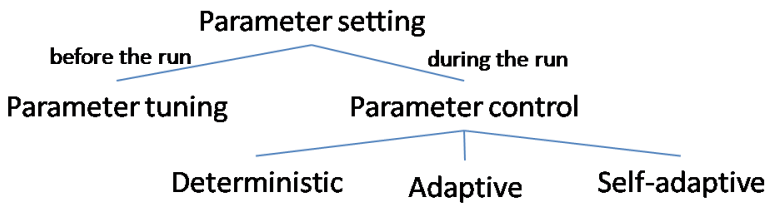


Figure 11 – A Taxonomy for Parameter Settings (Eiben et al., 1999)

Deterministic control influences parameters during the run based upon a predefined schedule. Frequently *cooling schedules* are used in deterministic control. This term first appeared in simulated annealing techniques where it was used to describe the process of accepting 'worse' solutions at a lower rate as the algorithm progresses (Kirkpatrick, 1984). Deterministic control does change based on the progress made during a run because the parameter values are decided before the run begins. One example of a deterministic control is the reduction in niche radius over time used in the Universal Evolutionary Global Optimizer (UEGO) (Jelasity, 1998). Cooling the niche radius over time allows the UEGO to consider niches of finer resolution as the algorithm progresses.

Adaptive parameter control uses measures of how the GA is progressing in order to change the parameter values during a run. Adaptive parameter controls usually behave according to some rule set but the values to trigger the rules may or may not be measured directly from the GA. An examples of adaptive parameter control can be observed in Wong et al. (2003) which introduces an adaptive a method called the Probabilistic Rule-driven Adaptive Model (PRAM) to control recombination and mutation rates by dividing the GA execution into epochs. The GA experiments with random marches in the first component of the epoch and adapts the recombination and mutation rates in the second portion based upon the results in the first part of the epoch. This approach allows the evidence in the first phase to guide the mutation and recombination in the second phase.

Self-adaptive control adjusts parameters by making the parameters a part of the genotype; thus allowing variables from different solutions to mutate and recombine. The theory behind self-adaptive control is that individuals with good parameters values will be more likely to generate good solutions and survive. Hence, the GA will select the parameters that work the best over many generations. Not all parameters can be influenced at the solution level and as a result some parameters are difficult to self-adapt (population size for instance). Self-adaptive control has been criticized for increasing the size of the overall optimization problem (Whitacre et al., 2006). The most commonly selected self-adaptive parameters are mutation and recombination rates. An example of a self-adaptive approach can be seen in Aickelin (2002) where mutation rate is encoded in the solution to increase the GA performance on solving set covering problems.

Recently, Eiben et al. (2007) proposed an additional dimension to the taxonomy to describe the types of evidence used for parameter control. Two categories were proposed: absolute and relative evidence. Absolute evidence is the when a parameter is modified based upon a predefined rule or set of rules but the actual trigger of the event is unknown prior to the run. One example of absolute evidence is in CHC: when zero children are placed in the new population, the value of the incest control parameter is decremented. Relative evidence is measured based upon a comparison of numbers being calculated in the GA. For example in PRAM, the selection of operators in the second part of an epoch is based entirely on the relative measures of how the different operators performed in the first part of the epoch.

2.3 Past diversity-based classifications

The first diversity classification scheme was proposed in Mahfoud (1995). Mahfoud classified algorithms based on which of three negative effects were minimized. The three negative effects map to effects most commonly defined in the literature as drift, operator bias (Eshelman et al., 1989), and high selection pressure. Unfortunately Mahfoud's diversity classification scheme does not effectively distinguish between diversity based methods. The

possible negative effects that can cause a GA to prematurely converge also include deception (Goldberg, 1989b) and epistasis (Davidor, 1991). Furthermore, any diversity mechanisms can combat most of the effects that degrade GA performance simultaneously so to classify GAs, the researcher must determine which effect is most prominently mitigated. Determining the dominant effect is subjective since there are no easy ways to measure the extent that each negative effect contributes to degraded GA performance. A second limiting factor to Mahfoud's approach is that diversity is described only in terms of the current population. However, diversity methods can use measures of inter-individual, inter-population, and intra-population diversity.

Unlike Mahfoud (1995) which classified algorithms as a whole, Ursem (2003) attempted to classify different diversity methods noting that algorithms could use one or more of these approaches simultaneously. Three categories were proposed: structures that lower gene flow, operators to control the selection procedure, and the reintroduction of genetic material. Structures that lower gene flow include spatially oriented EAs such as island or deme models, multinational EAs, and colony based methods. The reader is referred to Tomassini (2005) for a comprehensive review of spatially oriented GAs. Slowing how often individuals migrate between populations can slow overall convergence of the algorithm since convergence of a single population has lesser effects on the convergence of all populations. Operators that control and assist the selection procedure make up the second category. Examples include niching methods such as deterministic crowding (Mahfoud, 1992), and fitness sharing (Deb and Goldberg, 1989). Fitness sharing promotes diversity by penalizing solutions that are very genotypically similar and crowding changes the way in which survival is accomplished in order to slow the convergence of individuals to a single point in the search space. The third category of diversity methods, reintroduction of genetic material is accomplished when new genetic material is added to the gene pool in some way (usually through mutation and random restarts).

Ursem's (2003) classification scheme has some significant drawbacks. Firstly, the scheme classifies some of the more elegant diversity based methods but fails to evaluate some of the most basic diversity decisions in a GA. For instance recombination, a basic operator in almost all GAs, cannot be classified by its diversity approach using Ursem's method. However, we know that some recombination operators such as half-uniform crossover (HUX) generally create more diverse solutions than standard uniform crossover (UX) and single-point crossover (Eshelman, 1991). Further the classification scheme does not identify the fact that solution representation can be an important part of diversity and as a result it cannot differentiate diversity achieved by changing the solution representation such as the highly redundant representation in messy GAs (Goldberg et al., 1989). Lastly, the category of reintroduction of genetic material is really two categories since methods can either reintroduce genetic material or attempt to preserve existing material.

Bhattacharya (2004) added two more categories to Ursem's classification scheme. These additional categories were Dynamic Parameter Encoding (DPE) (Schraudolph and Belew, 1992) and diversity controlled algorithms. DPE is a solution representation technique that adds precision to a solution as the algorithm converges. The DPE category attempts to address diversity in solution representation but the category is incomplete as there are many other ways to approach diversity in GAs using solution representations. The diversity controlled category is undefined because Bhattacharya does not specify what is meant by control. We assume that control here is the measurement of diversity and adjustment of a population diversity based on that measure. The diversity controlled category is unfortunately not mutually exclusive of the original three categories. The attempt to classify some diversity methods as controlled highlights the fact that some diversity measures use an implicit level of control while others use a more direct level of control.

Although not proposed as a classification scheme, Abbass and Deb (2003) made the distinction between diversity promoting and diversity preserving methods. According to Abbass and Deb (2003), diversity preservation is the act of "maintain[ing] diversity which already exists in the population". In contrast, promotion is the act of adding "new [or rare] variations to the population." Diversity promotion typically happens through the restart of an algorithm or through the use of mutation. Diversity preserving methods, on the other hand, work only on the current genetic material in a population. Every operation in a GA that does not introduce random genetic material impacts diversity preservation.

The major shortcomings of past diversity classification schemes is that they have difficulty classifying the diversity approach of standard GAs. These past schemes also can have significant overlap within themselves making classification difficult and somewhat ambiguous. Classifying GAs by only a few diversity descriptors is difficult because diversity is affected by many things such as operators, problem formulation, problem representation, fitness function, and parameter settings. Furthermore since there are many areas in which diversity is influenced, diversity control is implemented in many different ways.

2.4 Classification of solution representations

Solution representation is the way in which a solution is manifested in the algorithm. Ultimately a solution representation maps to the decision variables of a problem. The addition of DPE to the classification of diversity methods by Bhattacharya (2004) highlights the need for classifying solution representations. For many of the first GAs solutions were represented as a string of binary digits. In the event that these problems were not naturally binary, the solution was often encoded into a binary form. In the past several decades solution representations in GAs have evolved to include more natural representations such as real encoded values or permutation strings. We use three categories to describe solution representations: encoded versus natural, interpreted versus direct, and static versus dynamic.

The encoded versus natural, and direct versus indirect classifications indicate how the variables are manifested into the solution representation. Some representations work on a set of variables that then imply a solution through a heuristic rule which maps the genotype to specific solutions (interpreted). Other representations are more direct: the variables are represented in the solution itself. Two examples of interpreted representations include (Aickelin, 2002) and (Carlson and Hougen, 2010). For instance in indirect set covering (Aickelin, 2002), the GA operates by evaluating row fitness rather than selecting individual rows. A heuristic is applied to determine which rows actually cover the columns. The heuristic that does the translation from the encoded to the decoded solution should be evaluated as an operator (see the section titled 'Classification of GA Operators' below) since such heuristics can have different approaches to diversity. Both direct and interpreted representations can also be encoded or in a natural form. Variables that have their natural representation are in their corresponding natural type. For instance if a problem has a real variable *x*, a natural representation of *x* could be a floating point type or double floating point type. If *x* is represented in a non-natural form, it is encoded. For instance if *x* was represented as a binary string and interpreted into a real number for the evaluation of the solution, the variable *x* is encoded in the representation. We define solutions that have any encoded variables as an encoded solution. In representations that are both natural and direct, the phenotype space is identical to the genotype space.

Solution representations that can change over the course of the GA are dynamic representations. DPE (Schraudolph and Belew, 1992) and messy GAs (Goldberg et al., 1989) are two examples that modify solution representation over the course of the run. In DPE, additional bits of precision are added to variables as the algorithm converges. In messy GAs, variables can be underspecified or redundantly represented and thus the representation string may get larger or smaller. While dynamic representations can be elegant, most GAs use a static representation – the components of the genotype are fixed.

2.5 Classification of constraint handling, fitness evaluation, and population structure

Approaches to constraint handling and the fitness function also define the search implied landscape(s). Constraint handling is typically approached in one of two ways: validity assured or penalized. In the validity assured approach, only valid solutions are allowed to be evaluated for fitness by the GA. This approach often requires an independent repair operator if the recombination and mutation methods by themselves cannot assure validity. The repair operator can be qualified as an operator (see the section titled 'Classification of GA Operators' below). The penalty approach to constraints gives a fitness penalty to invalid solutions but can be useful when it is difficult to generate many feasible solutions. In a penalty based approach, invalid solutions are represented in the population and can pass on genetic traits to offspring.

Fitness evaluation is the last component to define the problem landscape(s). Fitness evaluation can be either natural or modified. A natural fitness function is one that has a 1 to 1 correspondence with the original problem formulation's objective function. Modified fitness functions, such as functions that incorporate additional penalties or rewards to certain solutions, can be classified as either relative or absolute. Relative fitness functions have fitness that may depend on other members in the population (such as fitness sharing (Deb and Goldberg, 1989)) while absolute fitness functions only depend on the individual itself (such as commonly used invalidity penalty functions). The fitness function itself also may have any of the classifications of GA operators (see the section titled 'Classification of GA Operators' below).

2.6 Classification of Population Structure

As pointed out by Ursem's classification (2003), population structure can have a strong effect on diversity. There are a wide variety of names for structured populations including island, deme, multinational, and colony models. These models share the common characteristic that they create separate sub-populations that pass information between each other in some fashion. Often

these models use a structure such as a hub-and-spoke, grid, circle, or toroid to determine which individuals move between populations. Individuals move between sub-populations based on a migration operator. Migration operators can be classified by the general operator approach in the next section. We classify GAs based upon their approach to a population: panmictic or structured. Panmictic GAs are those GAs that use a single unified population while structured GAs divide the population into separate component collections. The reader is referred to Tomassini (2005) for a more complete description and classification of structured population GAs.

2.7 Classification of GA Operators

Typically major GA operators include initial population generation, survival selection, parent selection, recombination, mutation, and migration operators in spatially oriented GAs (Tomassini, 2005). It is also common to see repair operators independent of mutation and recombination and to see implementation of local search operators that are applied after solution creation. The study of *memetic* algorithms which combine local search procedures into a GA structure has received much interest in light of the NFL theorem (Moscato et al., 2004). Local search procedures and repair operators are typically invoked after either- or both-of the mutation and recombination operators. We classify local search and repair operators separate from the mutation and recombination operators when a distinctly independent set of code is used for those operators. The use of local search procedures often cause quick convergence and loss of diversity (Eiben and Smith, 2003) and as such these operators should be classified separately when possible. While the fitness function is not typically considered a GA operator, for classification purposes we categorize it as an operator. All of the ways we qualify GA operators also apply to the ways in which we can qualify the fitness function.

There are at least five major diversity categories in which we can divide GA operators: *preserving vs. promoting*, *directness*, *occurrence pattern*, *randomness*, and *complexity*. Of these, complexity, randomness, and occurrence pattern are general categories that qualify operators. In

contrast, preserving vs. promoting and directness are inspired by the operator's approach to diversity. Complexity and randomness are commonly used to qualify operators already and as such are not new categories.

2.7.1 Complexity

Rating operators in terms of their complexity is useful since, frequently, different diversity methods often use varying levels of complexity. Operators can be described in terms of their worst case, best case, and average case performance. The reader is referred to Sipser (1996) for more on software complexity theory.

2.7.2 Randomness

GAs are stochastic – they incorporate the use of random variables in the search process. Some operators such as survival selection are often deterministic while others such as recombination are often stochastic. Diversity methods such as the Cross-generational Probabilistic Survival Strategy (CPSS) in Diversity Control oriented Genetic Algorithm (DCGA) (Shimodaira, 1997) can use a stochastic process in different areas so we qualify operators accordingly.

2.7.3 Occurrence Pattern

Operators are executed at different times in the GA and thus operators can be classified in terms of their pattern of occurrence. For a given algorithm, a section of code could occur exactly once, could occur single times at multiple points in the algorithm, could occur repeated times in a given area of the algorithm, or could occur for every repeated step an algorithm performs. These correspond to the four major categories of occurrence: one-time execution, cyclic execution, phased execution, and recurring execution. One-time executed code is simply code that is only executed once. Cyclic events are singular events that occur once for every time a triggered condition is met. Restarts are typically cyclic in that they happen based upon one or more termination conditions. Phased-based code is executed when the algorithm is in a certain phase of

an overall larger search strategy. The Diversity-Guided Evolutionary Algorithm (DGEA) (Ursem, 2003) uses distinct diversification and intensification phases to control mutation and recombination respectively. In most GAs, recurring operations include survival selection, parent selection, mutation, and crossover.

2.7.4 Diversity Preservation vs. Promotion

Operators can be classified according to their general approach to diversity. Abbass and Deb (2003) defined *preserving vs. promoting* as a classification of diversity although it was discussed in the context of the overall GA approach rather than being operator-specific as proposed here. For describing operators here, the promotion of diversity is defined as the creation of genetic material that *may be* different from all of the possible material created by the recursive recombination of the current population. Thus, algorithms that are diversity promoting must be capable of adding "new" genetic material to the population. For this work, diversity preserving methods are those methods that do not promote diversity.

2.7.5 Directness

The last classification of operators is based upon how diversity is measured. Direct diversity methods are those methods that use a metric of diversity to control or guide their process. An example of a direct method of diversity control is the incest control mechanism in CHC where individuals do not mate unless they are at least a certain threshold apart. A second example of a direct method is the survival selection in DCGA that eliminates duplicate solutions and uses the distance from the best solution for selecting survival candidates. Indirect methods are more common. The majority of operators in GA literature are indirect because direct methods have an overhead associated with explicitly measuring diversity.

Direct measures of diversity can generally be divided into three different subcategories based on how diversity is measured. Sub-population measures are distance measures that are only assessed for a subset of the population at any one given time. A common sub-population measure

is the distance between two individuals. Overall population measures calculate a diversity measure with respect to every solution in a population. A common population measure of diversity is the population variance. Population-over-time methods use measures of time history of diversity. Figure 12 shows how diversity directness can be classified.

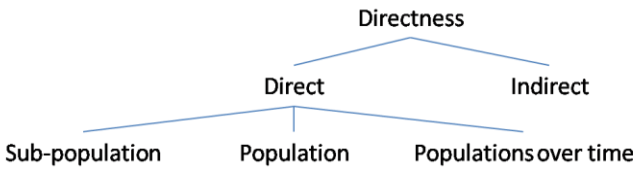


Figure 12 – Operator directness taxonomy

In addition to classifying direct diversity of operators by their scope of the measure, direct methods can be classified by the space and method used to measure the difference of solutions as noted in Mahfoud (1995). Most explicit diversity measures use a Euclidean distance in the encoded genotypic space. However, some use the decoded genotypic space or even measures on the objective values of solutions. Furthermore, measuring based on Euclidean distance is not required and often other distances such as Hamming distance are used. These distances can be quite different since there are a variety of ways in which solutions can be represented, levels in which diversity can be measured, and ways in which distances can be measured.

In summary the overall classifications for operators and the fitness function are *preserving vs. promoting*, *directness*, *occurrence pattern*, *randomness*, and *time order complexity*. Directness is further divided based on the measure of diversity with the options of *sub-population*, *population*, and *populations-over-time*. Directness is also classified in terms of what space distance is measured and can be measured in the phenotypic, genotypic, and objective spaces.

2.8 Example classifications of algorithms based on diversity

Three sample algorithms with different diversity approaches have been classified according to the proposed diversity classification scheme. The algorithms: CHC (Eshelman, 1991), DCGA (Shimodaira, 1997) and DGEA (Ursem, 2003) are discussed below.

Table 3 and Table 4 classifies the diversity approach used by CHC. For CHC the overall time order for all operators is no greater than O*(Nl)* and CHC only requires a single parameter to be tuned. Despite these two large advantages, CHC is highly tied to the HUX recombination operator. Furthermore, CHC has a hidden level of computational complexity because for each generation, there may be a number of random draws that result in no mating of solutions. While the operators are all no greater than O(*Nl*), a portion of parent selection does not result in creation of children. The percent of unproductive parent selections depends on the population size, population similarity, and current level of the incest control parameter. CHC only has diversity promotion in the cataclysmic mutation and initial population operators. This indicates that CHC may be vulnerable to strong hitchhiking effects but also that CHC may spend little time doing counterproductive mutations. CHC does not have a direct elimination of duplicate individuals so it can be vulnerable to drift effects. However, the authors claim in Schaffer et al. (1999) that drift can be postponed "almost indefinitely" due to the strong diversification properties of the HUX operator in combination with incest control.

For DGEA solution representation has the following properties: it uses encoded variables, it is direct, and it is static. Validity is assured when handling constraints. A natural fitness evaluation is used. Finally, the algorithm uses a panmictic population structure. Table 4 describes the two parameters used in the CHC method according to Eiben's parameter setting and control classification taxonomy discussed previously.

Table 3 – CHC operators in Eshelman (1991)

Operator	Directness	Promoting vs. Preserving	Occurrence Pattern	Randomness	Time Order (worst case)
Fitness function	Indirect	n/a	Recurring	Deterministic	*O(l)*
Initial population	Indirect	Promoting	One-time	Stochastic	*O(Nl)*
Cataclysmic mutation	Indirect	Promoting	Cyclic	Stochastic	
Survival (generational, elitism)	Indirect	Preserving	Recurring	Deterministic	*O(N)*
Parent selection (random pairing with incest control)	Direct (genotypic & sub-population)	Preserving	Recurring	Stochastic	*O(l)*
Recombination (HUX)	Direct (genotypic & sub-population)	Preserving	Recurring	Stochastic	*O(l)*
Mutation (none)	n/a	n/a	n/a	n/a	n/a

Table 4 – CHC parameters in Eshelman (1991)

Parameters	Description	Parameter Setting / Control
N	Population size	Tuned
l	Incest control	Adaptive (absolute evidence)

Table 5 characterizes the DCGA operators and their approach to diversity. The worst case time order of the operators is $O(N^2 l)$ which is more complex than CHC. However, the paper argues that the average case is significantly faster than that as the elimination of duplicates must only be accomplished for solutions with the same fitness value. DCGA, like CHC and DGEA, does not require restarts and the initial population is only considered once. Unlike CHC, DCGA uses mutation and also promotes diversity when the number non-duplicate solutions in the population fall below *N*. Unlike the other two algorithms, DCGA uses a stochastic process for survival selection with a probability of selecting a solution for survival based on the Hamming distance from the best solution and the α and *c* parameters. For DCGA solution representation has the following properties: it uses encoded variables, it is direct, and it is static. Validity is assured when handling constraints. A natural fitness evaluation is used. Finally, the algorithm uses a panmictic population structure. Table 6 describes the parameters used in the DGEA method according to Eiben's parameter setting and control classification taxonomy discussed previously.

Table 5 – DCGA with CPSS operators in Shimodaira (1997)

Operator	Directness	Promoting vs. Preserving	Occurrence Pattern	Randomness	Time Order (worst case)
Fitness function	Indirect	n/a	Recurring	Deterministic	O(*l*)
Initial population	Indirect	Promoting	One-time	Stochastic	*O(Nl)*
Survival (generational (μ+λ), distance & duplicate elimination, elitism)	Direct (genotypic & population)	Preserving	Recurring	Deterministic	$O(N^2 l)$
Survival (generational CPSS)	Direct (genotypic & sub-population)	Preserving	Recurring	Stochastic	O(*N*)
Survival (low population size)	Indirect	Promoting	Recurring	Stochastic	O(*N*)
Parent selection (random pairing)	Indirect	Preserving	Recurring	Stochastic	O*(1)*
Recombination (two point)	Indirect	Preserving	Recurring	Stochastic	O*(l)*
Mutation	Indirect	Promoting	Recurring	Stochastic	O*(l)*

Table 6 – DCGA with CPSS parameters in Shimodaira (1997)

Parameters	Description	Parameter Setting / Control
N	Population size	Tuned
p_m	Mutation probability	Tuned
c	CPSS shape parameter	Tuned
α	CPSS shape parameter	Tuned

The Diversity Guided Evolutionary Algorithm (DGEA) in Ursem (2003) is used here to demonstrate how the proposed classification scheme works. DGEA has two distinct phases; the exploration phase uses mutation to generate solutions that are different from the current population while the exploitation phase uses recombination and fitness evaluations to narrow the search. Threshold values for diversity using parameters d_{low} and d_{high} determine when to switch between phases based upon a measure of diversity across all individuals in the population. Unlike more typical GA variants, DGEA does not employ random restarts. DGEA solution representation has the following properties: it uses natural variables, it is direct, and it is static. Validity is assured when handling constraints. A natural fitness evaluation is used. Finally, the algorithm uses a panmictic population structure.

Table 7 characterizes the DGEA operators and their approaches to diversity. In the exploration phase, crossover and fitness evaluation are not performed – mutation is used to redistribute the population. In the exploitation phase, DGEA uses recombination to create solutions that are evaluated for fitness. As a result these three operators are phased. The phase determination operator uses a genotypic measure of the diversity across all individuals in the population to determine which phase the algorithm is in. DGEA uses thresholds of diversity defined by parameters d_{low} and d_{high} to control these phase switches. For the time order column the variable l refers to the number of alleles and the variable N refers to the population size. We can easily determine where the stochastic aspects of the model reside and note that the problems solved here are deterministic. Additionally, we can easily identify the places where new genetic material can be inserted into the population. Table 8 describes the parameters used in the DGEA method according to Eiben's parameter setting and control classification taxonomy discussed previously.

Table 7 – DGEA operators in Ursem (2003)

Operator	Directness	Promoting vs. Preserving	Occurrence Pattern	Randomness	Time Order (worst case)
Fitness function	Indirect	Preserving	Phased	Deterministic	O(l)
Initial population	Indirect	Promoting	One-time	Stochastic	*O(Nl)*
Survival selection - generational (μ,λ); elitism	Indirect	Preserving	Recurring	Deterministic	*O(N)*
Phase determination	Direct (genotypic & population)	Preserving	Recurring	Deterministic	$O(N^2 l)$
Binary tournament parent selection	Indirect	Preserving	Recurring	Stochastic	O*(1)*
Arithmetic / uniform hybrid recombination	Indirect	Preserving	Phased	Stochastic	O*(l)*
Mutation (20% of alleles)	Indirect	Promoting	Phased	Stochastic	O*(l)*

Table 8 – DGEA parameters in Ursem (2003)

Parameters	Description	Parameter Setting / Control
N	Population size	Tuned
p_c	Recombination probability	Tuned
d_{low}	Phase threshold parameter	Tuned
d_{high}	Phase threshold parameter	Tuned

From the above classification, CHC, DGCA, and DGEA all have strengths and weaknesses. These three are a very small sample of different GA techniques and algorithms. Classification of different algorithms can lead to a deeper insight about how they are different, especially when compared with performance. Filling in these tables does not, by itself, lead to an understanding of how the different components in a given GA interact to solve optimization problems so the classification scheme does not replace freeform descriptions of the algorithms.

Chapter 3 Multi-Objective Optimization (MOO) and MOEAs

Problems can be formulated in such a way that multiple objectives are desirable but incomparable in their natural state. For instance, a problem of maximizing safety and minimizing cost in production planning is a MOO problem. The objectives, increasing safety and lowering cost, are incomparable in their native form as safety is not frequently measured in dollars and many times may not have a numerical scale. Objectives in MOO are often, but are not required to be, in competition with one another. One of two major approaches can be used to solve a MOO problem: reduction to a single objective model or the direct search for many good and yet incomparable solutions.

A MOO problem can be transformed into a Single Objective Optimization (SOO) problem using preference models that represent tradeoffs between the objectives. There is a large body of research in the area of decision making focusing on creating preference models (and Reilly, 2001,Kirkwood, 1996,Bouyssou, 2006, Belton and Stewart, 2002). This research can be broadly categorized as Multiple Criteria Decision Making (MCDM) (Figueira et al., 2005). Preference models take the natural scales of several objectives and the preference tradeoffs elicited from a decision maker to create a single objective mathematical model for the problem. If a preference model exists, the problem can be solved using a SOO technique where the fitness function is simply the mathematical preference model.

The second of the two major approaches does not require prior preference elicitation as the approach attempts to find one, many, or all, non-dominated solutions. This approach does not require solicited preferences from the decision maker implying that there is no single best answer. Instead of having a single best solution,, the decision maker will be required to pick between

many non-dominated solutions. A solution is considered dominated by another solution if it is worse in at least one objective and no better in all other objectives. Non-dominated solutions are those solutions that are at least as good as all other solutions in all objectives. Pareto optimal is another name for a non-dominated solution (Coello Coello et al., 2007). The set of all Pareto optimal solutions is called the Pareto optimal set. The Pareto optimal set represents the solutions which a rational decision maker (Simon, 1982) would consider when making a decision. In other words, the Pareto optimal set represents the 'best' solution choices for the decision maker. When plotted on a graph of the solution space, the Pareto optimal set forms the Pareto frontier. Among other techniques, Multiple Objective Evolutionary Algorithms (MOEAs) have been used to search problems for the Pareto optimal set (Coello Coello et al., 2007).

3.1 Multi-Objective Evolutionary Algorithms (MOEAs)

MOEAs have been used to address a significant number of MOO problems (Coello Coello et al., 2007). These algorithms attempt to find multiple, non-dominated solutions that approximate the Pareto front by using various methods of determining solution dominance. Jensen (2004) identifies that most common MOEAs have three distinguishing features: fitness assignment based on Pareto-domination, elitism / archiving, and niching. In other words, MOEAs must do three things differently than single objective evolutionary algorithms: assign fitness based on incomparable objectives, remember (or survive) many Pareto optimal solutions rather than just one, and maintain diversity in order to search different areas of the problem space for non-dominated solutions. Many MOEAs remember solutions on the known Pareto front by maintaining an archive of non-dominated solutions. As the number of objectives increase, MOEAs generally have more difficulty finding good approximates to the Pareto front because as the Pareto front increases in size and dimensions, the general task of finding solutions on that front becomes harder. This is because more solutions in a given population could be on the

known front. Many solutions on the known front causes MOEAs difficulty because the algorithms have difficulty distinguishing differences in fitness.

MOEAs generally are slower than their single-objective GA counterparts in terms of their order of complexity as there are additional computations to track and maintain solutions on the Pareto front (Jensen, 2003b). Many MOEAs are $O(G\ M\ N^2)$ where G is the number of generations, M is the number of objectives, and N is the population size (Jensen, 2003b). Jensen argues that the runtime complexity of these algorithms can be reduced to $O(G\ N\ log^{M-1}N)$ without loss in functionality but the order of complexity is still greater than that of the 'basic' genetic algorithm which, in contrast, has a complexity order of $O(G\ N)$. To accomplish this improved $O(G\ N\ log^{M-1}N)$ performance, a set of recursive helper functions are used to divide the search space into the smallest units possible and assemble them into appropriately ranked solutions. While the worst case complexity of the MOEA algorithms may favor the basic GA, the actual runtimes on some problems may be better for the multi-objective problem formulated versions. One explanation for this is shown in Knowles et al. (2001) where adding objectives to a problem transforms the searched landscape and changes the problem difficulty. Further discussion on this topic is given in the following section. First, in order to understand the differences between several basic MOEAs, an examination of four commonly used MOEAs is presented here.

Common basic MOEAs include the Strength Pareto Evolutionary Algorithm version 2(SPEA2), the Non-dominated Sorting Genetic Algorithm version II (NSGA-II), the Pareto Envelope based Selection Algorithm version II (PESA-II), and the Pareto Archived Evolution Strategy (PAES) (Coello Coello et al., 2007). These four algorithms are discussed in more detail below. While there are more MOEAs than the four below, this sample is believed to be a good representation of the 'basic' MOEA because of the popular acceptance of these algorithms in literature and practice. Coello Coello et al. (2007) provides a more comprehensive list of MOEAs.

SPEA was introduced in Zitzler and Thiele (1999) and improved to SPEA2 in Zitzler et al. (2001). To calculate fitness, SPEA2 first determines the dominance count for a solution. The dominance count is the number of solutions in the population that a given solution dominates. Fitness for a given solution is calculated by adding the dominance count of a solution to all dominating solutions. Figure 13 shows example SPEA fitnesses for the archive (a set containing individuals that are thought to be on the Pareto front) and the population. *f1* and *f2* are the two objectives to be maximized. The points with a fitness value of 0 are archived solutions since they are on the currently known Pareto front. The other points are population members. In SPEA2, lower fitness values correspond to a more preferred solution. The dominance count fitness values in SPEA2 were an improvement on the original SPEA, which used a concept of hyperboxes to calculate fitness. SPEA2 captures more detail regarding the dominance relationships for solutions in the population. SPEA2 also adjusts fitness for solutions with equal dominance counts. Density adjustment, creates differentiation for solutions that have the same fitness in the non-dominated set by applying a density measure that adjusts the fitness to favor solutions that are in less dense areas. Finally, SPEA uses a truncation method on the archive set to ensure the Pareto front is covered as evenly as possible. The truncation method is based on a normalization of the measures of distance in the different objective spaces. The truncation favors solutions with extreme values.

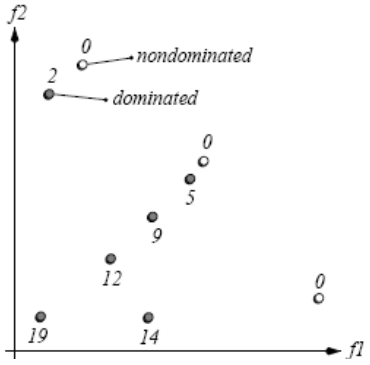


Figure 13 – A Sample SPEA2 Solution Fitness (Zitzler et al., 2001)

NGSA-II described in Deb et al. (2002) uses a crowding distance measure and a front ranking method to determine solution fitness. NSGA-II uses a front count (non-domination rank) to determine a solution's dominance value. First, all solutions that are in the current Pareto front are found and assigned a rank of zero. The solutions in this rank are not dominated by any solutions. Then, those solutions are eliminated from the rank consideration and a second front is found. Solutions on the second front are given a non-domination rank of one; indicating that these solutions are on the second 'front'. This process is repeated until all solutions are assigned a non-domination rank. Non-domination rank determines the highest order of fitness so that solutions with a high front number have worse fitness than solutions with a low front number. Crowding distance is used to break ties between solutions within a given rank. The crowding distance measure is relative to other solutions in each objective and is normalized by objective. Crowding distances are calculated only for solutions in the same rank and favor is given to solutions that are far apart from each other. Solutions in the parent population and child population compete to select the next generation. Since fitness is first based on domination rank, known Pareto optimal solutions are kept over other solutions and as a result a separate archive of Pareto optimal solutions is not required in NGSA-II. NSGA-II uses tournament selection when picking parents.

PESA and PESA-II were first described in Corne et al. (2000) and Corne et al. (2001). Both algorithms use an archive population in order to track the currently known Pareto optimal solutions. PESA-II uses a hyperbox approach to determining the spacing of individuals on the Pareto frontier. Hyperboxes evenly divide the search space. Parent selection in PESA-II is done by hyperbox rather than by individual. After a given hyperbox is selected, one of the parents in that hyperbox is selected at random. The PESA parent selection mechanism prevents search bias (drift) caused by having a high number of individuals in a given hyperbox. Individuals are eliminated from the hyperbox with the greatest 'squeeze factor' when the number of individuals in the archive size exceeds a maximum value.

Unlike the previous three methods, PAES is an evolutionary strategy where only a single solution is in a given population and the emphasis is placed on local rather than global search. PAES was first described in Knowles and Corne (1999). PAES is considered a multi-objective hill climber and has a (1+1) survival strategy. Mutation is the main operator in PAES and an archived list keeps track of visited non-dominated solutions. A single solution is mutated each iteration to generate a new solution. In order to keep track of non-dominated solutions, a grid is used to track the different non-dominated solutions and when the archive size is exceeded, a solution from the most populated grid is eliminated. PAES's main strength is its simplicity as it only requires two parameters to tune and has a very limited set of operators required for implementation. However, unlike the previous three algorithms, since PAES does not maintain a population, it cannot exploit the building block hypothesis. In other words, it cannot combine two or more good solutions to create better solutions.

Overall there are strengths and weaknesses to different MOEAs and there is no single algorithm that is clearly victorious for all problems. The NFL theorem (Wolpert and Macready, 1997) states that even though different algorithms may perform better or worse on different problems, if all of the algorithms incorporate the same amount of problem specific knowledge, no one algorithm can beat the other on average against the full set of all possible problems. While these basic MOEAs may be, in theory, similar, certain algorithms, like NSGA-II, appear to enjoy more frequent use due to their speed and broad based acceptance.

3.2 Solving single objective problems using MOO methods (multi-objectivization)

A recent thread of research has addressed classes of certain single objective problems with the use of multiple objective methods. While the concept of problem decomposition or 'divide-and-conquer' is not new, application of solving problems by splitting objectives is more current. The method of transforming single objective problems into multiple objective problems for finding solutions to the original problem has been referred to as multi-objectivization. Knowles et

al. (2001) first used the term and identified that some single objective problems could be solved easily using a multiple objective hill climber. In the paper a HIerarchical-iF-and-only-iF (H-IFF) problem was presented. The H-IFF problem can be split into two dimensions in which the transformed landscape did not have local minima to overcome. Several MOEAs, a simulated annealing approach, and a multiple dimension hill climber were compared with results favoring the multiple objective approaches. However, the H-IFF portion of this work can be critiqued because in real world problems knowing the appropriate transformation to eliminate all local minima implies full knowledge about the landscape. Since such knowledge is unavailable on real-world problems, predicting the best transformation for multi-objectivization is impossible. Thus the H-IFF work did not show how the technique would extend well to real-world problems.

In addition to the H-IFF transformation, Knowles et al. (2001) attempted to solve several Traveling Salesman Problems (TSPs) using multi-objectivization by splitting the single TSP into two sub problems. Cities *A* and *B* were chosen at random, or by distance, to divide the problem along the tour of cities. Minimizing the travel cost between cities between *A* and *B* was one objective and minimizing cost for travel between *B* and *A* was the other objective. Since the subtours *A to B* and *B to A* compose a full tour, all Pareto optimal solutions were feasible solutions to the original TSP problem. The paper compared the multi-objective PAES and PESA algorithms with a Single-objective Hill Climber (SHC),a single-objective simulated annealing algorithm, and with a single objective GA using deterministic crowding. The multi-objective algorithms outperformed their single objective counterparts in nearly all cases. The paper has been criticized by Jensen (2004) for using static selections for cities A and B because they can unevenly divide the search space. Knowles et al. (2001) conjectured that problems where the fitness function can be divided into sections are good candidates for multi-objectivization but did not give clear criteria for when this is the case.

Abbass and Deb (2003) explored using a MOEA to solve a single objective problem where the first objective was the solution fitness and the second objective was either the solution age, a random value, or the reverse of the first objective. Results show that solution age makes the best objective; however, the paper compares the single objective GA without significant diversity mechanisms to a MOEA that survives one or more random solutions. The unbalanced comparison makes it difficult to determine how the results may be skewed by the early convergence of the single objective GA. A stronger diversity mechanism or multiple restarts of the simple GA would have made for a better comparison.

Multi-objectivization was applied to the Job Shop Scheduling Problem (JSSP) in Jensen (2003a) and later in Jensen (2004). The goal of the JSSP is to schedule a number of jobs on a number of machines in order to minimize total job flowtime. The total flowtime objective can be divided into sections since total job flowtime is simply the sum of all individual job flowtimes. To make the SOO problem into a MOO problem, the concept of helper-objectives were introduced where a given helper-objective was simply the flowtime for a single job. Because it was important to focus on the original main objective, and the number of jobs could be large, helper-objectives were randomly changed during the search and each helper-objective was used once. The "slots" for the currently used helper-objectives were titled *dynamic helper-objectives.* Dynamic helper-objectives were used simultaneously with the overall objective by a MOEA to solve the single optimization problem. Throughout the rest of this document *helper-objectives* are referred to as *helpers*.

Jensen used NSGA-II for the research although any MOEA can be extended to handle dynamic helpers. By optimizing individual job flowtimes and total flowtime simultaneously in a MOEA, the helper method finds better results than a simple GA that optimizes only total flowtime. Jensen theorized that the dynamic helpers assist the algorithm in escaping local optima through the creation and application of good genotypic building blocks that would have otherwise

been difficult to find. Because helpers need adequate time to build good building blocks during the search, "each helper was used for one period of the maximal length possible" (Jensen, 2004). Jensen did not provide direct evidence supporting a hypothesis on whether helper methods lead to an increased number of fitness improvements or larger steps in fitness improvement. The switching of helpers over time transforms the portion of the problem landscape searched by the GA. Through these changes, helpers assist in finding better solutions. A secondary effect of changing helpers is that the search is more diverse since solutions vary in fitness value with respect to the different objectives.

A small number of dynamic helpers appears to work best. "Experiments indicate that when many helpers are used simultaneously, the disadvantage of the bad moves (with respect to overall fitness) outweighs the advantage of escaping local optima" (Jensen, 2004). With a high number of helpers, the algorithm does not place enough emphasis on the main objective – the reason for doing the optimization in the first place. Jensen's analysis used each helper only once each run and in a random order. The re-use of helpers and the 'smart' switching off helpers are areas that have research potential. Perhaps methods can determine which helpers are critical and use them accordingly. Identification of areas of premature convergence may be possible with the incorporation of problem specific details.

Additional evidence was given in Jensen (2004) in demonstrating the effectiveness of using helpers on the JSSP and on the TSP. For the TSP, helpers were defined using a random division of the cities into two subdivisions. Since random divisions could be made multiple times, any number of helpers can be created. The total fitness for the helper dimension was the cost of all selected links to and from the selected cities in the helper. If a link from one city in the helper connected to another city in the helper, that link was counted twice for fitness. This approach does not take into account any details about relative distances between the set of cities selected.

The method does not group cities based on anything other than random draws and likely misses the opportunity to incorporate additional knowledge about the data associated with the cities.

In addition to the TSP details, Jensen (2004) introduced specific operators (*niche enforcement* and *in-breeding control*) to manage diversity and improve the performance of the MOEA on the JSSP. *Niche enforcement* forces each niche to have no more than a specified number of individuals. *In-breeding control* attempts to keep solutions with the same objective values from mating. Both diversity methods of *niche enforcement* and *in-breeding control* were found to be statistically important. An approach that incorporates details about the problem is likely a smarter one. Data for the JSSP may help govern smart use of the helpers, for instance.

Since the introduction of Jensen's helpers in 2003, two independent works have applied Jensen's helper concepts to problems. Greiner et al. (2007) used helpers to do multi-objectivization of a frame bar structure optimization. Jahne et al. (2009) mixed components of helpers with the concept by Knowles et al. (2001) for solving the TSP.

An optimization of frame bar structures using helpers was examined in Greiner et al. (2007). In addition to the overall goal of minimizing mass, the optimization used the number of different Cross-Section Types (CST) as a second objective. The number of CST was a static helper - unlike the original dynamic concept for helpers. Several different MOEAs were compared to the single objective GA using these two objectives. The results indicated that the dual objective methods outperformed the single objective methods. More significantly the research revealed that the helper methods may be much less sensitive to the rate of mutation because of their natural diversity mechanisms. With the added dimension, mutation rate did not have as large of an effect on solution quality.

Jahne et al. (2009) attempted to remedy weaknesses found in both Knowles et al. (2001) and Jensen (2004) for solving the TSP. Knowles et al. was criticized in Jensen (2003a) for problems of symmetry. Since Knowles et al. only subdivided the TSP once the solution quality was highly

dependent on a good identification of two cities that are far from each other in the solution space. Jahne et al. (2009) blended the concept of dynamic helpers with the concept of maximal problem division used by Knowles. The dynamic helpers were used to address the static and symmetry problems of Knowles' approach. In addition by keeping the division of the problem to two objectives, Jahne et al. (2009) attempted to address the issue in Jensen's previous work where there was an "unacceptable shift away of the search focus when additional objectives are used (Jahne et al., 2009)." This blended method was titled Multi-Objectivization via Segmentation (MOS). MOS uses the concept of creating many random divisions of the problem. These random divisions corresponded to helpers and were changed over time. Unlike Jensen's helper concept, MOS did not use the main objective in conjunction with the helpers but rather used pairs of helpers simultaneously. Three different metrics were used to create random sets of helpers. The *expected value of distances* metric attempts to divide the cities into two sections that are as far apart from each other as possible. The second metric, *standard deviation of distances*, divides the cities according to the relative bandwidth of each city. The third metric, *expected value of different neighbors*, attempts to divide the cities according to criticality. Each of the three proposed MOS approaches outperformed the basic helper and problem division approaches for the TSP problem. The MOS approach shows how various problem divisions that use problem specific knowledge can lead to better optimization techniques. Although the MOS approach studied three divisions for the TSP, it is possible that a better set of divisions exist. Jahne et al. (2009) provided a logical argument for splitting the problem in multiple halves but little supporting evidence was provided to back whether smaller divisions of the problem would be less or more effective for similar methods.

Scharnow et al. (2004) proved that the Single Source Shortest Path (SSSP) decomposition of the objective function into multiple objectives gave faster expected running times to find the optimal solution using evolutionary algorithms. The minimum spanning tree problem attempts to

find the minimum set of interconnected edges that visits all nodes in a graph. Neumann and Wegener (2007) showed that for some cases on the minimum spanning tree problem, multi-objective evolutionary optimization performed faster than the single objective evolutionary counterpart.

Brockhoff et al. (2007) studied several small plateau problems and showed through several proofs that adding objectives to some single objective problems can be harmful while in other problems multi-objectivization can help in finding optimal solutions in polynomial time with an exponentially increasing search space. The work used relation graphs to show how different objectives transform a problem into one that can be harder or easier. However, this work gave no general indication for how to identify when complex real-world problems benefit from multi-objectivization.

There is no current method to predict when a problem benefits from multi-objectivization although some have theorized it is related to decomposition properties of the problem. Since real world problems are large and contain many non-linear effects, it may be difficult to accurately predict when a problem does benefit from multi-objectivization. However, it is has been proven that some basic problems benefit from multi-objectivization. Empirical evidence indicates that some complex problems such as the TSP and JSSP also benefit from multi-objectivization (Knowles et al., 2001, Jensen, 2004, Jahne et al., 2009). A common thread between these problems is their ability to be separated into sub-sections.

3.2.1 An example of multi-objectivization

A hill-climber example is used here to show why a problem can be easier when solved with a multi-objectivization algorithm. Take, for instance, the basic hill-climber (a gradient decent method) which takes small steps in the direction of increasing fitness and accepts the solution if it is better. Hill-climbers are well known to not overcome local optima because of their local nature. Imagine the maximization problem depicted in Figure 14 where $0 \leq x \leq 5$ and the goal is to

maximize y_{total}. Futhermore, y_{total} is decomposable into the sum of the sub-objectives y_1 and y_2. The optimal fitness value of y_{total} is 3.0 and corresponds to an x value of 4.65. A single-objective hill climber with a sufficiently small step size will only find this optimal value if the starting x value is between 3.5 and 5.0. If a single hill-climber were started at a random x, it would find the optimal value approximately 30% of the time. That might be an acceptable rate, but problems can be envisioned where the rate is much lower than the example. If we instead used the single-objective hill climber on sub-objective y_1, the hill climber will never converge to the optimal value and at best it may pass the optimal as it converges to the local optima at $x = 4.75$. The same is true if a single-objective hill climber is used on sub-objective y_2; the algorithm will not converge on the global optima. In all three of these objective functions, the hill climber cannot overcome local optima. The multi-objective hill climber, however, can overcome local optima by switching the objectives as it climbs. Imagine a hill climber that starts at $x = 0$ working on y_1. The climber will look through increasing values of x until it stops at $x = 2.25$. No additional small changes in x cause increases in y_1 so at this point so the climber goes no further in y_1. However, if the climber could examine another objective such as sub-objective y_2, it would find that it is no longer at a local maxima there are x values close by that increase values of y_2. If the hill climber were to switch to climbing on y_2, the multi-objective climber would move to $x = 4.05$. At $x = 4.05$, there are no small changes in x that make y_2 larger, but if the hill climber could look back at y_1, it moves even further. This process of switching between sub-objectives allows the multi-objective hill climber to overcome local minima. If the multi-objective hill climber also recorded the best y_{total} value for x, in this example, it would find the optimal regardless of where the climber started on the landscape.

From this example we can see that there are instances where multi-objectivization benefits hill climbing methods. This is not always the case – examples can be concocted where multi-objective hill climbers are worse than single-objective hill climbers. Multi-objective hill climbers

must take care not to oscillate between several points. For this reason they may have more overhead than single objective hill climbers. Understanding when problems in general are best served by multi-objectivization has not been clearly established and is still under study.

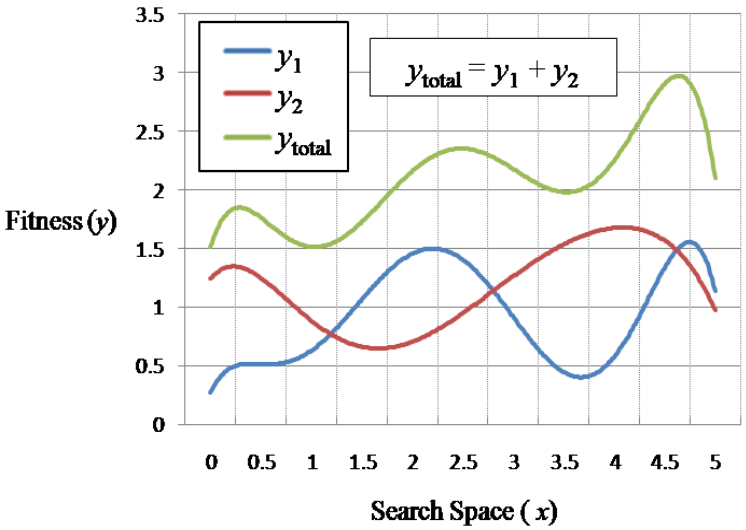


Figure 14 – A Multi-objectivization Example

3.3 Foundational development of helpers

Sub-costs, introduced in Wright (2001), were derived from the idea that it is likely useful to reward diversification steps that contain at least one element that improves the search in some way. In searching, it may be useful to make moves in a negative direction if it has the potential to help the longer term objective of the optimization (Wright, 2001). Wright gives the practical example of when a person is optimizing a schedule manually and gets to a point where no single move provides improvement. At this point the individual may consider a change that improves one aspect of the timetable even though the change has an overall worsening effect on the schedule. Hopefully, after several additional moves the overall schedule configuration improves. In the basset hound example, if scent and ground speed appears to be as good as possible, breeders may focus on increasing visibility even if it has the possibility of reducing one of the

two previous attributes. They can always breed more sensitive smell back into the line if it degrades too much. The principle idea behind sub-costs is that possible improvements in solutions to a problem can be identified and exploited in order to potentially improve the overall solution fitness. Wright used sub-costs in simulated annealing to increase the probability of accepting a worse solution in the event that it improved fitness in at least one sub-cost category. "Sub-costs are an idea closely related to helper-objectives" (Jensen, 2004). While the idea of helpers was clearly related to sub-costs, methods proposed to date that use helpers do not focus on picking helpers based on their ability to improve solution quality. This is a deviation from the original idea of sub-costs and the intuitive idea of guided evolution. Instead, helper research to date has used random draws to select new helper functions.

The smart definition of helpers and logical switching of helpers are at least two major ways in which the helper approach may be improved. As (Jahne et al., 2009) pointed out with the MOS approach, the definition of helpers based on problem specific knowledge can make a difference on the search results. How helpers are defined and calculate fitness may make broad differences in their effectiveness. For some problems, like the JSSP, natural helpers can be easily identified while for others like the TSP there may be no clear logical divisions. In addition to the definition of good helpers, creating good strategies for utilization of the helpers can be developed so that building blocks are exploited as they are being built. Helper selection approaches can utilize runtime and preprocessed data to tailor switching details to problem specifics. Furthermore, little research has made the case for how helpers creates benefits in an EA, outside of Jensen's original postulate that helpers assist in building good building blocks. More supporting evidence is required to understand why helpers perform the way they do so that underlying theory can be developed.

Chapter 4 Helper methods applied to the JSSP

This chapter summarizes effort accomplished in studying helper methods applied to the JSSP.

4.1 Specifics of the JSSP

A JSSP consists of *n* jobs and *m* machines. For each job there are at most *m* operations. Each of the operations has a specified machine, a specified job, and a specified processing time. A machine can process no more than one operation at a given time and no operation can be interrupted once it is started (no preemption). The goal of a JSSP optimization is to schedule the operations on the machines to minimize some measure of cost. The number of operations in the JSSP determines the problem size. Problem size can be expressed as a combination of the number of jobs and machines. For example, a 20x5 JSSP contains twenty jobs and five machines and at most 100 operations. The JSSP is notoriously difficult. With the exception of some small problem instances that have been shown to be solvable in polynomial time, the JSSP is NP hard (Garey et al., 1976).

The JSSP is defined below as adapted from (Vaessens et al., 1996). We use the objective function based on total flowtime. Define set $\mathcal{M}$ as containing *m* machines, set $\mathcal{J}$ contains *n* jobs, and set O contains *l* operations. For each operation $v \in O$ there is a unique machine $M(v) \in \mathcal{M}$ on which the operation must be processed, a unique job $J(v) \in \mathcal{J}$ to which the operation belongs, and a processing time $p(v) \in \mathbb{N}$. A set of binary relations, A, represents precedence relationships between operations; if $(v, w) \in A$, then w must be performed after v. Precedence relationships within a job, sometimes called the *technological constraints*, are the only relationships given in A. If $(v, w) \in A$ and there is no $u \in O$ with $(v, u) \in A \; and \; (u, w) \in A$, then $M(v) \neq M(w)$. A

schedule is a function $S: O \rightarrow \mathbb{N} \cup \{0\}t$ that for each v defines a start time $S(v)$. A schedule S is feasible if:

- $\forall v \in O: S(v) \geq 0$,
- $\forall v, w \in O, (v, w) \in A: S(v) + p(v) \leq S(w)$, and
- $\forall v, w \in O, v \neq w, M(v) = M(w): S(v) + p(v) \leq S(w)$ or $S(w) + p(w) \leq S(v)$.

Several different objective values can be used in a single objective JSSP optimization. Minimizing makespan is most frequently studied in the JSSP literature. Makespan is defined as the single longest completion time for all jobs: $\max_{v \epsilon O} S(v) + p(v)$. Rather than makespan this study minimized total flowtime as (similar to work by Jensen (2003a, 2004)). The total flowtime of the schedule is $\sum_{i=1}^{n} max_{v \ni J(v)=\mathcal{J}_i} S(v) + p(v)$.

Two common ways to represent a JSSP solution are as a Gantt chart or as a directed graph. In Figure 15, a three job and four machine schedule is shown in each of these two representations. The various colors in each representation correspond to different machines. In the Gantt chart time is on the horizontal axis, operations are the colored blocks and therefore operations to the left precede operations to the right. In the directed graph, the nodes are the operations. For the directed graph in Figure 15, the operation processing time is displayed in the node. The dashed arrows correspond with selected machine sequence links. These links are picked by the optimization from a set of possible machine links shown in Figure 16. The solid black arrows in both figures are the job sequenced links (given for the JSSP). In a valid schedule all operations have at most one incoming link from a previous operation in the job, one incoming link from a previous operation using the machine, one outgoing link to the next operation in the job, and one outgoing link to the next operation for the machine. The start time for any operation is the larger of the previous job operation and previous machine operation completion times. Since the JSSP does not allow preemption, the completion time is always the start time plus the operation time. Frequently a source and drain node are added to the directed graph representation to give a single

entry and exit point for all nodes. The makespan for this example is associated with job J_2 and has a value of 41 minutes. The total flowtime for this schedule, 94 minutes, is the sum of the completion times for all three jobs.

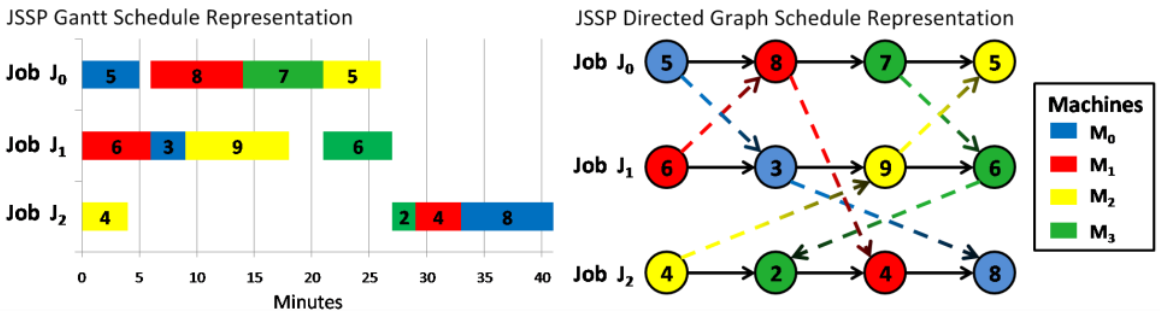


Figure 15 – JSSP Schedule Representations

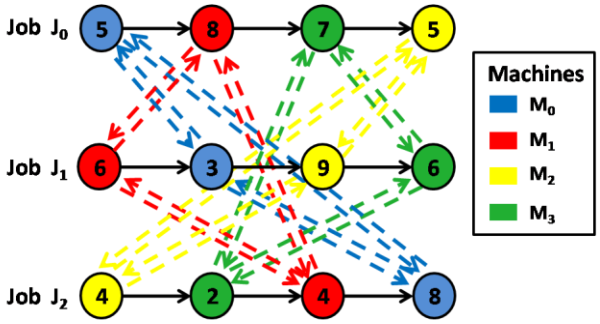


Figure 16 – JSSP With All Possible Machine Links

Figure 17 shows a sample active schedule for the swv12 problem described in Storer et al. (1992). The swv12 problem is a 50 job 10 machine problem with a total of 500 operations. The figure illustrates the complexity of the JSSP as later operations become increasingly more dependent on a set of earlier operations and dependencies. Each machine works on at most a single operation at a given time so no more than 10 operations can be processed simultaneously in the swv12 problem. The optimal solution for this problem will still have delays as 10 machines cannot process all 50 jobs simultaneously.

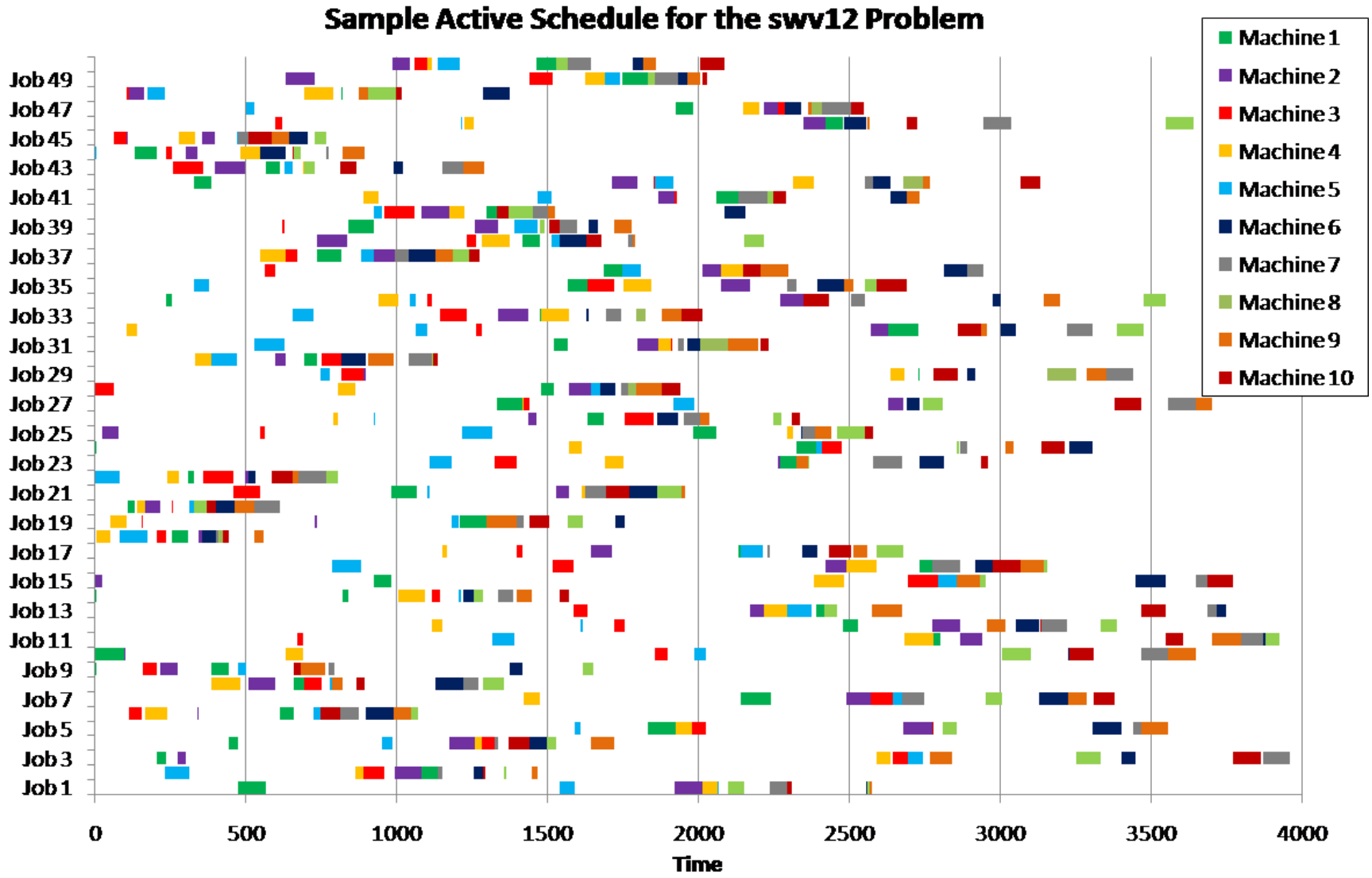


Figure 17 – A sample active schedule for the swv12 instance

4.1.1 JSSP solution methods

The JSSP has been approached by a variety of techniques. Methods include: simple greedy heuristics based upon differing priority metrics, relaxation techniques such as Lagrangian relaxation, branch-and-bound, tabu search, simulated annealing, GAs, hybrid approaches (Blazewicz et al., 1996), and shifting bottleneck procedures. GAs, branch-and-bound, tabu search, and simulated annealing techniques are common approaches to finding solutions to JSSP optimization problems. Refer to (Fisher, 2004) for an overview of Lagrangian relaxation. Chapter 7 of Wolsey (1998) has more details on branch-and-bound. Glover and Laguna (1997) is a comprehensive text on tabu search and Saidi-Mehrabad and Fattahi (2007) provides a recent JSSP example using tabu search. Kirkpatrick (1984) details the method of simulated annealing. The shifting bottleneck technique successively solves single machine problems for the JSSP and schedules the machine that is the largest bottleneck to the schedule (Adams et al., 1988). As the name implies, scheduling one operation can cause a different machine to become the next bottleneck so all single machine problems must be solved again after an operation is added to the schedule.

On the JSSP, GAs that do not incorporate local search techniques perform poorly when compared to tabu search and simulated annealing methods (Vaessens et al., 1996). However, when incorporating local search as a part of the global search process GAs are competitive at JSSP problems. GAs that incorporate local search are called *memetic* algorithms. The shifting bottleneck GA introduced in Dorndorf and Pesch (1995) is a memetic algorithm because it incorporates elements of the shifting bottleneck technique to locally improve solutions after they have been generated through crossover.

GAs without local search capability may perform poorly compared to simulated annealing and tabu search because the GA's global nature does not allow the GA to quickly focus on locally promising areas. In other words, the GA fails to find local improvements easily and as a result

spends much of its time trying to combine solutions that are obviously ‘poor’ in comparison. One reason the JSSP is a good candidate for multi-objectivization is because the technique may distinguish between solutions that are globally ‘poor’ and locally ‘poor’ so that the algorithm can find and recombine solutions that have complimentary features.

4.1.2 GA details for the JSSP

Some JSSP optimization details are specific to GA applications since the other approaches to the JSSP do not perform recombination. According to the survey (Cheng et al., 1996), there are nine basic ways in which solutions have been represented in GAs. Of these nine representations, the preference list-based encoding is most common (Cheng et al., 1999). A preference list assigns priorities to the different operations in the JSSP. Solutions with a list of preferences use a permutation-based representation for the genotype. These priorities are then decoded with the Giffler-Thomson (GT) schedule builder discussed below. The GT builder creates a schedule so that an individual’s fitness can be calculated. The highest priority item in a preference is typically the first item. The second item is the second highest priority and so on. Thus, an example three job, two machine solution {5,3,2,6,4,1} encodes a solution where the fifth operation is the most important, the third operation is next most important and so forth. Because the operations are structured by job, another common representation instead contains a set of job numbers where for each instance a given job number appears it that job number corresponds to the next operation in the job. The number associated with each job J_i appears at most m times. The example three job two machine solution above would be encoded as {3,2,1,3,2,1} where the first 3 is the first operation in job 3 and so on.

Figure 18 gives a pseudo-code description of the GT schedule builder. The GT builder ensures all built schedules are active and attainable. Active schedules are those schedules where no operation can start earlier without delaying another operation. It is possible to generate the

optimal solution using the GT builder since the optimal schedule must be active and the GT builder can construct all active schedules (Giffler and Thompson, 1960).

```
let O be a collection of operations
let t be a time
let o and p be operations
add the first operation for each job into O
while O is not empty
{
   set t to the earliest possible end time for all operations in O
   pick a random operation p from O which has end time t
   set o to the best priority operation in O that has the same
        machine as in p and that has start time prior to t
   add o to the schedule
   remove o from O
   add the next job operation that follows o into O
}
```

Figure 18 – GT Schedule Builder Pseudo-code

There are a variety of permutation-based crossovers, such as Cycle Crossover (CX), Partially Mapped Crossover (PMX), and Order Crossover (OX) (Wiese and Glen, 2003). Order Crossover OX operators are a general family of crossover methods that attempt to maintain the order of values in the genome when handling crossover (Goldberg, 1989c). To create offspring OX operators take a string of integers from one parent and attempt to insert the string into a copy of another parent. Various OX operators differ from each other on how they repair the final solution and how they select the donor string. OX operators are often used for TSP and JSSP solution recombination since common representations for solutions use a permutation-based genotype.

Generalized Order Crossover (GOX) is a specific order crossover that inserts a substring from the donor parent into a receiving parent. GOX maintains the order in which jobs appear for the donated substring. In GOX the newly created child maintains solution feasibility because the operator eliminates duplicate genes (jobs) that correspond with the donor string. GOX contains an implicit level of mutation because eliminated duplicate genes will not always be the original

sequences. An introduction and complete description of the GOX operator can be found in Bierwirth (1995). GOX has been compared to other crossover operators such as Generalized Position Crossover (GPX) and Precedence Preserving Crossover (PPX) and was found to perform favorably when compared on different JSSP problems (Mattfeld, 1996).

There are a variety of mutation operators that can be used on preference list-based solutions. Three such operators that work on these solutions are Order Based Mutation (OBM), Swap Based Mutation (SBM) and Position Based Mutation (PBM) (Jensen, 2001). Trade-offs regarding the strength of the mutation versus the ability to keep good solution qualities intact are inherent to different mutation operators. OBM swaps the value of two random positions and is the strongest of the three in terms of changing solution makeup. PBM takes a job at a random position and inserts it at another position in the solution and is the middle of the three mutation operators in terms of strength. SBM swaps the values for two adjacent positions and is the weakest of the three in terms of changing the solution. SBM is the special case of both OBM and PMB where the two random positions are adjacent. PBM performs favorably compared to OBM and SBM (Mattfeld, 1996).

4.2 Helpers applied to the JSSP

The remainder of this chapter studies strategies for using helpers with MOEAs in solving the JSSP. New helper based concepts are shown to outperform previous helper methods. Evidence indicates that helper methods may break dimensions of epistasis by evaluating a component of epistasis individually rather than collectively. Results indicate that multi-objectivization has the potential to work past certain dimensions of epistasis by changing the perceived value of solutions.

A non-random selection and sequencing of helpers has been given no known emphasis to date and little analysis to date has been placed on deciding good helper sizes. It is suspected that both helper size and helper sequence matter in the search. Helpers that are too small likely

account for too little sectioning of the problem while helpers that are too large become highly complex making them hard to interpret for the GA. Large helpers may have the same weakness as the simple GA – the algorithm cannot decide what it needs to focus on, making search difficult.

Optimization of helpers is further explored in this chapter. Creating good strategies for the utilization of the helpers may ensure that building blocks are created in a timely manner and are exploited as they are being built. The experimental objectives of our study are to:

- investigate the effects (if any) that helper sequence has and what strategies may be promising when picking helper sequence,
- investigate the appropriate size for helpers and what effects size of helpers may have on the search,
- investigate how helpers accelerate search, and
- investigate the nature of the non-dominated front size and the tuning of helper-objective algorithms.

Implementation details for the analysis follow (Jensen, 2004) to best represent Jensen's helper methods so that direct comparisons can be made. Details that differ from Jensen's implementation are identified as they are described. Code to support this analysis was implemented in C++.

NSGA-II was used to handle the multiple-objective aspects of the search. As in NSGA-II, binary tournament selection was used for parent selection and survival selection was handled collectively on both the parents and children ($\lambda+\mu$). As Jensen indicated, other MOEAs could have performed the job that NSGA-II is doing here, but NSGA-II was selected to stay consistent with Jensen's original implementation.

Jensen's helper concept called for the MOEA to swap in different helpers over time as "using many helpers simultaneously is probably more harmful than beneficial" (Jensen, 2004). A given run can have h simultaneous dynamic helpers 'slots' where $h \epsilon 1 \ldots n$. Good values of h are generally low (one or two) as otherwise "the focus of the search will be shifted away from the primary objective to an unacceptable degree" (Jensen, 2004). This implementation used only one

and two dynamic helper slots as findings by Jensen indicated that three or more helpers were detrimental to the search.

Individual solutions are represented as an ordered list of preferences for the different operations. The number associated with each job J_i appears at most *m* times in a given genome and these numbers represent the different operations associated with the job. For example an individual for a 2x4 problem might be encoded as (1,2,2,2,1,1,2,1) where the first "1" is the first operation of J1, the second "1" is the second operation for J1 and so forth. The earlier an operation appears in the encoded string, the higher its priority. Operations to the left are always higher priority than operations to the right. From the individual's genome, schedules were built using the well accepted Giffler-Thompson (GT) schedule builder (Giffler and Thompson, 1960).

Crossover was handled using the GOX operator. PBM was accomplished on every solution generated by GOX. Jensen studied the rate of PBM and GOX crossover and determined for the JSSP problems in Table 9, always applying GOX and PBM produces the most favorable results (Jensen, 2004).

The benchmark JSSP problems used for this analysis are indicated in Table 9**.** Problems with the *la, ft,* and *swv* descriptors are from Lawrence (1984), Giffler and Thompson (1960) and Storer et al. (1992) respectively. Data sets for the problems were obtained online from J. E. Beasley's OR Library (Beasley, 1990).

Table 9 – JSSP problems analyzed

Size	Instances	Size	Instances
10 x 5	la01, la02	20 x 10	la26, la27, swv01, swv02
15 x 5	la06, la07	30 x 10	la31, la32
20 x 5	la11, la12, ft20	15 x 15	la36, la37
10 x 10	la16, la17, ft10	20 x 15	swv06, swv07
15 x 10	la21, la22	50 x 10	swv11, swv12

Jensen specified the dynamic helper swapping interval to be $D = \frac{hT}{H}$ where T is the time budgeted for the run and H is the total number of helpers used. In the JSSP, H is typically the number of jobs n. However, in some problems, n does not divide evenly by h which causes an issue in the final time period where the number of remaining unused helpers does not equal the number required h. When n does not divide evenly by h, Jensen did not specify how to calculate the total number of helpers to be used (H). For instance, in a fifteen job problem with two dynamic helpers, the first seven periods use the two random helpers selected in the specified random sequence. However, in the final period only one dynamic helper remains available even though the method calls for two. In order to give each job a chance at being a helper, and to keep the number of objectives constant for the whole run, one or more of the first used helpers were reused in the final period to bring the number of dynamic helpers used in that period up to h. For instance, in a fifteen job problem with two dynamic helpers, the first seven periods use the random helpers selected in the specified random sequence. For the eighth and final period, the remaining random helper is only one of two objectives. To fill the remaining dynamic helper slot with a helper, the first helper from the original randomized helper sequence is used. This implies the number of periods for helpers $b = \left\lceil \frac{n}{h} \right\rceil$ where ⌈ ⌉ is the ceiling function which rounds a number up to the next highest integer. The total number of helpers is $H = bh$. This clarification does not affect the one dynamic helper cases because n always divides evenly by one. For the two dynamic helper cases, this clarification only affects problems with an odd number of jobs: la06, la07, la21, la22, la36, and la37.

Unlike Jensen's 2004 paper that uses CPU time to assign a computational budget, this analysis assigned budget based on the number of fitness evaluations. There were several reasons for the change in budget focus to fitness evaluations. First, unlike Jensen's work, the time order for the algorithms evaluated here are roughly equivalent. Hence, comparison based on fitness

evaluations is fair. Second, fixing the fitness evaluations ensures that the number of generations do not fluctuate with the algorithm's runtime performance. The runtime is variable due to the stochastic process of building schedules, performing crossover, and calculating fitness. Fixing the number of generations allows direct comparisons with reported statistics rather than requiring recoding of previous methods. This was not possible with the comparison to Jensen's methods here because source code details are virtually guaranteed to differ which can result in runtime performance differences.

There is a non-linear increase in the amount of CPU time required to build schedules for larger problems so using a number of fitness evaluations scaled by problem size rather than a scaled CPU time ensures that the number of generations run for the large problems continues to increase with problem size. Since for this analysis the number of fitness evaluations are constant for a given problem, dynamic helpers were switched every g generations where $g = \frac{G}{b}$ and G equals the total number of generations.

G was set as two times the problem size ($2nm$). A population size of 100 was used in all problems as Jensen's previous works used that population size. Because Jensen indicated that one dynamic helper problems ran an average of 20408 evaluations, all problems were assured a minimum of 20,000 fitness evaluations.

4.2.1 Shortest Job First (SJF) helper sequencing

Jensen stated that "the sequence in which helpers are used may have an influence on the search, but since we have no way to know which order is the best one, a random ordering … was used" (Jensen, 2004). However, it may be possible to determine helper sequence based on knowledge about the problem. For any single job in the JSSP, the minimum possible job flowtime is the sum of all operation times associated with the job. This lower bound assumes there is no lag in the schedule caused by other jobs. Selecting jobs with the lowest minimum possible flowtime first may be a better strategy than random selection. It is well known that for the single machine

problem, sequencing jobs by lowest operation time first creates a minimum total flowtime schedule (Hopp and Spearman, 2001). In general a short operation waiting for a machine while a long operation takes place on that machine produces a larger total flowtime than having the longer one wait on the short one. Since short jobs by definition must have some short operations, they are likely to produce lower total flowtime if they are in an earlier part of the schedule. It is highly unlikely that all jobs attain their minimum possible flowtime in the best schedule since this implies that there are no conflicts between jobs that result in a delayed operation. However, the job minimum value does give an estimate of the extreme point to which the helper can move.

Shortest Job First (SJF) helper selection sequences helpers from shortest to longest based on their job's minimum possible flowtime. In the event that two or more jobs have the same minimum possible flowtime, the SJF method randomly picks the sequence of those jobs. Selecting helper sequence based on minimum job possible flowtime may have a positive influence on the MOEA's ability to improve solution quality as the MOEA is essentially guided in optimizing what is most important first. Preliminary results of the study of the SJF selection method were previously reported in Lochtefeld and Ciarallo (2010).

Random helper selection as specified in Jensen (2004) was implemented and compared to the SJF helper selection method. Jensen established that greater than two dynamic helpers were less effective than one or two dynamic helpers for the set of problems in Table 9. Therefore this comparison was carried out for both one and two dynamic helper cases. All problems were run for 200 random replications per method and number of dynamic helpers to help establish statistically significant comparisons. Table 10 displays a summary of the results for the experiment.

In all problems, the SJF helper selection approach found lower best values on average compared to its random counterpart. Based on the paired student's t-test in 16 of the 24 problems the SJF one-helper selection had statistically lower mean best values than the random one-helper

selection method. For the two-helper cases, the SJF helper selection method was shown to be statistically better than the random selection method in 21 of the 24 problems. These results clearly establish that the SJF helper selection method is superior to random helper selection.

The percentages from best known value for the random helper approach appear to be relatively similar to those reported by Jensen for the random helper selection cases. For example the reported percentage from best known value for la01 for the one-helper and two-helper cases in Jensen (2004) were 2.92% and 2.47% compared to the 3.43% and 2.74% found here. The 20,000 fitness evaluations used here are likely still lower than the number of fitness evaluations used by Jensen for the la01 problem which likely explains why the percentages from the best known value in la01 are slightly larger in these results. Larger problems take increasingly longer CPU time to build schedules. Due to the change in using a fixed number of fitness evaluations based on problem size, this analysis observed better values on average than those previously published for the large problems. CPU versus fitness function scaling also explains why many new best solutions were found for the bigger problems as more fitness evaluations were likely devoted to them. This analysis found new best known total flowtimes for 15 of the 24 problems.

In general it appears that for the SJF helper selection method, larger problems benefit more from using two dynamic helpers and smaller problems do better with one dynamic helper. As the number of objectives in the MOEA increases, the focus on the optimization of the main objective decreases since there are more objectives to consider. Keeping the number of objectives small is important as it emphasizes what really matters – solutions with low total flowtime in this case. This raises the question: why are many of the larger problems benefitting from the two dynamic helpers if more focus is taken away from the main objective?

Table 10 – Average best found solution as a percentage from the best known solution

Problem	Best Known Solution	Random 1 Dynamic Helper % from Best Known	SJF 1 Dynamic Helper % from Best Known	Random 2 Dynamic Helper % from Best Known	SJF 2 Dynamic Helper % from Best Known
la01 (10x5)	4832	3.426*	2.297+	2.741*	**2.190+**
la02 (10x5)	4468	3.260*	2.903+	3.752*	**2.875+**
la06 (15x5)	8694	5.894*	**4.941+**	7.615*	6.251+*
la07 (15x5)	**8196**	6.076*	**5.142+**	7.573*	6.933+*
la11 (20x5)	14801	5.497	**5.216**	8.072*	7.791*
la12 (20x5)	**12484**	6.677*	**5.563+**	9.260*	8.507+*
la16 (10x10)	7393	5.082*	4.230+*	5.054*	**3.795+**
la17 (10x10)	6555	3.830*	3.555	3.771*	**3.375+**
la21 (15x10)	**12953**	5.510*	**4.400+**	5.864*	4.687+
la22 (15x10)	12106	4.904*	4.581	5.030*	**4.526+**
la26 (20x10)	**20234**	5.447	**5.290**	6.601*	6.581*
la27 (20x10)	**20844**	5.817*	**4.917+**	6.731*	6.651*
la31 (30x10)	**39007**	5.642	**5.321**	7.159*	6.869+*
la32 (30x10)	**42523**	4.774*	**4.039+**	6.177*	5.623+*
la36 (15x15)	17073	5.209*	5.052*	4.634*	**4.309+**
la37 (15x15)	17886	5.279*	5.114*	5.512*	**4.725+**
ft10 (10x10)	**7501**	8.072*	6.822+*	7.856*	**5.838+**
ft20 (20x5)	**14279**	9.837*	**7.616+**	12.400*	9.689+*
swv01 (20x10)	**20688**	12.074*	9.768+*	11.570*	**8.764+**
swv02 (20x10)	**21682**	10.111*	8.334+	9.880*	**8.058+**
swv06 (20x15)	**28863**	8.571*	7.022+*	7.536*	**5.960+**
swv07 (20x15)	**27385**	9.720*	7.937+*	8.358*	**6.623+**
swv11 (50x10)	**108842**	9.330*	8.912*	8.966*	**8.032+**
swv12 (50x10)	**109128**	10.151*	**8.737+**	9.986*	9.036+
Average	**-**	**6.675**	**5.738**	**7.171**	**6.154**

Flowtimes in bold in the 'Best Known Solution' column indicate a new best flowtime was found in the analysis. Best flowtimes prior to this analysis were reported in Jensen (2004). Percentages in bold indicate the lowest percentage for all four cases. '*' values indicate statistically higher average best flowtime as compared to the lowest percentage for all four cases. '+' values indicate statistically lower averages between the SJF and random method for the similar number of dynamic helpers. Statistical significance was tested at the $\alpha = 0.05$ level. Mean best flowtime values were compared with a two-tailed student's T test assuming unequal variance.

The relative size of the helper may be the reason the one-helper cases performed worse than the two-helper cases for the larger problems. For a ten job problem, a single helper constitutes 1/10th of the number of operations. In contrast for a fifty job problem, a single helper constitutes 1/50th of all operations. It is possible that helpers have to be of sufficient size to be effective. A

helper that is too small may not reduce significant epistasis. A helper that is too large may have too many internal interactions and contain too much epistasis making it nearly as hard as the original problem. Since adding objectives pulls the MOEA's focus away from the original objective, it may be better in the larger problem to create helpers that consist of more than a single job. Helper fitness in this case is the sum of all the flowtimes of jobs in the associated helper. This strategy places more of the emphasis on the overall objective since it uses only a single helper. Additionally it allows sizable helpers to assist the search. However, if there is significant interaction between the jobs within the helper, epistasis within the helper can be large. High epistasis within helpers may hinder the algorithm as internal epistasis makes improvements within the helpers difficult to discover.

The algorithms were also evaluated in terms of their runtimes. No appreciable increase or decrease in run speed was observed when using the SJF helper selection method. For determining the selection sequence for helpers both algorithms are $O(n/h)$. The real runtime difference between the two falls in the margin of error in measurement since fitness evaluation and survival selection are considerably higher order and have some variability. Operations related to the helper selection contribute only a minute fraction of the total run time.

4.2.2 Investigating how helpers accelerate search

In the process of obtaining the results above showing the benefit of using helpers, a number of questions arose regarding how helpers accelerate the search. Why is it that helpers can beat single objective GAs while taking the focus away from the main objective? Why does the SJF selection method work better with one helper in smaller problems and better with two helpers in larger problems? What is the "right" helper size? A pursuit of answers to these questions may help to forward understanding of multi-objectivization methods in EAs.

The average of all best flowtimes across replications for each generation is plotted for the various problems to determine the behavior differences between the SJF and random helper

selection methods. Figure 19 shows the la01 and la06 problem results using the two different helper selection methods for both one and two dynamic helpers. The random sequence helper selection appears to match the shape of data reported in Jensen (2004). For instance, in la01 Jensen showed that the two-helper method overtook the one-helper method in later generations which is also the case here. Furthermore, the shapes of the random helper selection curves in both charts appear consistent with those reported previously. These face value results help to validate that Jensen's helper selection method was correctly implemented. For all cases the SJF helper selection method establishes an early lead when compared to its random counterpart. In the la01 two dynamic helper and the la06 two dynamic helper cases the early lead is somewhat diminished later in the run. In other comparisons, such as the difference between the random one dynamic helper and the SJF one dynamic helper method, the early lead was maintained throughout the run. Effective helpers used early by the shortest job helper selection had some marginally smaller benefit when used late by the random selection method. Strong early improvement was established for the SJF helper selection because moving short operations to the start of the schedule generally lowers overall flowtime. In some problems and helper definitions, these moves become increasingly difficult to accomplish later in the search because as the population starts to converge it can contain less solutions that are able to viably improve fitness. These biased populations are generated naturally by an EA as convergence in the genotype starts to occur. As the GA proceeds, the population becomes increasingly biased by earlier decisions and the helpers are not as effective on a biased population because they cannot introduce new genetic material. Since early SJF helper selection gains were never fully overcome by the random selection methods throughout any of the runs, it is likely that the conclusion that SJF is a better strategy for helper selection is not highly sensitive to the number of fitness evaluations used in the analysis. Thus helpers given several more or several less generations will not likely impact the conclusion that helper sequence makes a difference on the quality of the search results. The

results from these experiments clearly establish that the sequence of helpers is important. SJF helper selection provides focus so that the GA optimizes the best components first.

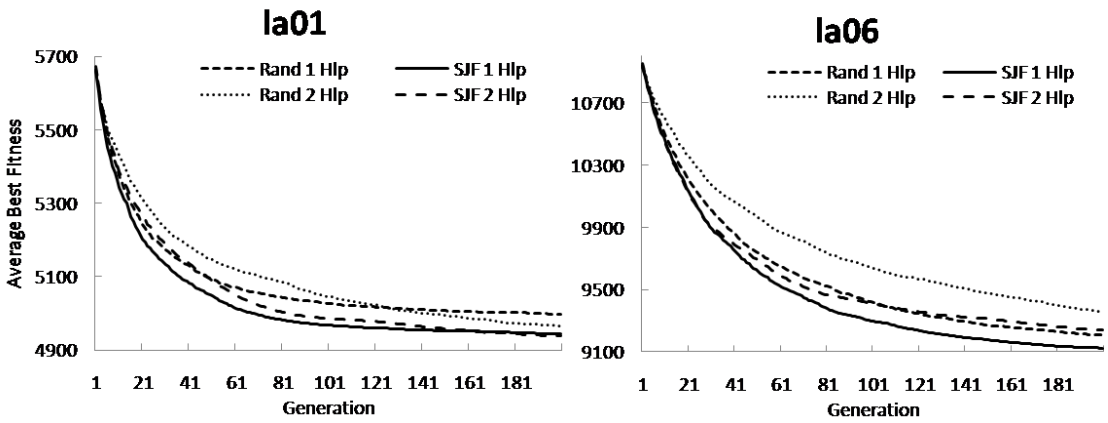


Figure 19 – Average Best Fitness by Generation for la01 and la06 Problems

Some relative gains made in early generations by SJF sequencing are never recovered by the random helper sequencing in later generations. Because both methods use the exact same number of helpers and the same number of fitness evaluations it might be expected that the final result of the optimization would be equivalent with SJF having an early lead and losing that lead by the end of the run. However, Figure 19 shows no signs of 'catch-up' for the one dynamic cases and limited 'catch-up' in the two dynamic helper cases. Inspection of the other 22 problems not depicted in Figure 19 showed similar phenomena where the SJF helper selection gained an early lead and often maintained much of that lead throughout the run.

Swapping helpers offers the GA an opportunity to find better solutions because of partial removal of at least one dimension of epistasis and/or deception. Epistasis can cause difficulties for a GA because the GA cannot identify the true value of fitness for solutions with good building blocks since epistasis can hide portions of that value. Epistasis between jobs can be broken by using a dynamic helper from the set of jobs that are cooperating. With helpers, one or more jobs

are no longer required to cooperate with the other jobs to express higher solution fitness. For example, imagine that two jobs are interacting in certain portions of the population and a single step away from that interaction causes poor fitness solutions but two steps away leads to better solutions. It may be difficult for any one solution improvement to be pursued because of this interaction. However, when one of those two jobs is swapped in as a helper, new solutions are generated in that job's dimension in the multi-objective process thus supporting creation of better building blocks.

It has been argued that deception is best viewed as a dynamic phenomenon that changes with the population members and problem constraints (Grefenstette, 1993). Epistasis, with its similar properties, is likely also best viewed as dynamic. The fact that the random helper selection method rarely 'catches-up' with the SJF method may be due to the SJF method breaking certain instances of epistasis early and breaking it early gives more progress than breaking it later in the optimization process. Search bias generally gets stronger the longer a search progresses and influencing it early can give better results than influencing it late. A deeper investigation into the ability of a helper method to break epistasis is warranted to determine if this hypothesis is correct.

The variance of best found solutions was also explored to see if there are substantial differences between the two methods. Instead of reporting standard deviations, values were normalized to account for differences in the problems so that meaningful averages could be drawn across all problems. Numbers reported in Table 11 are the standard deviation of best found solutions divided by the average best found solution for that case. Jensen reported that "For all experiments the standard deviation lies between 1.0% and 3.0% of the average best performance" (Jensen, 2004). Here the range of values is similar: between 0.95% and 3.15% with the two dynamic helper methods having significantly lower variances than the one dynamic helper cases. This is likely caused by the longer period of time between helper swaps which keeps the MOEA working with a fixed set of objectives and results in some search convergence that lowers overall

diversity. SJF helper selection generally has a lower variance than random helper selection. The path through the optimization is more deterministic in SJF – fewer random calls are used to determine helper sequence. Overall the SJF method with two dynamic helpers had the lowest variability. Low solution variability did not necessarily correspond with good average solution performance so few solid conclusions about the quality of algorithms can be made from the examination of variance.

Table 11 – Helper selection variability as a percentage of standard deviation divided by average best solution

Problem	Random 1 Dyn. Hlp. % StdDev / Ave. Best	SJF 1 Dyn. Hlp. % StdDev / Ave. Best	Random 2 Dyn. Hlp. % StdDev / Ave. Best	SJF 2 Dyn. Hlp. % StdDev / Ave. Best
la01 (10x5)	2.103*	1.464+	1.606*	***1.326*+**
la02 (10x5)	1.530*	1.244+*	1.532*	***0.951*+**
la06 (15x5)	2.162*	*1.692*+*	1.685*	**1.368+**
la07 (15x5)	1.810*	*1.691**	1.530	**1.424**
la11 (20x5)	1.843*	*1.727*	1.533	**1.510**
la12 (20x5)	1.933*	*2.130**	1.659	**1.638**
la16 (10x10)	1.734*	1.306+*	1.884*	***1.044*+**
la17 (10x10)	1.512*	1.532*	1.425*	***1.072*+**
la21 (15x10)	1.743*	*1.648*	1.623	**1.505**
la22 (15x10)	1.574	1.616	1.600	***1.453***
la26 (20x10)	1.495+	*1.731**	1.486	**1.461**
la27 (20x10)	1.765*	*1.796**	1.456	**1.426**
la31 (30x10)	1.586*	*1.538**	**1.222**	1.408
la32 (30x10)	1.596*	*1.488*	1.489*	**1.279+**
la36 (15x15)	1.479*	1.409	**1.265**	*1.296*
la37 (15x15)	1.426	1.443	1.491*	***1.272*+**
ft10 (10x10)	2.719*	2.019+*	2.494*	***1.713*+**
ft20 (20x5)	2.841*	*2.499*+	2.456	**2.348**
swv01 (20x10)	3.031*	2.971*	2.489	***2.312***
swv02 (20x10)	3.150*	2.644+*	2.197	***1.969***
swv06 (20x15)	2.543*	2.179+*	2.258*	***1.859*+**
swv07 (20x15)	2.515*	2.447*	2.231	***2.062***
swv11 (50x10)	3.058*	3.031*	2.224	***2.046***
swv12 (50x10)	2.778*	*2.968**	**2.138**	2.222
Average	**2.080**	**1.928**	**1.806**	**1.585**

Values in italics indicate the best performing algorithm for the problem as found in Table 10. Percentages in bold indicate the lowest percentage for all four cases. '*' values indicate statistically higher percentage as compared to the lowest percentage for all four cases. '+' values indicate statistically lower variability between the SJF and random method for the similar number of dynamic helpers. Statistical significance was tested at the $\alpha = 0.05$ level with F-tests.

To outperform a standard GA, helper methods need to find some combination of a larger number of improvements and/or larger sized improvements in fitness. The ft10 problem was examined using the one dynamic helper methods to determine the extent that these two rates change. The fitness value of the best found fitness was recorded for each replication and each generation. Two measures were extracted from this output. The average difference between the best fitness from one generation to the next generation is an indication of the average magnitude of solution improvement. The number of replications finding a better solution for a single generation divided by the total number of replications approximates the frequency that a generation creates better solutions. These two metrics are not perfect because more than one new best solution could be found in a given generation. If more than one new best solution is found in a generation, the estimate of the frequency of finding better solutions will be lower than its true value and the estimate of average solution improvement will be higher than its true value. Both the random helper method and SJF method will have similar bias if they have relatively similar rates of finding better solutions.

To overcome noise in the results and clarify trends regarding helper function behavior, the ft10 problem was run for 20,000 replications for each selection method. Average results from the runs are presented in Figure 20. The left graph in the figure is the average improvement in fitness generated by a given generation. The right graph is the percentage of replications that produced an improvement by generation. The overall trend of both curves is exponentially decreasing – this find is consistent with the fundamental understanding of how GAs work: Evolutionary algorithms tend to find many larger improvements early, while later they find smaller improvements at a less frequent rate. For the graph of the average improvement over time, little difference between the two dynamic helper methods is evident. Furthermore, no periodic data appears to be present at the helper transitions which occur 10 times through the run at generations 20, 40, 60, and so on. However, for the frequency of solution improvement, it can be seen that there are periodic

changes at many of the helper transitions. Generations 21, 41, and 61 appear to have a marked increase in the number of better solutions found. These two graphs indicate that while helper methods may not increase the average solution improvement, they can increase the probability of finding solutions that have improvements in overall fitness. For the ft10 problem, the SJF method finds significantly more improved solutions earlier in the run but later the random helper selection method finds marginally more solutions, thereby accounting for the lack of 'catching-up'.

MOEAs like NSGA-II reward solutions with fitness that are better than other solutions in at least one dimension of the objective. Before the helper associated with a particular job is swapped in, the job is only represented in the population through the overall fitness objective. The job is one of many jobs that are represented in overall fitness and as such the MOEA, looking at overall fitness is not as sensitive to the individual job's flowtime. However, after the helper is swapped in, the MOEA becomes keenly focused on the job's fitness. This focus allows the MOEA to recognize good solutions in that dimension and gives them a higher chance to procreate. The change in recognition of new 'good' solutions manifests itself by the increases seen at generations 21, 41, 61 and so on in the right graph in Figure 20. Thus, in addition to combating epistasis a helper's role can be viewed as providing short-term focus for the MOEA so that the EA can select solutions with a better chance at improving overall fitness.

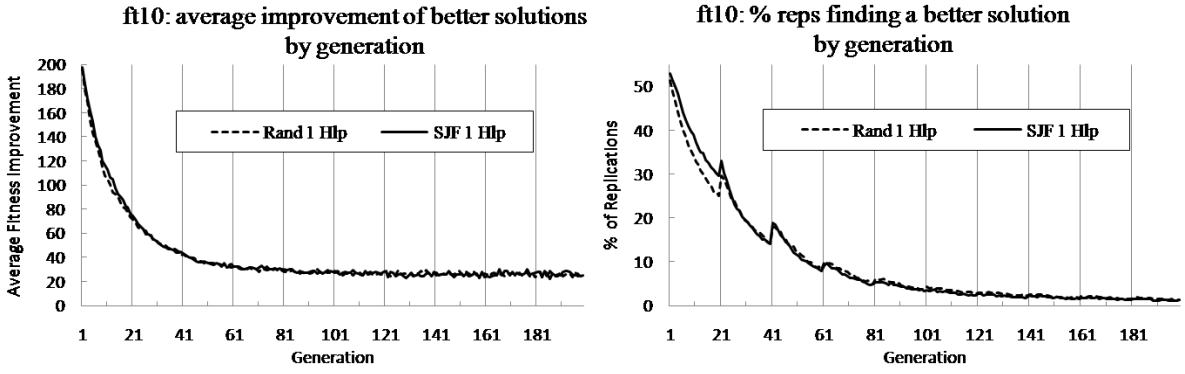


Figure 20 – Frequency and magnitude of improvement in best current fitness by generation for ft10 instance

4.2.3 Exploring helper size

Past research has not examined the size of helpers. Previous works on the JSSP only used one job per helper(Jensen, 2004, Jensen, 2003a). Jobs appear to be a natural division for the JSSP although there are many other ways in which helpers could be divided. Jahne et al. used problem divisions that were half the size of the TSP reasoning that a maximal division would give best results (Jahne et al., 2009).

We next examine the appropriate helper size. Increasing helper size will increase the difficulty of optimizing on the helper while reducing helper size may render the helper trivial and ineffective at assisting the overall search. Helper size was examined using a single dynamic helper slot because our results are consistent with (Jensen, 2004) that the one dynamic helper method was best overall for the 24 problems studied. The fitness for each helper was the sum of all flowtimes of jobs included in the helper. To vary the size of helpers, individual helpers were defined as a combination of one or more jobs. Specifically, the number of Jobs Per Helper (JPH) is indicated by the variable p. For this experiment p could take the values of one, two, five, ten, and $\lceil n/2 \rceil$. Since larger helpers use more jobs and jobs appear only once, larger helpers run for a longer period of time thereby, perhaps, compensating for the larger complexity.

Helpers were switched every g generations where $g = \frac{G}{\lceil n/p \rceil}$. In the 15 job problems n does not always divide evenly by p. For these cases the final helper iteration does not have exactly p jobs remaining to use. Instead, the remaining jobs (however many there were) were used as the final helper in a run. For instance let us examine 15 job problems like la06 and la07. With 2 JPH, the first 7 helpers use the first 14 jobs 2 at a time and the final helper uses the single remaining job. Since we established that the SJF method dominated random helper selection, this experiment only used the SJF helper selection method. To implement the SJF method for helpers with multiple jobs, the jobs were assigned to the helpers in a greedy fashion such that the first helper obtained the first p shortest jobs and the next helper obtained the next p shortest jobs and so on.

Other settings regarding operators, parameters, and number of fitness evaluations in this experiment were consistent with those specified in the experiment above.

Table 12 shows results of the experiment for six different JPH cases for the 24 problems in Table 9. Values for the different case and problem combinations are reported as the percent of the average best found solution for that case-problem combination divided by best known solution for the problem. Lower percentages indicate that the model, on average, got closer to the best known solution. This experiment found new best known solutions for 7 of the 24 problems. Helper size clearly can have an influence on the search. For the one dynamic helper cases, 18 of the 24 problems benefited from less than the maximal problem division of $\lceil n/2 \rceil$. Furthermore, of the six problems that benefited from the largest helpers studied, the average problem size was 95.8 operations. In contrast the problems that did best with the smaller than half sized helpers had an average of 223.6 operations. Clearly large problems do not benefit as much from maximal helper divisions. Helpers that are too large may be almost as difficult to search as the original objective since the helpers will have a high number of internal interactions. Thus, large helpers generally provide less benefit to the algorithm than smaller helpers. On average, the one dynamic helper with one JPH was best for the 24 problems studied. This consistent with the hypothesis that a main benefit of helper-based methods is the breaking of epistasis.

A single-job dynamic helper allows the algorithm to create improvements in that helper dimension at the expense of the main objective. For two-job dynamic helpers, each helper is allowed to create improvements in that helper at the expense of the main objective. Additionally, the helpers are allowed to make improvements at the expense of each other. This third level of interaction between the helpers can remove epistasis between the two dynamic helpers. Thus, two dynamic helper methods should generally remove more epistasis than the one helper methods since two dynamic helper methods not only remove some interaction between each of the two current helpers and the overall fitness, they also remove some of the interactions between the two

helpers themselves. A measure of the difficulty created by interactions in the problem may provide benefit by being able to determine an appropriate number of dynamic helpers. However as discussed before, literature on direct measures of epistasis indicate that determining the number and intensity of significant interactions from sampling the problem space can be complex, time-consuming, and challenging.

It is possible that we can determine the significance of interactions partly based on the problem structure and data. The best solutions for 5 of the 6 swv problems in Table 12 were achieved with a two-job dynamic helpers and the remaining swv problem is only marginally better with the one-helper case. The best solutions to these six problems tend to be found with two dynamic helpers because the swv problems are structured differently than the la and ft problems. In the swv problems, exactly half of the machines appear in the first half of the operation sequence associated with every job. This structure is not present with the la and ft problems where machines generally appear throughout the required job sequence.

For all problems in Table 10 where the two dynamic SJF helper method is shown to be statistically better than the one helper SJF method, in Table 12 the two dynamic helper is also better than any of the different one helper sizes examined. This may indicate that tuning the number of dynamic helpers can be accomplished before tuning helper size although the sample of problems here is too small to determine if this conclusion is robust.

As before, average best solution by generation was plotted for problems and cases. Figure 21 shows the average best solution found by generation for the la06 and swv07 problems. These two problems were chosen for their contrast in helper size effectiveness. With 75 operations, the la06 problem is relatively small compared to the 300 operations in swv07. In the la06 problem the five and $\lceil n/2 \rceil$ JPH cases appear to gain an early lead and then maintain most of that lead throughout the remainder of the run. Nearly the opposite is true for the swv07 problem where the early lead gained in the five and 10 helper cases is quickly lost as these helper methods slow down their rate

of improvement between generations 50 and 200. The two dynamic helper with one JPH in swv07 starts slower but later makes improvements at a better rate than the other methods. This momentum results in the final 'win' for the two dynamic helper with one JPH in that case. The mitigation of epistasis later in the run may be the reason for the lack of slowdown.

Table 12 – Average best found solution as percentage from the best known solution for various JPH

Problem	Best Known Solution	1 Dyn Hlp 1 JPH	2 Dyn Hlp 1 JPH	1 Dyn Hlp 2 JPH	1 Dyn Hlp 5 JPH	1 Dyn Hlp 10 JPH	1 Dyn Hlp $\lceil n/2 \rceil$ JPH
la01 (10x5)	4832	2.297	**2.190**	2.431	2.992*	-	2.992*
la02 (10x5)	**4459**	3.111*	3.083*	3.177*	**2.447**	-	**2.447**
la06 (15x5)	8694	4.941*	6.251*	4.503*	4.137	-	**3.828**
la07 (15x5)	**8174**	5.425*	7.221*	5.245*	**4.451**	-	5.091*
la11 (20x5)	**14756**	5.537*	8.120*	5.484*	5.072	**5.027**	**5.027**
la12 (20x5)	**12403**	6.252*	9.216*	6.034*	5.528*	**4.893**	**4.893**
la16 (10x10)	7393	4.230	**3.795**	3.993	4.449*	-	4.449*
la17 (10x10)	6555	3.555*	3.375*	3.696*	**2.617**	-	**2.617**
la21 (15x10)	**12942**	4.489	4.776	**4.320**	4.503	-	4.537
la22 (15x10)	12106	4.581	4.526	5.182*	4.876*	-	**4.337**
la26 (20x10)	20234	**5.290**	6.581*	5.480	5.583	5.924*	5.924*
la27 (20x10)	**20764**	**5.321**	7.061*	5.428	5.529	5.768*	5.768*
la31 (30x10)	39007	**5.321**	6.869*	5.815*	6.259*	6.449*	6.123*
la32 (30x10)	**42189**	4.863*	6.459*	4.728*	**4.304**	4.529	5.026*
la36 (15x15)	17073	5.052*	**4.309**	5.348*	4.864*	-	4.943*
la37 (15x15)	17886	5.114*	**4.725**	5.337*	6.034*	-	5.321*
ft10 (10x10)	7501	6.822*	**5.838**	6.456*	7.267*	-	7.267*
ft20 (20x5)	14279	7.616	9.689*	**7.320**	7.796	8.633*	8.633*
swv01 (20x10)	20688	9.768*	**8.764**	10.301*	12.124*	13.259*	13.259*
swv02 (20x10)	21682	8.334	**8.058**	9.956*	10.841*	10.867*	10.867*
swv06 (20x15)	28863	7.022*	**5.960**	7.669*	10.381*	9.782*	9.782*
swv07 (20x15)	27385	7.937*	**6.623**	8.403*	10.770*	10.834*	10.834*
swv11 (50x10)	108842	8.912*	**8.032**	9.991*	13.077*	14.125*	13.068*
swv12 (50x10)	109128	**8.737**	9.036	10.912*	13.417*	14.470*	13.750*
Average	-	5.855	6.273	6.134	6.638	-	6.699
Average (10)	-	6.993	7.728	7.502	8.514	8.812	8.689

Flowtimes in bold indicate a new best flowtime was found in the analysis. Prior best flowtimes prior to this analysis were reported in Table 10 and (Lochtefeld and Ciarallo, 2010). Percentages in bold indicate the lowest percentage for all cases shown. Percentages in italics indicate the lowest percentage for the one dynamic helper cases. '*' values indicate statistically higher percentage as compared to the lowest percentage for all cases. Percentages in the '1 Dyn Hlp $\lceil n/2 \rceil$ JPH' that are grayed are repeated values from a previous column where $\lceil n/2 \rceil$ was equal to 5 or 10. Since not all cases were run with the 10 JPH setting, the 'Average 10' row reports the averages of only the problems that ran the 10 JPH setting. Statistical significance was tested at the $\alpha = 0.05$ level using a two tailed student's t-test assuming unequal variance.

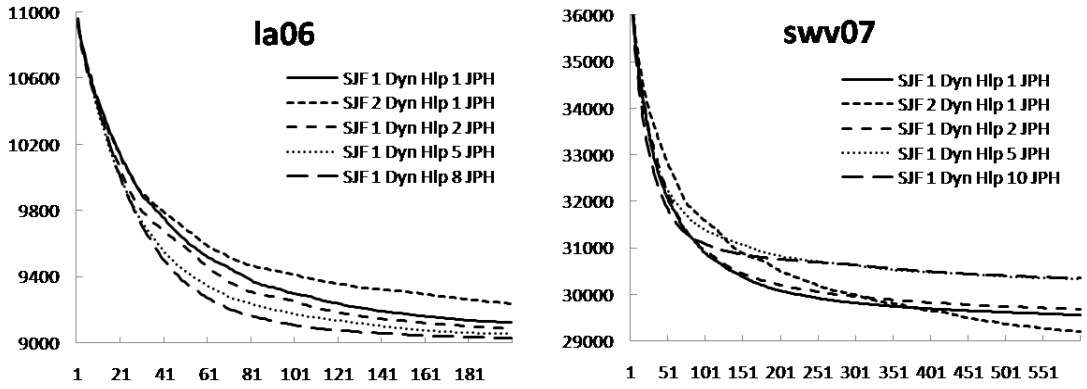


Figure 21 – Problems la06 and swv07 Average Best Solution by Generation for Various Helper Sizes

Perhaps the properties of fast solution improvement early can be combined with the late improvements observed in the two dynamic helper methods. It may be useful to change the number of dynamic helpers during a run so that a larger number of dynamic helpers are used later in the run. Smaller helpers could help establish good progress early in a run with larger helpers helping to break stronger epistasis later in a run. A time phased strategy for dynamic helpers may produce better performance than the fixed number of dynamic helpers studied here. It may be possible to break the remaining epistasis late in a run by increasing the number of helpers when the search begins to stall. Based on the supposition that epistasis is dynamic, it is probably more useful to use additional helpers later in the search when the search becomes more biased as a result of epistasis and random error (drift).

4.2.4 Helper effectiveness and non-dominated front size

The average size of the non-dominated front was examined here to explore the relationships between the front size, number of dynamic helpers, and helper effectiveness. Additionally, this analysis explored impact of helper sequence on front size. Average front size by generation was plotted for all of the problems and cases. Figure 22 shows average front size by generation for cases with one JPH for problems 1a01 and la06. Some differences between the SJF method and the random helper selection were evident. When compared to the random helper selection method

the SJF helper selection method generates larger or smaller front sizes depending on the time period observed. For example in Figure 22 for the la06 problem a marked increase in front size can be seen for the fourth and sixth helper periods for the two dynamic helper SJF case. These variations show that for certain helpers there are a larger number of solutions that have competitive trade-offs between the helper's fitness and the main objective. This richer trade space makes front size increase for the fourth and sixth helpers. In the random helper selection method, variations were averaged out across many replications since helpers could appear anywhere, but the SJF helper selection method highlights the differences in jobs since helper selection sequence is mostly deterministic.

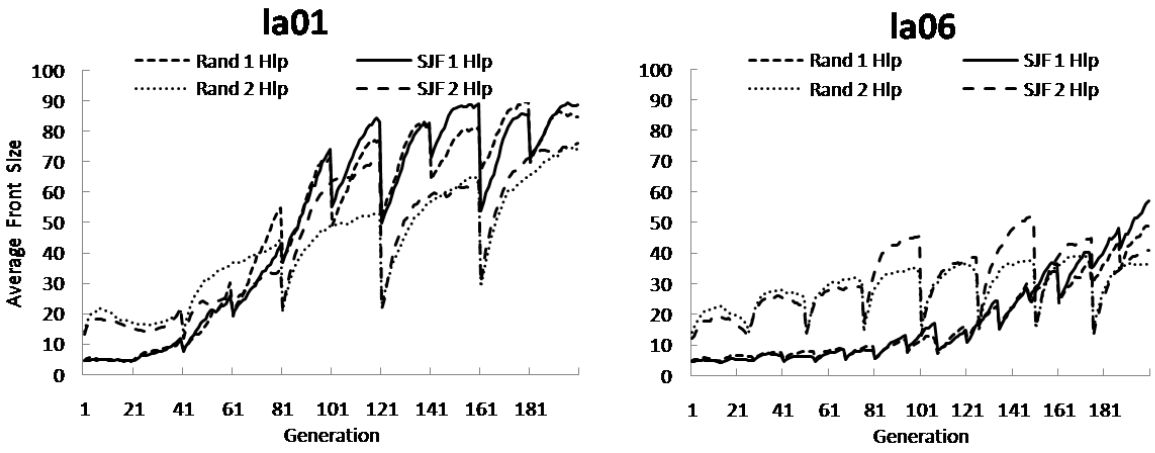


Figure 22 – Average front size by generation for problems la01 and la06

Jensen observed that in cases where the two dynamic helper outperformed the one dynamic helper, the two-helper front size was smaller than the one-helper front size early in the run. From this observation he concluded that "this strongly indicates that keeping the size of the first non-dominated front small is important in tuning this kind of algorithm" (Jensen, 2004). The overall trend of the front sizes with the same number of dynamic helpers appears to be similar for all problems. When front size climbed quickly for the SJF method, a similar effect for the random helper selection method was observed. In previous work, two dynamic-helper cases only

performed better than one dynamic helper cases when their front size became larger than the one dynamic helper cases within the first one third of all generations. However, observations here are not consistent with that finding for eight of the thirteen cases where the SJF two dynamic helper performed on average better than the SJF one dynamic helper. The attainable front size is likely more determined by the problem details, helper definition, and helper size rather than by front size reduction operators. The problem likely determines how many robust solutions are easy to find for a given helper definition. Robustness is determined by the number of helper swaps that would result in that solution's survival. Solutions that are robust survive throughout much of the run and generally keep front size large. If many robust solutions can be found early, front size rises quickly. This brings to question whether tuning the algorithm based on front size is a reasonable conclusion since the data here appears inconsistent with Jensen's original finding.

One might think that the two helper cases should have a larger Pareto front after a helper swap since the front is three dimensional rather than two dimensional and the added dimension would cause more solutions to be incomparable. However, this view does not take into account how the results of previous helpers affect the front size of the latest helpers. While the two-helper algorithms are searching for solutions on the Pareto front, they can focus on the two current helpers at the expense of the main objective. This rewards solutions with good low values in at least one of the two jobs associated with the current helpers but, these rewards are often at the expense of the main objective. The currently known Pareto optimal solutions incur a price – other jobs in the solutions are allowed to have poor values since the main objective is not critical to a larger portion of the Pareto optimal trade space. With more than one dimension focused away from the main objective, the two-helper cases build solutions that are poor in job values other than those associated with the current helpers. After a helper swap, these poor solutions may be easily dominated by one or more other solutions. That domination causes the marked decrease of front size for the two-helper cases. In one-helper cases, less focus is taken away from the main

objective. To survive through the current helper period solutions must be competitive in at least one of two dimensions rather than being competitive in one of three dimensions This more limited focus causes increased competition for main objective fitness in one helper cases. After one-helper swaps, fewer solutions are dominated because the solutions had more selection pressure with respect to the main objective. Higher main-objective selection pressure rewards overall better values throughout more jobs and with less jobs that have poor fitness it is less likely for many solutions to be easily dominated after a one-helper swap.

Jensen suggested there was strong evidence that keeping front size low was important for tuning the algorithm. Quantitative evidence between the correlation of average front size and problem effectiveness may help to determine if such tuning is beneficial. To test this hypothesis, for each problem and setting combination, data was collected for the average front size across all generations for each replication and the final best fitness for each replication. These two data values were tested for correlation to determine if a relationship exists between average front size and performance. Table 13 shows the correlation coefficients for the problem-case combinations. There are no strong correlations in the table as the lowest correlation is -0.494 and the highest is 0.375. From most of the average correlations, a slightly negative value appears to exist indicating a possible link. As front size gets smaller the solution improves or vice-versa. There appears to be a difference between the one helper cases and the two helper cases. The average correlation for the two dynamic helper cases is almost neutral, and certainly weaker than one helper correlations.

It is believed that the somewhat negative correlation between average front size and performance is a property of this type of algorithm and that little evidence exists in the 24 problems studied that indicates that front size should be tuned directly. NSGA-II determines front size based upon Pareto dominance. Since the total flowtime is one of the objectives in this case, finding a new best solution for the run always adds a single point to the Pareto front. However, this point may also dominate zero to many other points on the front. A large jump in solution

fitness would frequently result in elimination of potentially many more points on the front. Several or more of these points may be in the population. As a result, larger jumps in fitness likely produce lower average front size. For the two helper cases, more incomparable solutions exist because of the additional objective. When a large improvement in fitness is found for the best known solution in a two dynamic helper method, it will likely eliminate fewer solutions on the current Pareto front because fewer of these solutions were on the front due to their overall fitness value. Two helper methods on average have lower correlation because "shifting focus away from the main objective" creates more opportunities for solutions that are poor in the main objective to survive.

Low front size having low but perceptible correlation with effectiveness is a part of the nature of the algorithm since early appearance of good solutions in a run leads to lower average front size. Low front size can be thought of as a byproduct of finding better solutions rather than as a cause of why the algorithm finds good solutions. Jensen used operators such as niche-enforcement and in-breeding control, on the JSSP with helpers. Without question, these operators can be useful to the search as his results show. However, his intuition for why the operators were helpful, through reduction of front size, may be a questionable conclusion. No evidence here indicates low front size gives better results. Instead of viewing front size as 'tunable parameter', it is likely better to interpret the weak correlation between non-dominated front size and effectiveness for what it is - a simple product of this type of algorithm.

Table 13 – Correlation between average front size and quality of best found solution

Problem	Random 1 Dyn Hlp 1 JPH	SJF 1 Dyn Hlp 1 JPH	Random 2 Dyn Hlp 1 JPH	SJF 2 Dyn Hlp 1 JPH	SJF 1 Dyn Hlp 2 JPH	SJF 1 Dyn Hlp 5 JPH	SJF 1 Dyn Hlp 10 JPH	SJF 1 Dyn Hlp $\lceil n/2 \rceil$ JPH	All 1-5 JPH Average
la01 (10x5)	-0.224	-0.344	-0.351	-0.259	**-0.494**	***-0.063***	-	-0.063	-0.289
la02 (10x5)	-0.195	-0.266	-0.261	***0.074***	**-0.309**	-0.166	-	-0.166	-0.187
la06 (15x5)	-0.181	-0.064	0.004	***0.337***	-0.171	**-0.198**	-	-0.164	-0.045
la07 (15x5)	**-0.373**	-0.015	-0.002	***0.112***	-0.103	-0.099	-	-0.100	-0.080
la11 (20x5)	-0.123	-0.125	0.062	0.001	***0.026***	**-0.134**	-0.044	-0.044	-0.049
la12 (20x5)	-0.155	**-0.177**	0.053	***0.141***	0.037	0.121	-0.059	-0.059	0.004
la16 (10x10)	-0.304	-0.344	**-0.363**	-0.205	-0.280	***-0.186***	-	**-0.186**	-0.280
la17 (10x10)	***-0.315***	-0.424	-0.319	-0.326	-0.319	**-0.495**	-	**-0.495**	-0.366
la21 (15x10)	-0.257	-0.254	-0.051	***0.101***	-0.234	**-0.386**	-	-0.165	-0.180
la22 (15x10)	-0.248	**-0.299**	-0.310	***0.002***	-0.366	-0.155	-	-0.278	-0.229
la26 (20x10)	-0.247	-0.284	0.184	***0.236***	**-0.291**	-0.222	-0.079	-0.079	-0.104
la27 (20x10)	-0.302	**-0.317**	***0.159***	0.104	-0.146	-0.292	-0.074	-0.074	-0.132
la31 (30x10)	-0.140	-0.048	0.057	***0.314***	**-0.318**	-0.186	-0.143	-0.061	-0.054
la32 (30x10)	-0.136	-0.106	**0.257**	***0.375***	-0.169	-0.161	-0.238	-0.246	0.010
la36 (15x15)	-0.260	**-0.314**	-0.274	-0.233	-0.290	***-0.204***	-	-0.209	-0.263
la37 (15x15)	-0.303	**-0.308**	-0.167	***-0.141***	-0.169	-0.207	-	-0.261	-0.216
ft10 (10x10)	-0.222	-0.341	**-0.336**	-0.332	***-0.057***	-0.245	-	-0.245	-0.256
ft20 (20x5)	-0.191	-0.140	***-0.073***	0.181	-0.129	-0.190	**-0.243**	**-0.243**	-0.091
swv01 (20x10)	-0.174	**-0.284**	***-0.072***	-0.077	-0.272	-0.164	-0.168	-0.168	-0.174
swv02 (20x10)	-0.198	-0.244	-0.119	***-0.061***	**-0.286**	-0.152	-0.283	-0.283	-0.177
swv06 (20x15)	-0.203	-0.154	**-0.236**	***-0.064***	-0.068	-0.115	-0.140	-0.140	-0.140
swv07 (20x15)	-0.258	**-0.332**	-0.097	***-0.007***	-0.212	-0.193	-0.065	-0.065	-0.183
swv11 (50x10)	-0.071	0.012	***0.058***	-0.043	**-0.228**	-0.173	-0.190	-0.143	-0.074
swv12 (50x10)	**-0.242**	-0.152	-0.078	-0.101	-0.086	-0.225	-0.080	***-0.006***	-0.147
Average	-0.222	-0.222	-0.095	0.005	-0.206	-0.187	-	-0.164	-
Average (10)	-0.188	-0.181	0.012	0.077	-0.165	-0.160	-0.139	-	-

Correlations in bold indicate the lowest correlation for a given problem for all cases. Correlations in bold and italics indicate the highest correlation for a given problem for all cases. Grayed correlations in the 'SJF 1 Dyn Hlp $\lceil n/2 \rceil$ JPH' are repeated values from a previous column where $\lceil n/2 \rceil$ was equal to 5 or 10. The 'All 1-5 JPH Average' column reports the average of the first six columns of data since all problems were run for those cases. Since not all cases were run with the 10 helper setting, the 'Average 10' row reports the average correlation of only the problems that were run for the 10 JPH setting

4.3 Chapter Conclusion

Establishing that helper size matters and the possible relationship between helper effectiveness and time increases overall understanding of multi-objectivization using MOEAs. This paper supports the theory that epistasis is likely best viewed as a dynamic phenomena – one that changes with the population members and problem constraints. Some evidence indicates that

the early breaking of epistasis is better than attempting to break it later in the run. The evidence from these experiments supports that helper methods may break epistasis and help identify good building blocks in solutions. Helpers have been shown to provide breakthroughs in finding better solutions by making it easier to find a larger number of solution improvements after a helper change. These breakthroughs are likely caused by a combination of epistasis breakage and solution differentiation.

Problem specific knowledge has been shown to help govern helper sequence to achieve better search results. SJF helper selection may be able to identify and break epistasis by biasing the search to productive areas early. The SJF helper selection method dominates random helper selection for the 24 different JSSP problems studied here. In addition to helper sequence, helper size is shown to be important. Unlike prior logical arguments, some evidence here indicates that maximal problem division is not the best approach. Especially for large problems, two large helpers yield significantly worse results than smaller divisions.

Further research in this area may explore the extent and ways in which helper methods can combat other potentially negative forces in the optimization. Confirmation of helper method's ability to break epistasis may be explored in a theoretical problem where problem features such as epistasis can be explicitly setup and studied. This study would help to understand what causes multi-objectivization methods to perform better than single objective methods on certain problems. Further research can be applied to determining the appropriate sizes and sequencing of helpers so that specific strategies can be developed for efficiently solving different types of problems. For example, a generic greedy helper selection method like the SJF method here can be tested against other problem types such as the TSP to establish if helper sequence is sensitive in these other problems. Lastly, a study of the phasing of the number of dynamic helpers could attempt to determine if using more helpers later in the search is indeed beneficial as suspected.

Chapter 5 Multi-objectivization with the Tunable Objectives Problem (TOP)

Prior works on multi-objectivization hypothesized that the breakage of epistasis and the subsequent overcoming of local optima may be a significant factor leading to the success of multi-objectivization in Genetic Algorithms (GAs) (Knowles et al., 2001, Jensen, 2004, Lochtefeld and Ciarallo, 2010). There has been, however, little direct evidence to show that this hypothesis is true. Furthermore, one multi-objectivization method, helper-objectives, has been shown to distinguish between seemingly similar solutions to create more frequent improvements in fitness (Lochtefeld and Ciarallo, 2011b). A shortcoming of research in multi-objectivization to date is that there are few general theories proposed for determining when multi-objectivization should be used *ab initio* - from the beginning. The study of general problem structures may assist in understanding when multi-objectivization approaches are useful so that their utility can be determined *a priori*.

Helper-objectives and more recent methods such as Multi-Objectivization via Segmentation (MOS) have been shown to produce good results when solving single-objective problems using multi-objectivization via decomposition (Jensen, 2004, Jahne et al., 2009, Lochtefeld and Ciarallo, 2010). Helper-objective methods use a Multi-Objective Evolutionary Algorithm (MOEA) that simultaneously optimizes the main objective and one or more decomposed objectives derived from the main objective function. There may be many possible decomposed objectives so helper methods often use only a few of these objectives simultaneously. The simultaneously used helper-objectives are called dynamic helper-objectives. A helper-objective algorithm that uses fewer than the maximal number of helper-objectives at a given time is called

an *x* dynamic helper-objective algorithm where *x* is the number of simultaneously used helper-objectives.

Study in multi-objectivization to date has been focused on practical rather than abstract problems. However, an abstract problem can bridge the gap between a highly complex problem where little evidence points to exactly how an algorithm works and a simple problem where no insight can be gleaned about a sophisticated algorithm. Furthermore, an abstract problem may be used to identify general problem features that increase or decrease the effectiveness of multi-objectivization. Identification or creation and the study of an abstract problem can show the utility of multi-objectivization and can help identify the value behind multi-objectivization.

The background section reviews several abstract problems described to date and identifies some important requirements for a new problem to study multi-objectivization. The following section proposes the Tunable Objectives Problem (TOP) model. Then several experiments examine TOP's basic features. Additional experiments are performed on the TOP model to determine the effectiveness of multi-objectivization relative to problem features. Algorithms are studied for their ability to overcome sequential and parallel optima in the face of conflicting objectives. Conclusions are drawn from these experiments that identify several different features that, when present, make multi-objectivization effective. Underpinning one of these features is the concept of signal-to-noise ratio (SNR). In a general sense SNR is the ratio of the desired or interesting features of a process versus the undesired or uninteresting features of the process. Finally, we provide a summary of this chapter in the final section.

5.1 Background

The background section here is divided into two major sections. The first section discusses multi-objectivization research and the second section discusses abstract problems with bit string based search spaces.

5.1.1 Multi-objectivization

The term *multi-objectivization* refers to the process of reformulating a single-objective problem into a multi-objective problem and solving it with a multi-objective method in order to provide a solution to the original single-objective problem (Knowles et al., 2001). Multi-Objective Evolutionary Algorithms (MOEAs) are most commonly used to perform the role of solving the newly reformulated problem. Multi-objectivization methods can be arrived at by adding new objectives, or by decomposing the original objective into multiple objectives (Handl et al., 2008a). Decomposition of the original objective is natural for many problem types where a sum-of-parts fitness function can be easily split into different objectives. In a sum-of-parts multi-objectivization via decomposition, the main objective has been shown to have at least as many local optima as the sum of the number of all local optima present in the multi-objective space created by the decomposed objectives (Handl et al., 2008a). Multi-objectivization via decomposition has demonstrated better performance than single-objective methods for a variety of Large combinatorial optimization problems including the Traveling Salesman Problem (TSP) (Jensen, 2004, Lochtefeld and Ciarallo, 2010) and the Job Shop Scheduling Problem (JSSP) (Jensen, 2004, Knowles and Corne, 1999, Jensen, 2003a, Jahne et al., 2009). Furthermore, for less complex problems such as the Single Source Shortest Path (SSSP) (Scharnow et al., 2004) problem and the minimum spanning tree problem with certain properties (Neumann and Wegener, 2007), multi-objectivization has been proven to on-average outperform single-objective methods.

One version of multi-objectivization uses the concept of *helper-objectives* to assist in the search process. Helper-objectives are new objectives that are added to the original problem and used simultaneously with the original objective to solve the single-objective optimization (Jensen, 2004). In prior research on helper-objectives, the helper-objectives were generally decomposed parts of the main objective (Jensen, 2004, Jahne et al., 2009, Lochtefeld and Ciarallo, 2010). For

instance, in the TSP, a potential helper-objective could be the minimization of cost of travel between cities A and B. Since the full problem in the TSP is to minimize the cost of traversal from cities A to B and back from B to A, the helper objective of minimizing the cost of traversal from A to B is always a component of the larger cost function (Knowles et al., 2001).

The number of potential helper-objectives could be large as these problems have numerous possible divisions. Since using many divisions is detrimental to search, it is desirable to have fewer helper-objectives in-use at any one time during the execution of the MOEA (Jensen, 2004). The reduced set of helper-objectives in-use are called *dynamic helper-objectives*. An algorithm that uses *x* dynamic helper-objectives simultaneously is called an *x*-dynamic helper-objective algorithm. Jensen (2004) shows that for the JSSP and TSP, one or two dynamic helper-objectives are best for improving search quality. For the remainder of this chapter, helper-objectives are referred to as *helpers*.

To better understand multi-objectivization research and how it contributes to the optimization process, it may be best to understand the basic concepts that make a problem hard for a GA. Negative problem features such as deception (Goldberg, 1989a,Goldberg, 1989b), epistasis (Davidor, 1991), and neutrality (Barnett, 1998) may explain these GA difficulties. Epistasis is the most general of these three problem properties. Deception is a special case of epistasis (Beasley et al., 1993) and most levels of neutrality are special cases of epistasis. Epistasis is the non-linear interaction between components of a solution as reflected in the solution fitness. Components of a solution that contribute to epistasis can be the original variables (genes) (Beasley et al., 1993) or components of the encoded variables' representations (bits) (Naudts et al., 1997; Naudts and Verschoren, 1999). From a Design of Experiments (DOE) perspective, epistasis can be viewed as the higher-order or non-main effects in a model predicting fitness values based on interactions of the problem variables (Reeves and Wright, 1995).

5.1.2 Basic related work

Abstract problems based on general problem features have been used in the past to provide a general analysis platform for different algorithms. Two major types of abstract problems explored by GA researchers have been Royal Road problems and NK landscapes. A third abstract problem with elements of both the Royal Road and NK landscapes has been recently introduced. Often NP hard problems can be too complex to provide conclusive evidence about how an algorithm performs and thus these problems provide limited insight about how a method can be extended into other problem domains. It is useful to study abstract problems when little is known about a particular algorithm or process as these problems can be general enough to capture elements of many problems, simple enough to provide strong evidence of algorithmic behaviors, and yet complex enough to capture a (limited) number of the nuisances inherent in real problems.

The Royal Road problem, introduced in Mitchell et al. (1992), is a problem with a sectioned genotype where each section of bits contributes to the fitness if all bits in that section contain values of 1. An additional amount of fitness is added to the overall fitness when certain neighboring sections also contain all 1s. Epistasis clearly exists in the Royal Road problem because changing a single bit can have either no impact or considerable impact to fitness depending on the values of other bits. Neutrality also exists since the fitness does not change even if several bits change value, unless all bits in a given section have a "1" value. The basic Royal Road problem does not contain deception. Naudts et al. (2000) generalized the Royal Road problem in order to study the effects of epistasis on varying sizes of the problem. Like NK landscapes, Royal Road problems are symmetric.

Kauffman's NK problem landscapes introduced in Kauffman and Weinberger (1989) contain controlled levels of epistasis introduced through ruggedness. Ruggedness is the degree of peaks and valleys contained in a fitness landscape. By varying the parameter K, the ruggedness of the landscape can be dialed up or down – changing the number and height of peaks. The number of

peaks also varies with the number of bits in the solution N. NK landscapes have been extensively studied and have been extended several times. Fontana et al.(1993) and Weinberger (1990) are two example studies using NK landscapes. Two extensions of NK landscapes are NKp and NKq landscapes. NKp landscapes add the concept of neutrality to the NK problem structure by mutating random bits to correspond to locally neutral values (Barnett, 1998). By mapping real fitness values to an integer fitness function, NKq landscapes (Newman and Engelhardt, 1998) add neutrality to NK landscapes but do so in a considerably different manner than NKp landscapes (Geard et al., 2002). The NK model has been criticized for uniformity in epistasis by gene when in fact real problems rarely have such uniformity (De Jong et al., 1997).

5.2 The Weise model

An abstract problem introduced in Weise et al. (2008) combines elements of NK landscapes and Royal Road landscapes and adds the ability to model stochastic noise as well as multiple objectives. This abstract problem is further referred to here as the Weise model. The Weise model uses a series of transformations in order to map a given set of bits from the genotype space to a given set of variables in the phenotype space. Additional transformations are performed to map variables in the phenotype space to one (or more) objective values in the solution space. Figure 23 shows the steps involved in and parameters used to map a given genotype to one or more objective values. When associated parameter(s) are set to default value(s) each transformation in the Weise model may be disabled so that only the transformations of interest are modeled in the problem. The goal of the Weise model problem is to minimize the value of one or more objectives. The optimal solution for the overall string is always the decoded binary bit string 010101... Solutions are assigned an objective value derived from their Hamming distance (Hamming, 1950) from the optimal solution. The following sub-sections describe each of these transformations in the order in which they are performed.

5.2.1 Genotype creation

The first process in the Weise model is to create a genotype of the appropriate length. The genotype is defined by a string of 0-1 bits of length g where $g = nm\mu$. n is the number of bits per objective, m is the number of objectives, and μ is the redundancy parameter. Since no solutions are eliminated due to invalidity, the size of the genotype search space is 2^g.

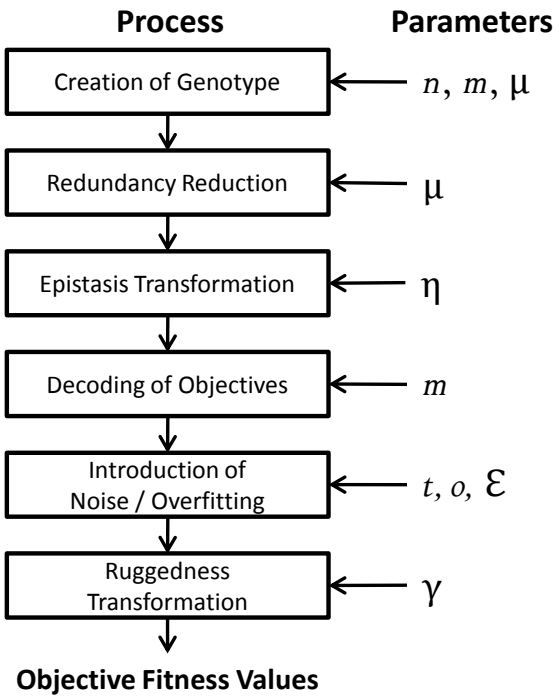


Figure 23 – The Weise model landscape transformation process

5.2.2 Redundancy reduction

After the genotype is created, a redundancy reduction is applied to the genotype. The redundancy reduction process uses the parameter μ to reduce μ bits of the genotype into a single bit. This transformation creates plateaus in the landscape because bit changes do not always result in a change in their mapped values. To reduce a block of μ bits, the algorithm counts the number of 1s and the number of 0s in the block and assigns the block's value to be the greater of the two values. For instance, if μ = 4, a block of 4 bits with values {0,0,1,0} would reduce to the single

bit value of 0 because there were more 0s in the block than 1s. When the count of 0s and 1s in a block is equal, a block is assigned the value of 1. Since the optimal solution is a set of alternating 0s and 1s, assigning a value of 1 to the tie-breaker does not bias the redundancy reduction process. When μ has the value of 1, no additional neutrality is added to the model. The redundancy reduction is the most direct way in which neutrality is introduced in the Weise model.

5.2.3 Epistasis transformation

After the redundancy reduction is applied, a transformation is accomplished to partially model objective-to-objective interactions (conflicts and synergies). This transformation is referred to as the epistasis transform in the Weise model. 'Epistasis' in the transformation name may be misleading to the casual reader. The Weise model epistasis transformation is only one component part of modeling the total epistasis concept since portions of the other transformations also contribute to epistasis. The parameter η ranges from 2 to nm and governs the transformation. A gene is defined by η consecutive bits. As η increases, more bits conflict with each other. Epistasis transformation is applied to a gene as described in the pseudo-code in Figure 24. Objectives are interleaved in the genotype so that the epistasis transform applies across as many objectives as possible. Objectives are then later decoded (discussed further below). The epistasis transformation maps a set of bits to a different set of bits of equal size such that every mapping is unique and the transformation is bijective. The epistasis transformation on a gene of size η is applied to η consecutive bits until all bits in the genotype have been transformed. An η value of 2 causes no change in the genotype and implies that the objectives are not conflicting. For nm values that do not divide evenly by η, a smaller transformation is performed on the final remaining $nm\%\eta$ bits. The epistasis transformation is a special case of NK landscapes where $N = n$ and $K \approx \eta - 2$ (Weise et al., 2008). Weise et al.(2008) offers additional details regarding the description of the transformation.

5.2.1 Decoding of objectives

After the epistasis transformation, the objectives are decoded from the newly transformed genotype into an objective value. Recall that for a given solution there are m objectives with n non-redundant bits each. Since objectives were interleaved in the original genotype, the z^{th} objective contains n bits and consists of the bits at positions (z, $z+m$, $z+2m$, ... , $z+nm$) in the transformed genotype. The Hamming distance from the optimal solution for each objective gives an objective value for that objective. Exactly $n+1$ objective values exist for an n bit objective when the overfitting and oversimplification processes (discussed in the next paragraph) are not present. Since $n+1$ is much smaller than the search space of 2^n, we know there is an implicit amount of neutrality in the Weise model. However, since the Hamming distance from the optimal solution is used to generate objective values, the implicit neutrality only exists when changing an even number of bits in an objective. Due to the nature of the Hamming distance from an arbitrary solution in the search space, as the number n of bits per objective gets larger, more solutions exist that are far from the optimal solution. This is a good feature of an abstract problem since, like most NP hard problems, random solutions have a very low probability of being optimal (or near optimal).

```
Integers: η, i
Boolean: b
Array of 0-1 Integers: originalGene, newGene

r ← EpistasisTransformOnGene(η, originalGene)
begin procedure
    b ← false
    if originalGene[0] equals 1
    begin if
        originalGene[0] ← 0
        b ← true
    end if

    for i = 0 to η − 1
    begin for
        newGene [i] ← XOR(all bits in originalGene
        except bit at (i − 1)% η)
    end for

    if b equals true
    begin if
        for i = 0 to η − 1
        begin for
        newGene [i] ← NOT(newGene [i])
        end for
    end if

return newGene
end procedure
```

Figure 24 – Epistasis transformation pseudo-code for a gene

5.2.2 Overfitting and oversimplification

An overfitting and oversimplification process allows the Weise model to sample multiple objective values close to the optimal by using a modified Hamming distance function. The process can simulate stochastic variation in data so that the model can be used to study pattern recognition or stochastic optimization. The overfitting and oversimplification process is governed by the parameters *t*, Ɛ, and *o*. Overfitting causes the model to only use a subset of bits to handle objective evaluation. Ɛ governs the number of bits reduced by overfitting. Oversimplification is governed by the parameter *o*. Oversimplification introduces *o* wildcard bits that can cause a solution to have an objective value of a neighboring solution in the landscape. Both overfitting and oversimplification use a random process to introduce and reduce bits and so *t* samples near the actual solution are generated by the process and used to determine the objective value. If default values are set for the overfitting and oversimplification process, the process always samples the Hamming distance from the optimal solution. The default values of *t*, Ɛ, and *o* are 1,

0 and 0 respectively. When these parameters have their default values, the model always assigns the same objective value to a given objective.

5.2.3 Ruggedness transformation

The last transformation applied in the Weise model is the ruggedness transformation. This transformation attempts to model "the nature of the problem" with respect to the objective value landscape by varying the roughness of the objective value landscape directly (Weise et al., 2008). Adding ruggedness to an objective value landscape increases the number and magnitude of local optima. A highly rugged landscape contains many high peaks and valleys while a landscape with low ruggedness contains smaller changes in objective values and fewer local optima. According to (Weise et al., 2008), a good ruggedness transformation should: maintain a bijective 1:1 mapping of old objective values to new values, preserve (not transform) the optimal value, and increase ruggedness with increasing value of the γ parameter. To measure overall ruggedness the absolute difference between the heights of objective values for neighboring solutions is summed across all possible objective values. Each objective value between 0 and q appears exactly once in the ruggedness map since the mapping of objective values is bijective. The measure of ruggedness uses the following equation: $\Delta(r) = \sum_{i=0}^{q-1} |r_i - r_{i+1}|$ where q is the maximum possible objective value. With default values in the overfitting and oversimplification process, $q = n$. The ruggedness parameter γ can range from the value of 0 to $q(q+1)/2 - q$.

No research to date has been performed on a generic problem to better understand multi-objectivization as it applies to general problem features. A series of experiments on an abstract problem could provide researchers a better understanding of the fundamentals behind multi-objectivization.

5.3 The Tunable Objectives Problem (TOP)

In order to study more details of multi-objectivization, a derivative of the Weise model called the Tunable Objectives Problem (TOP) was developed. TOP is a good candidate for the

study of multi-objectivization via decomposition in genetic algorithms for a number of reasons. The model has a single objective with distinguishable Subordinate-Objectives (SO) that can be grouped together. These SOs and their groups create natural helpers for multi-objectivization. Like the Weise model, TOP allows us to isolate effects from different transformations by selecting default values for other transformations. Thus studies of TOP can be accomplished while varying relatively few parameters. TOP contains many aspects of epistasis including ruggedness, neutrality, and objective interaction, and has the ability to model infeasible solutions through poor objective values. With use of different ruggedness transformations and objective layers, TOP enables us to study the ability of multi-objectivization to optimize past local optima in both serial optima that must be overcome sequentially and parallel optima that can be overcome simultaneously.

To accomplish the experimental objectives, several modifications from the Weise model were required. These modifications include a method to model a single-objective problem with multiple summed-up objectives, a modified way to accomplish ruggedness transformations, ruggedness transformations at more than one layer of the problem, and a modification of the epistasis transformation for a gradual increase in problem hardness. An overview of the TOP model is shown in Figure 25. Processes in Figure 25 that are in grey are identical to those in the Weise model while the processes in black are those that have received modification and are described in further detail below. Unless otherwise noted here, sections of the TOP model that correspond to sections of the Weise model use the transformations described in Weise et al. (2008).

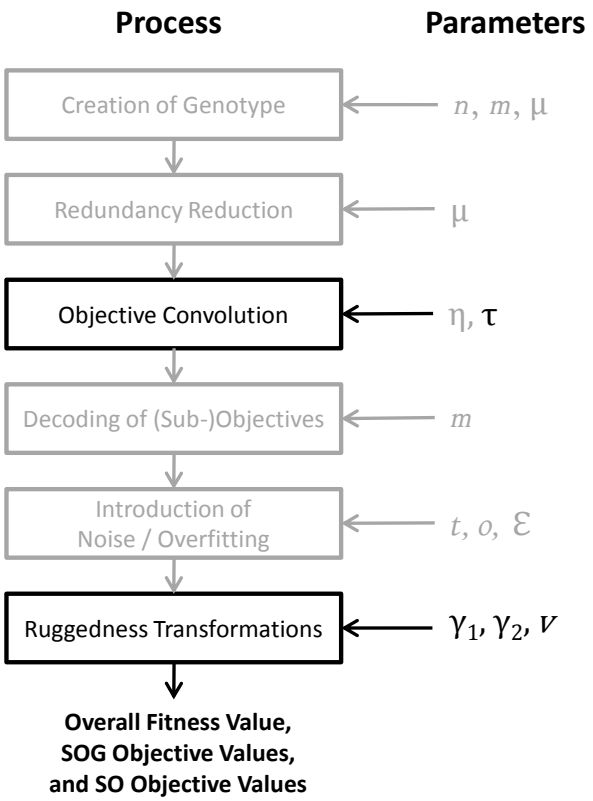


Figure 25 – The TOP landscape transformation process

In constrained problems, some or many solutions are infeasible. For example in the JSSP many potential schedules are not valid due to order and capacity constraints for machines and jobs. In TOP, instead of eliminating potential solutions in the search space, infeasibility can be modeled by giving certain solutions worse than average objective values. Conceptually, considering some or all above-average objective values as infeasible directly parallels the common practice of applying a penalty function to infeasible solutions. Thus, there are two ways to interpret above average objective values. These poor objective values could stand for infeasible regions the algorithm must avoid. Alternatively, these objectives can be viewed as simply poor areas in the search landscape which the algorithm must overcome or avoid because of their bad objective values. To handle multi-objectivization elegantly, TOP has implemented several layers of objectives which are discussed next.

5.3.1 Modeling problems with layered and summed objectives

To create a single-objective problem that has multiple sections, the TOP model aggregates separate objective values into a single fitness value. Similar to prior multi-objectivization research that has focused on sum-of-part type fitness functions such as in the TSP and JSSP, the TOP model sums objective component values together into a single overall fitness value. Much like the TSP and JSSP, TOP has the capability to model objectives that are created by combining component objectives. Having only a single layer of objectives is too restrictive to model interactions between objectives that can occur at multiple levels. While the TSP and JSSP can be thought of as having many objective layers, we explore the effect of layered interactions in multi-objectivization through the use of a second objective layer in TOP. The fitness of an individual solution in TOP is the sum of the objective values associated with the top-most (most aggregate) layer of objectives.

The TOP with two hierarchical layers of objective values enables the modeling of interactions between objective values and/or the modeling of hard-to-generate solutions caused by the regions of solution infeasibility. The most primitive objective layer in TOP is the SO layer. The SO layer models the smallest arbitrary objective slice of the problem. The number of SOs is determined by the parameter *m*. After redundancy reduction, the number of bits in a SO is *n*. The Subordinate Objective Group (SOG) resides above the SO layer. SOGs model the aggregate behavior of one or more SOs. The number of SOGs in the SOG layer is determined by the parameters *m* and *v*. The parameter *v* determines the number of SO's that are combined into a single SOG. *v* must be an even divisor of *m* to maintain a consistent ruggedness transformation across the SOG layer. The number of SOGs in the SOG layer is *m*/*v* and the number of bits in a SOG after redundancy reduction is *nv*. The default value of 1 for *v* implies there is exactly one SO in each SOG.

Like the Weise model, for TOP, the optimal string for the overall solution is always defined by a set of alternating 0s and 1s. The goal of the optimization is to minimize fitness. Since values of *m* and *v* can cause objectives in the SO and SOG layers to have an odd number of bits, the optimal string in a given SO or SOG can start with either a 1 or 0. Figure 26 illustrates how an optimal individual is divided into SOs and SOGs based upon the parameters *n*, *m* and *v*. The overall optimal string for any solution always remains 010101... As such, the optimal string for alternating SOs will be 1's complements of each other if there are an odd number of bits per SO. These alternating 1's complement optimal SOs are depicted in Figure 26. A similar trend occurs for the SOGs if the number of bits per SOG is odd (not shown).

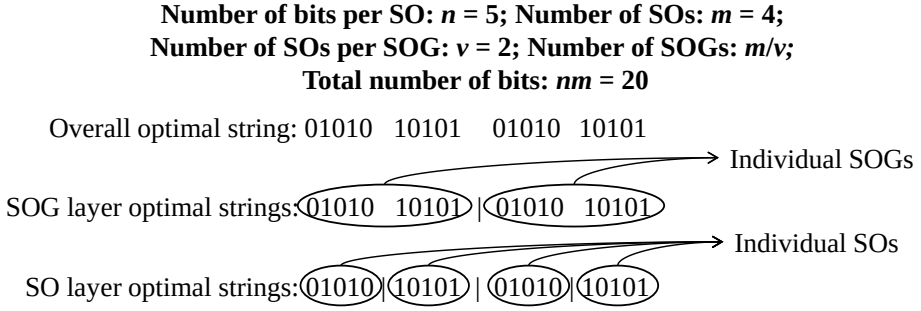


Figure 26 – An example breakdown of a 20 bit optimal solution into the SO and SOG layers with *n* = 5, *m* = 4, *v* = 2

Because there are two hierarchical objective layers in TOP, instead of the single-objective layer in the Weise model, TOP applies two ruggedness transformations. Two ruggedness transformations are required even for a simple sum-of-parts fitness function like in the TSP or JSSP because this enables TOP to model solution infeasibilities and interactions. The ruggedness transformations at the SO level are separate from the ruggedness transformations at the SOG level. The first ruggedness transformation is associated with γ1 and applies a transformation to the SO layer. The results of this transformation are only used to determine the individual objective values of the SOs and are not summed directly into the overall fitness of solutions. The second ruggedness transformation is associated with γ2 and applies a transformation to the SOG layer objective values. This transformation directly affects the overall fitness of a solution since

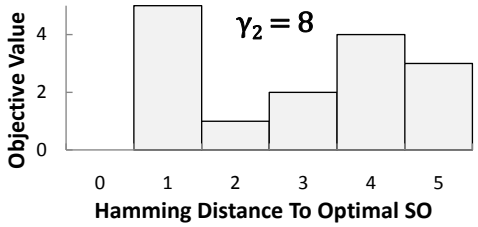
the objective values of all SOGs are added to determine the overall fitness of an individual solution. Figure 27 illustrates, for a notional solution, how Hamming distances are converted to objective evaluations, and how the final fitness value is calculated from objective values associated with the SOG layer. Note that objective values in the SOG layer are independent of the objective values in the SO layer. The Hamming distances referred to in the figure are based on the distance from the optimal solution without overfitting and oversimplification. If the overfitting and oversimplification process is active, the Hamming distances used for lookup in the ruggedness maps are based upon samples near the optimal solution. Unlike the decomposition of scalar cost functions in Handl et al. (2008a), ruggedness transformations between the SO and the SOG level do not have to be consistent with an additive fitness function model where the number of local optima in the decomposed objectives must be less than or equal to the number of local optima in the summed objectives. Allowing more local optima at the SOG level than in the SO layer enables TOP to model constraints between the SOs as poor fitness solutions. While the actual ruggedness may differ between layers, the same ruggedness transformation methodology is applied to both the SO and SOG layers.

Objective evaluation for the SO layer for $\gamma_2 = 8$

Step 1: Find Hamming distances to optimal SO bit strings.

Sample individual: 10000 | 10100 | 10010 | 10101
Hamming distances: 3 | 1 | 2 | 0

Step 2: Lookup hamming distances in the ruggedness map to obtain SO objective values.



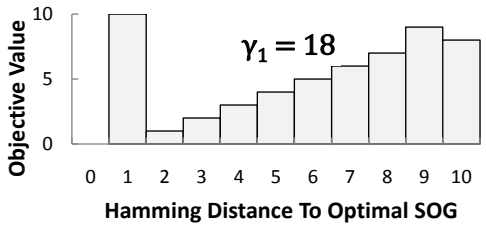
SO objective values: 2 | 5 | 1 | 0

Objective evaluation for the SOG layer for $\gamma_1 = 18$

Step 3: Find Hamming distances to optimal SOG bit strings.

Sample individual: 10000 10100 | 10010 10101
Hamming distances: 4 | 2

Step 4: Lookup hamming distances in the ruggedness map to obtain SOG objective values.



SOG objective values: 3 | 1

Fitness evaluation

Step 5: Sum objective values of all SOGs to obtain total fitness. 3 + 1 = 4

Figure 27 – Example objective and fitness evaluation(s) for the SO and SOG layers with *n* = 5, *m* = 4, *v* = 2, γ_1 = 18, and γ_2 = 8

5.3.2 Modified ruggedness transformation

A modified version of the ruggedness transformation was implemented for the TOP model. This modified transformation meets the original objectives that the mapping of Hamming distances from the optimal solution to objective values must be bijective, that $\Delta(r)$ should increase incrementally as γ increases, and that the optimal solution should be preserved at the same location in the mapping. The pseudo-code for the modified ruggedness transformation is described in Figure 28. Like the Weise model, this ruggedness transformation creates landscapes that have alternating deceptive and non-deceptive portions (Weise et al., 2008). Figure 29 illustrates how various ruggedness transformations are generated for different values of γ for a 10-bit objective. Each graph in Figure 29 is a map that assigns objective values based on the

Hamming distance to the optimal solution. Values on the horizontal axis indicate the Hamming distance while the values on the vertical axis indicate the resulting objective value contribution. Landscapes created in Phase B of the algorithm will be generally more deceptive than those created in Phase A. Landscapes generated in Phase A can be considered simply rugged rather than deceptive.

```
Integers: γ, q, steps, start, end
Array of Integers: r

r ← BuildRuggenessMap(γ, q)
begin procedure
    r ← (0, 1, 2, ... q − 1, q)
    return RuggednessMapHelper(r, γ, 1, q − 1)
end procedure

RuggednessMapHelper(r, steps, start, end)
begin procedure
  for i = 0 to end − start − 1
  begin for
    swap r[end − i − 1] with r[end]
    steps ← steps − 1
    if steps ≤ 0 return r
  end for
  for i = 0 to ⌊(start + end)/2⌋
  begin for
    swap r[start + i] with r[end − i − 1]
  end for
  return RuggednessMapHelper(r, steps, start + 1, end)
end procedure
```

Figure 28 – Modified ruggedness transformation pseudo-code

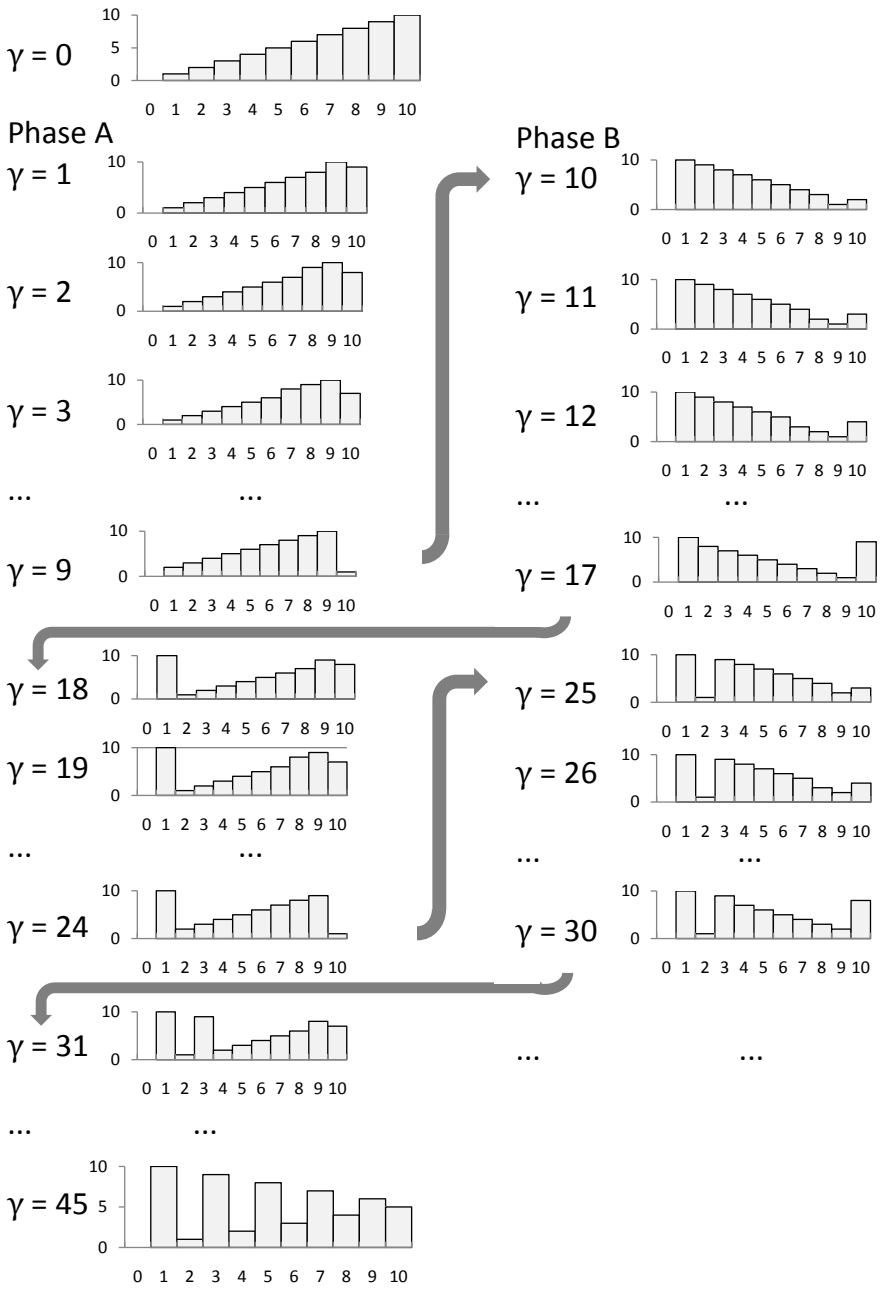


Figure 29 – An illustration of the creation of various ruggedness transformations for q = 10 with rising γ values. Each graph is a separate ruggedness transformation that maps the Hamming distance to the optimal solution to the objective value. Note the algorithm that builds ruggedness transformations can be considered in two alternating phases, Phase A and Phase B.

5.3.2.1 The difficulty of ruggedness transformations

The basic "hardness" of this landscape was established by running a 'standard' GA with a population size of 100, tournament parent selection with a tournament size of two, single-bit

mutation on all children, uniform crossover (UX), and a (μ+λ) survival selection strategy. Figure 30 shows the average number of generations required to find the optimal solution for $m = 1$ with $n = q = 80$ for different levels of γ_2. For points not plotted on the graph, a successful solution was not found in the allotted 350 generations. Due to the fact that some local optima will be in places where the algorithm will rarely traverse, only local optima that are in the likely path of the optimization, the relevant local optima, are indicated in Figure 30. For example, the most rugged landscape corresponding with $\gamma_2 = 3160$ has 40 local optima but is still relatively easy for the GA. 20 or more of those local optima are essentially irrelevant as they will be likely to be overcome by the generation of initial random solutions, and rarely sampled due to the combinatoric nature of the Hamming distance to the optimal and the selection pressure emphasizing above average fitness solutions. The standard GA must overcome the remaining 20 relevant local optima through crossover and mutation. The intervals in the graph where no solutions were found correspond to the Phase B portions which generate largely deceptive landscapes. The number of mappings in a given phase approaches 1 as γ_2 gets large. The i^{th} phase in the ruggedness transformation contains the mappings with γ parameter values $\left[(i-1)q + 1 - \sum_{j=1}^{i-1} j\,,\, iq - \sum_{j=1}^{i} j\right]$. The first phase is a special case as it contains the mapping $\gamma_2 = 0$ that starts the algorithm and as such it is associated with the γ_2 parameter values $[0, q - 1]$. Figure 30 demonstrates that a basic single-objective GA is good at overcoming two-bit sequential local optima when large deceptive regions are not present in the objective value landscape. Furthermore, each additional sequential optima adds a lower additional margin of difficulty to the problem than the previous addition. As additional local optima are added at increasing distances from the global optimum, moves that improve an even number of bits in the solution have a greater probability of occurring since many even number of bit changes lead to a fitness value improvement. Two-bit moves that overcome these local optima are generated efficiently by the GA. More generally the results in Figure 30 may indicate that GAs can overcome many, lower-order sequential optima well, possibly because

sequential fitness improvements can be easily recognized and solutions with fitness better than the best known solution always survive.

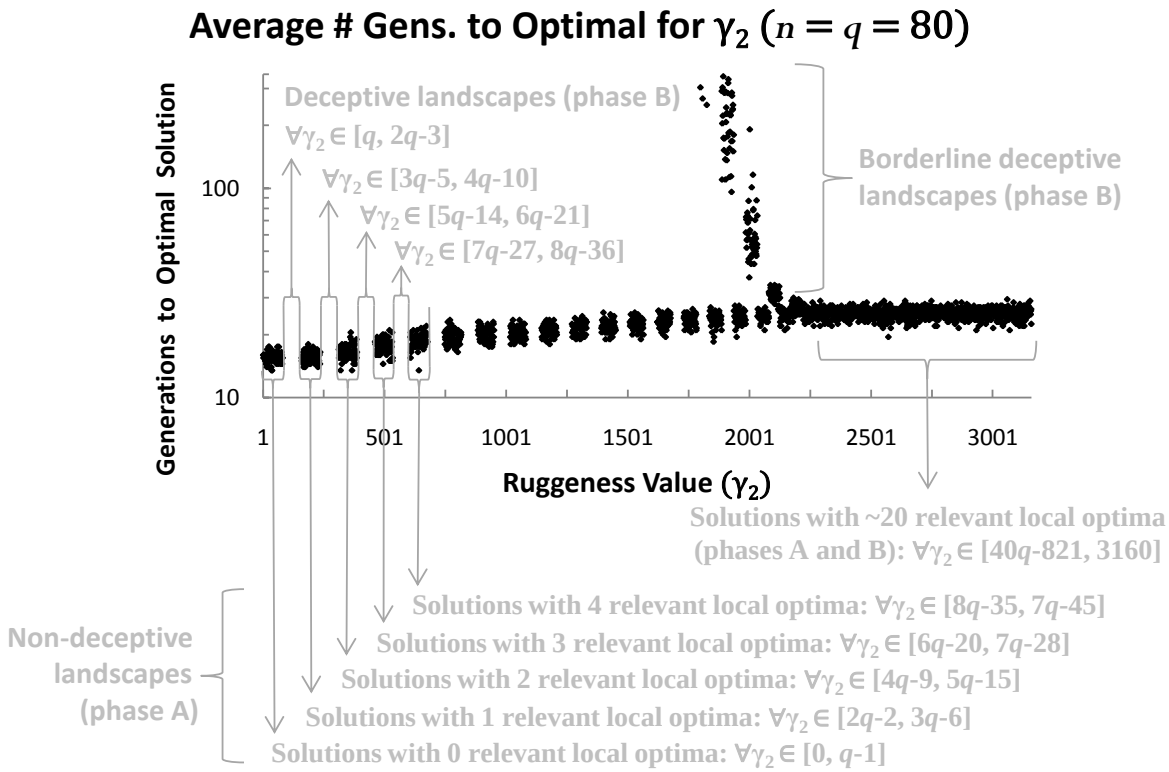


Figure 30 – Number of generations to find the optimal solution for $n = q = 80$ for all possible γ_2 values

5.3.2.2 Interpreting the difficulty of ruggedness transformations

Since the number of relevant local optima are similar for a number of consecutive ruggedness transformations it may be best to simply study the first ruggedness transformation at the start of each phase when attempting to understand mechanisms that overcome local optima (such as multi-objectivization). In studying multi-objectivization, we are more interested in the mechanisms of overcoming local optima than the mechanisms for overcoming deception. Because we were not interested in the deceptive cases, we studied the first ruggedness transformation in each group of Phase A transforms. The first transformation for Phase A is the first place where a new single local optimum is added to the problem. Assuming we would like d local optima with non-deceptive landscapes, the formula $\gamma = 2qd - 2d^2 - d - 1$ derived from

the intervals specified above can be used to find the first such Phase A transformation with *d* local optima. So, for instance, if we wanted to generate the first non-deceptive landscape with one local optimum, γ would have a value of 2*q*-2. However, this formula does not hold for the first transformation since $\gamma = 0$ is the first transformation with 0 local optima. Higher values of *d* add more optima within an objective. These optima are serial – they will generally be overcome in order. In realistic problems there are combinations of optima that must be overcome in a serial fashion and those that can be overcome simultaneously. TOP's ability to add multiple groups of objectives allows us to contrast the effectiveness of a GA in overcoming serial local optima to the effectiveness of a GA in overcoming parallel local optima.

If desired in future studies, it is possible to reorder the phases of the algorithm such that all Phase A landscapes appear first in their original order and then Phase B landscapes appear in reverse order. This reordering ranks the landscapes in terms of increasing difficulty to reach the optimal solution. The "hardness" of the ruggedness transformations will then rise with γ and transcend from an identity mapping towards ruggedness, to maximally rugged, to slightly deceptive and finally to maximally deceptive. Such transformation would order the i phases in the order $\{1, 3, 5, \ldots, \frac{q}{2} - 5, \frac{q}{2} - 3, \frac{q}{2} - 2, \frac{q}{2}, \frac{q}{2} - 2, \frac{q}{2} - 4, \ldots 6, 4, 2\}$. A similar transformation was provided in Weise et al. (2008).

5.3.3 Epistasis and objective-convolution

The previous Weise model epistasis transformation has been renamed in TOP to objective-convolution because epistasis is generally thought of as a larger concept than just the effects modeled by the Weise model epistasis transformation. One weakness of the Weise model epistasis transformation is that the amount of inter-objective conflict does not increase on a gradual scale. In the Weise model, the genotypes are divided into consecutive groups of η bits and each group forms a single gene which is transformed. Bits within a gene conflict according to the algorithm in Figure 24. The Weise model epistasis transformation causes an all-or-nothing

change in the genotype since all genes are affected by the transformation. Causing only some genes to conflict may provide additional insight into how objective interaction affects multi-objectivization. The objective-convolution in TOP allows a more gradual increase in problem difficulty.

To achieve this gradual increase in objective conflict and problem difficulty, an additional parameter τ is required. τ determines the number of genes to which objective-convolution is applied. The default value of 0 for τ implies that no objective-convolution is modeled. The maximum value for τ is $\lceil nm/\eta \rceil$ where $\lceil\ \rceil$ is the ceiling function which rounds a number up to the next highest integer. When values of η are greater than 2 and values of τ are greater than 0, some objective conflict will exist within objectives. Increasing values of τ and η increase the levels of objective-convolution in the problem. The relationship between problem difficulty for various levels of η and τ is established in Experiment 3 below.

5.4 Experiments

The objectives of the experiments are the following:

- Determine when multi-objectivization via helpers is beneficial for different general problem features.
- Determine the interrelationship between the effectiveness of different numbers of dynamic helpers and problem features.
- Determine how inter-objective conflicts change the performance of helper-methods.
- Determine the effects of layered objectives on multi-objectivization.

5.4.1 Experimental settings

A basic GA and a MOEA were used in the following experiments to explore concepts of multi-objectivization. The Non-dominated Sort Genetic Algorithm version II (NSGA-II) (Deb et al., 2002) was used as the MOEA due to its prior use in multi-objectivization research via helpers (Jensen, 2004, Lochtefeld and Ciarallo, 2010). NSGA-II ensures that the best found solution in the main objective will not be lost during the search since the main objective is always an objective of the search. NSGA-II uses the combination of Pareto ranking and a penalty function

for similar solutions to determine the results of survival and parent selection (Deb et al., 2002). NGSAII requires no parameters in addition to those required by a basic GA.

Based on recommendations set forth in Weise et al. (2008) single-bit mutation was used and was applied to every solution after creation by crossover. Even though common convention is to use a per-bit mutation scheme, single-bit mutation was used here because it prevents possibly skewing results based on the tuning of mutation rate in favor of one algorithm or another. Picking a default per-bit mutation rate such as the conventional $1/l$ where $l = nm$ was also considered but not used. The number of bits in these experiments is large (>=900) and a $1/l$ rate would have little to no impact on overcoming multi-bit optima in solutions. While multi-bit mutation can escape local optima and thus can be important to discussion on multi-objectivization, this work focused on the effects of recombination methods as these methods also overcome local optima. For recombination UX was chosen over single point crossover because it does not have positional bias. Avoiding positional bias may be important when modeling objective-convolution and layered objectives. For the basic GA a $(\mu+\lambda)$ survival selection strategy was used since it ensures that the best known solution survives and because it is the most consistent with NSGA-II. NGSAII uses the crowded comparison operator to determine fitness and a $(\mu+\lambda)$ survival selection strategy as described in Deb et al. (2002). Both algorithms used tournament selection with a tournament size of 2. The size of the population was set to 100 for all algorithms and runs. 100 new candidate solutions are created between every generation.

The TOP model and associated algorithms were implemented in C++. Since real problems often take considerably longer to perform fitness evaluation than the time the GA requires to perform survival selection and solution creation, a budget based on the number of fitness evaluations was used here. The maximum number of generations G was fixed for all experiments to $2mn$.

Prior research to date on multi-objectivization has focused on problems with highly deterministic fitness evaluations. Since we know multi-objectivization is interesting regardless of whether the problem has overfitting and oversimplification, the default values for t, *o*, and $\mathcal{E}$ were used in the overfitting and oversimplification process to ensure that a stochastic process is not modeled in the TOP.

Even with one objective and default values for all transformations, TOP contains some level of difficulty for a GA since multi-bit neutrality is present. Multi-bit neutrality exists in TOP because many solutions in the genotype space map to the same objective value. However, with the neutrality parameter μ set to the default value 1, TOP has no one-bit neutrality. μ was set to a value of 1 so that no additional neutrality was modeled in these experiments as prior multi-objectivization work has not indicated that neutrality effects are important to multi-objectivization processes.

Previous work has shown that the sequence of helpers is important to the effectiveness of helper methods (Lochtefeld and Ciarallo, 2010). Experiments 2 and 3 focus on helper methods' utility regardless of helper sequence and therefore use a simple random sequence as proposed in Jensen (2004). Two helper sequences other than random are explored and discussed in Experiment 4.

Each combination of settings in an experiment was run for 50 replications. The parameters *m*, *n*, η, τ, and γ_2 were varied by the experimental run and their values are noted as the experiments are introduced. The SO layer was exactly the same as the SOG layer in Experiments 1, 2 and 3. In those experiments *v* had a value of 1 and γ_1 had the same value as γ_2. Following previous precedent where helpers were associated with natural objective divisions (Jensen, 2004, Lochtefeld and Ciarallo, 2010), for the helper methods analyzed in Experiments 2, 3 and 4, each helper was associated with exactly one SO and was used once during the course of each run. Since *m* defines the number of SOs in the SO layer, exactly *m* helpers are used in each

experiment. A different set of dynamic helpers were swapped in every g generations based upon a random selection without replacement where $g \approx Gx/m$ and x is the number of dynamic helpers. All other details for helper swapping are consistent with the helper swapping in Lochtefeld and Ciarallo (2011b).

5.4.2 Experiments and results

Experimentation was accomplished in a phased approach where additional details were phased into TOP so that effects could be isolated to the extent possible. First, in Experiment 1, the basic problem hardness was explored as it related to multi-objectivization by executing a single-objective GA against various problem settings. In addition to understanding how the problem scales with certain parameters, Experiment 1 also explores the relative hardness of parallel versus serial local optima. Experiment 2 uses a number of comparative experiments to determine the relative algorithm efficiency for both a single-objective-GA and multi-objectivization via helpers in overcoming parallel local optima. The goal is to determine if multi-objectivization is indeed assisting in overcoming parallel local optima. Experiment 3 further studies objective-convolution to determine how genotypic conflict affects multi-objectivization via helpers. The experiment provides evidence that conflict between objectives causes weaker types of multi-objectivization to have increased performance relative to their stronger counterparts. Lastly, a final experiment explores the relationship between the SO and SOG layer. Experiment 4 demonstrates how picking helpers at the right level and picking helpers in a complementary sequence can be important to such algorithms.

5.4.3 Experiment 1 – Basic problem hardness with local optima

The basic problem hardness was first explored to determine the effects of dividing the search space into more than one SOG. Dividing the search space into more than one SOG allows us to examine the effect of adding local optima in parallel locations in the search space. The difficulty of parallel landscapes where a number of SOGs are used to introduce local optima is contrasted

with the difficulty of a single SOG landscape where serial optima are introduced. In this experiment the basic GA was used to assess the difficulty of the problem.

5.4.3.1 Experimental setup

The total string length of *nm* was set to 3000 bits implying a total search space size of 2^{3000}. For this experiment the parameters τ and η were set to default values to eliminate objective-convolution from contributing to problem hardness. The number of bits in a given SOG, *n*, was calculated from values of *m* using the equation $n = 3000/m$.

Algorithm success was measured both in terms of how quickly the algorithm was able to find the optimal solution (when it found the optimal one) and how effective the algorithm was at getting close to the optimal solution. $n = q$ since overfitting and oversimplification were not modeled in these experiments. For the parallel landscapes with a varied number of SOGs, γ_2 was given the value of either 0 or $2q$-2 where the $2q$-2 ruggedness transformation results in a landscape that has the worst objective value in the landscape as a 'hill' to overcome in order to reach the optimal value in that objective. Figure 31 shows the resulting $2q$-2 ruggedness transformation for a notional 40 bit string with 1, 2, 4, and 8 SOGs. This figure demonstrates how the $2q$-2 transformation adds parallel local optima for a fixed length string and various levels of *m.* For all cases, the landscape is largely non-deceptive but to achieve the optimal solution in a SOG, the GA must find a way to move past solutions that are exactly one bit from SOG optima since those solutions have the worst possible objective value. Like all single-objective solutions in TOP, in the example in Figure 31, fitness for any solution is the sum of objective values for all of the SOGs. The ruggedness landscape with $\gamma_2 = 2q$-2 creates one relevant local optimum within a SOG. This SOG local optimum has a Hamming distance of 2 from the SOG optimal solution and has a mapped objective value of 1.

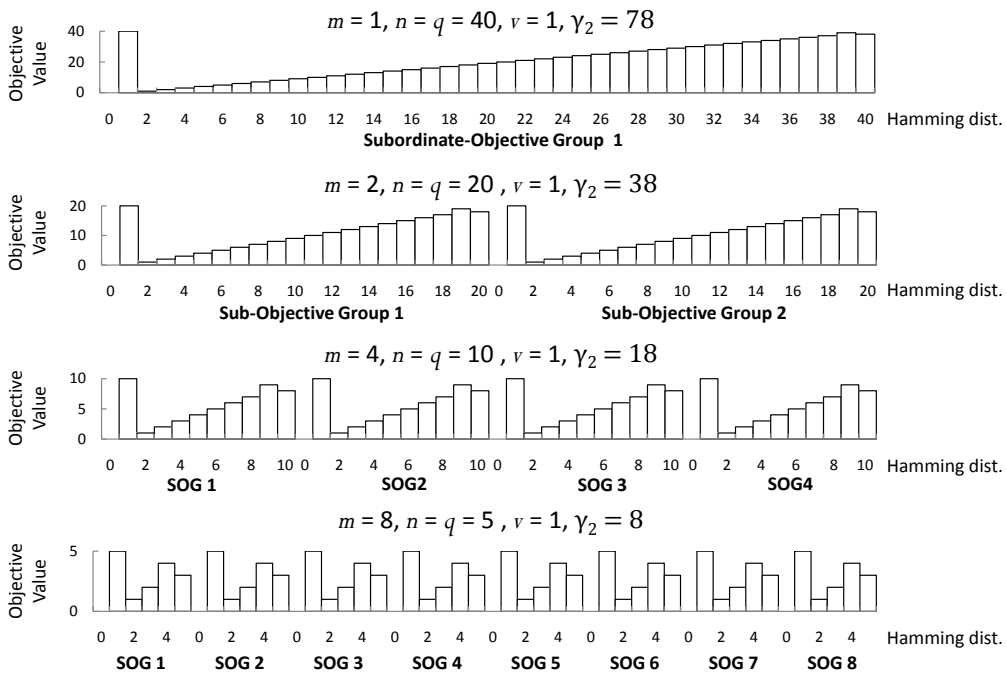


Figure 31 – Ruggedness transformation for various *m* with *nm* = 40, *n* = *q*, *v* = 1, $\gamma_1 = \gamma_2 = 2q\text{-}2$. This figure illustrates how a problem with a fixed number of bits is divided into one or more parallel local optima by changing the values of γ_2 and *m*.

For parallel landscapes where the number of objectives was varied, two ruggedness transformations were explored. The landscapes with $\gamma_2 = 0$ introduce many parallel linearly decreasing landscapes but do not introduce local optima to be overcome. The landscapes with $\gamma_2 = 2q\text{-}2$ introduce one local optimum per objective group. For the $\gamma_2 = 2q\text{-}2$ landscapes the number of local optima in a given problem is equal to the number of SOGs (m). In the parallel landscapes the number of objectives, m, was given the values of all the factors of 3000 except $m = 1500$ and $m = 3000$. As m approaches 3000, the size of the SOGs approaches 1 bit. For the $\gamma_2 = 2q\text{-}2$ cases at $m = 1500$ and $m = 3000$, the $2q\text{-}2$ ruggedness transformation is degenerate. For these two cases, the SOGs have only 2 and 1 bits respectively which implies that no local optima can be present in the SOG.

For the landscapes with serial optima, only one SOG is present so m equals 1 and the number of local optima is controlled by changing the ruggedness parameter γ_2 with γ_2 having values of $2dq - 2d^2 - d + 1$ where d is the number of local optima. We are interested in the

relative comparison of difficulty between problems with the same number of parallel local optima and serial local optima, and so d was given the same values as the values of m in the parallel optima cases. Since the overall search space is the same size for both the parallel and serial landscapes and the parameters m for the parallel local optima and d for the serial local optima both introduce the same number of local optima, when m for the $\gamma_2 = 2q$-parallel landscapes equals d for the $\gamma_2 = 2dq - 2d^2 - d + 1$ serial, we can directly compare the difficulty of problems with serial versus parallel optima.

5.4.3.2 Results and analysis

Figure 32 demonstrates results of average best found fitness for three different ruggedness transformation strategies. Linearly decreasing landscapes are compared with landscapes that add serial local optima and landscapes that add parallel local optima.

For the parallel landscapes with no local optima ($\gamma_2 = 0$) and different values of m, the ruggedness mapping makes no changes in the fitness based on the Hamming distance from the optimal solution. The ruggedness transforms with $\gamma_2 = 0$ result in one or more linearly decreasing sections with no local optima. One-bit changes that move a solution closer to the optimal solution also improve the fitness by 1. As a result, the GA always finds the optimal solution at roughly the same number of generations regardless of the number of SOGs. The number of SOGs does not matter since all similar bit moves reflect the identical change in overall fitness. Splitting the basic problem with no local optima does not make the problem more difficult.

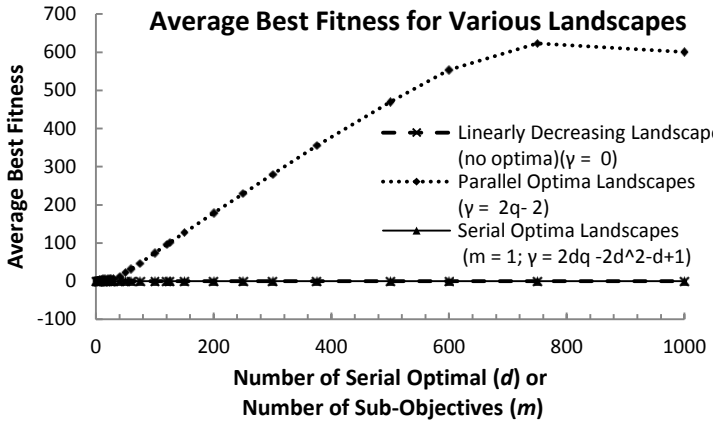


Figure 32 – Average best fitness by number of SOGs for $\gamma_2 = 0$ and $\gamma_2 = 2q\text{-}2$. Lower average best fitness values are better. Both the serial optima landscapes and the linearly decreasing landscapes find the optimal solution on all runs and settings.

For $\gamma_2 = 2q\text{-}2$, as shown in Figure 32, the addition of each SOG adds an additional local optimum exactly 2 bits from the best solution in that SOG. For the basic GA, there is an overall increase in average best fitness as the number of SOGs, *m*, increases. This demonstrates that the problem generally gets harder from the addition of local optima in different SOGs and that, as more SOGs add more parallel local optima, the problem difficulty continually rises. In the range of *m* values between 50 and 500, the addition of parallel local optima appears to increase problem difficulty in a linear fashion. Results of these two cases are not reported. Observe that the average best fitness value decreases from the $m = 750$ case to the $m = 1000$ case. The cause of this downturn was explored and is discussed later in this experiment.

For the serial optima case where $m = 1$ and $\gamma_2 = 2dq - 2d^2 - d + 1$, the GA always found the optimal value in the allotted 6000 generations. This is in stark contrast with the parallel optima case where a similar number of local optima are added. We can see further evidence of this in Figure 33.

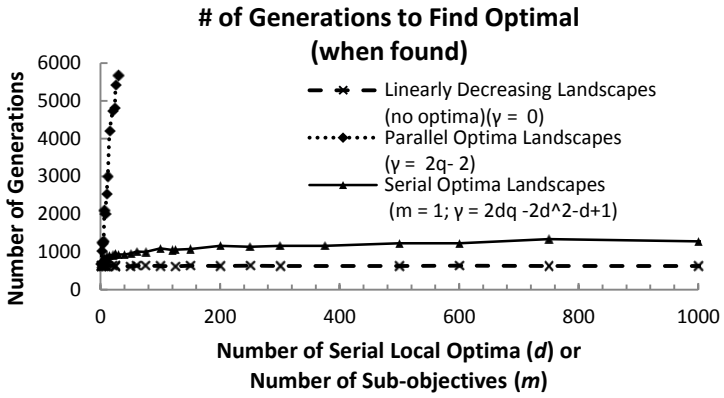


Figure 33 – Average number of generations to find the optimal solution. This figure illustrates how the TOP scales in difficulty with the addition of SOGs that have zero and one local optimum present ($\gamma 2 = 0$ and $\gamma 2 = 2q-2$ cases respectively). The graph also shows how difficulty scales with the addition of serial local optima.

Figure 33 shows the average time required to find the optimal solution by the standard GA (when it was found) by ruggedness parameter value and number of SOGs. In the cases of serial local optima where $m = 1$ and $\gamma_2 = 2dq - 2d^2 - d + 1$ the difficulty of the problem is exponentially saturating. This is due to the way in which the algorithm overcomes serial local optima. Suppose an arbitrary solution with 1 SOG and 5 serial local optima ($d = 5$; $\gamma_2 = 10q - 191$) is 10 bits from the optimal solution in Hamming distance. To sequentially overcome these local optima the algorithm must generate solutions that are an even number of bits closer to the optimal solution. The algorithm can change any 2 of the 10 bits to overcome the first local optimum. Then the algorithm can change any 2 of remaining 8 bits to overcome the next local optimum and so forth until the optimal solution is found. For serial optima, TOP allows changes in any two bits out of the bits with incorrect values to overcome a local optimum. In contrast, for parallel landscapes with a single local optimal objective value per SOG, the algorithm must change precisely the two specific bits that have an incorrect value and are associated with a single SOG to improve the associated objective value.

Intuitively for the parallel optima cases were $\gamma_2 = 2q-2$, the number of generations required to find the optimal solution increases rapidly as the number of SOGs increase. The GA did not find the optimal solution for m values greater than 30 and n values greater than 2 as the problem gets

more difficult with the addition of many parallel local optima. For all $\gamma_2 = 2q\text{-}2$ cases, convergence to the local optimum in each SOG was relatively quick with typically about 10% of generations used to achieve fitness at or below the total number of objectives. To understand the general fundamentals behind the difficulty of ruggedness transforms in TOP, we examined the data further.

5.4.3.2.1 Deeper exploration into the difficulty of ruggedness transformations

Assuming the GA can easily climb-down the primary hills in the $2q\text{-}2$ cases to the local optima, we can use the average final best fitness value and the number of optima per objective to examine the percentage of local optima overcome by the algorithm. It was expected that as the number of local optima increased, the GA would perform increasingly worse at overcoming a high percentage of those local optima. An examination of percentage of local optima overcome revealed additional detail about the nature of TOP for the $2q\text{-}2$ ruggedness transformation. For these cases, the percentage of local optima overcome with respect to the number of objectives was plotted and is shown in Figure 34. The figure also indicates the effective change in certain features of the problem and algorithm as indicated by the scales below the figure. Each of these forces has an effect on the phenomena in the chart portion of the figure. A downward trend in the algorithm's effectiveness at overcoming SOG local optima is present implying that as the number of local optima increases, the problem gets more difficult. However, there are also two interesting phenomena in the graph counter to this overall trend. Observe the local maximum around m values of 20 to 30 and the increasing percentage of local optima overcome for m values greater than 375. These phenomena are discussed next.

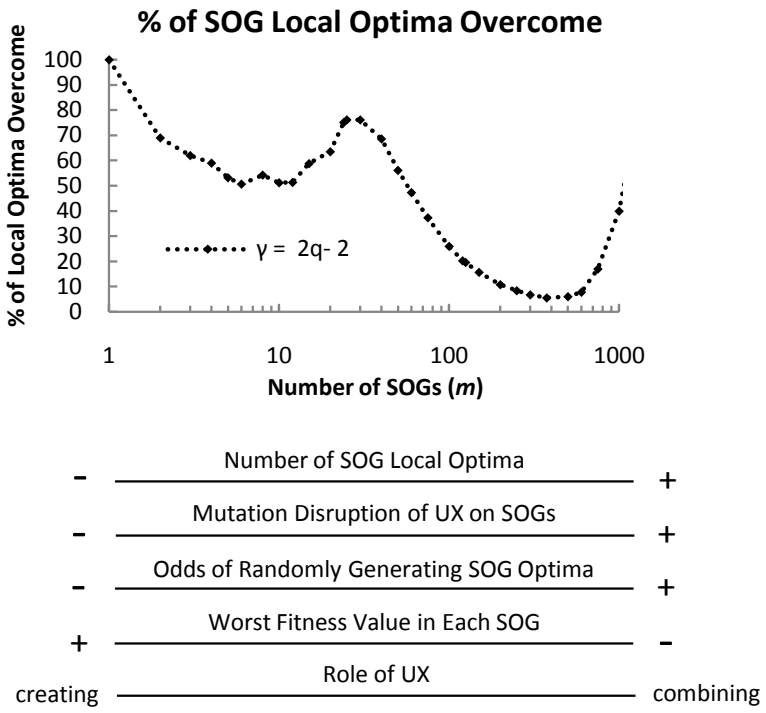


Figure 34 – Percentage of local optima overcome by number of SOGs for $\gamma_2 = 2q\text{-}2$ cases. The scales at the bottom of the figure describe how different effects change with the number of SOG local optima.

Observe the maximum in the graph in Figure 34 around *m* values of 20 to 30. In this region the percentage of SOG local optima overcome is greater than the neighboring problem divisions. The number of times a SOG optimum was created when that optimum was not present in either parent was tracked to determine the cause of this phenomenon. Data was recorded for the number of SOG optima created by UX, the number created by UX and then lost by mutation, and the number created by the combination of UX and mutation. Figure 35 shows the effect of UX alone and the net effect of UX with mutation for the $\gamma_2 = 0$ and $\gamma_2 = 2q\text{-}2$ settings. Clearly UX is highly effective at creating SOG optima for the $m = 1$, $\gamma_2 = 0$ case. Here, UX alone creates 3 orders of magnitude more optimal solutions than UX with mutation. Since the $\gamma_2 = 0$ cases are simple linearly decreasing landscapes, this fact is not surprising. Single-bit mutation, for low values of *m*, is highly disruptive as it causes many SOG optima that were found in crossover to be eliminated. As the number of SOGs increase, the difference between the UX effect and the

combined effect diminishes. Mutation becomes much less disruptive since the number of SOGs is increasing and mutation only affects one SOG for every solution created. In summary, the local maximum in the graph around *m* values of 20 to 30 is caused by two opposing forces. These forces are the force of making the problem harder by adding more local optima and the force of reducing the effectiveness of crossover by mutation.

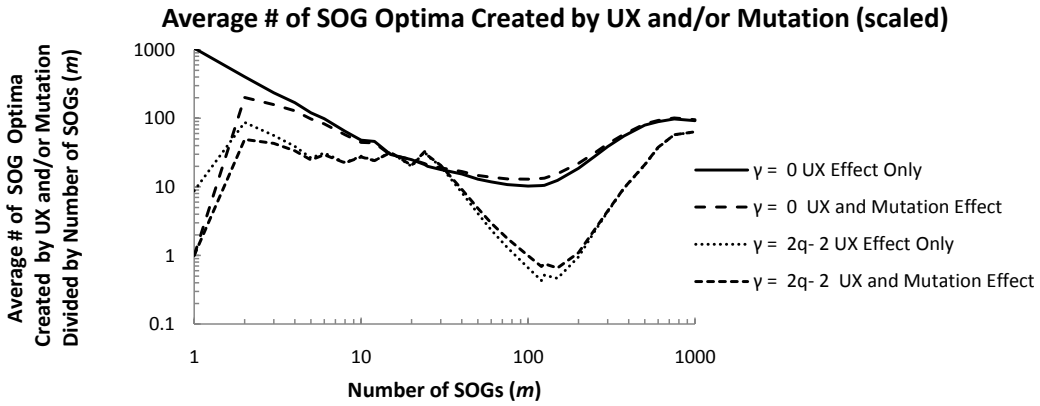


Figure 35 – Effect of UX only and UX with mutation. This graph illustrates how single-point mutation is highly disruptive for a small number of parallel local optima but effective when a higher number of parallel local optima are present.

5.4.3.2.2 Interpreting Large *m* Value Cases

In Figure 35, for *m* values larger than 375 objectives it appears to be easier to create SOG optima. There are two predominate forces at work in this range: random SOG optimal creation and hill-standing. When a SOG is large, it is typically difficult to find the SOG optimum by simple random solution generation due to the combinatoric nature of Hamming distances from a single arbitrary solution. However as the number of bits in the SOGs gets small, local optima are more likely to be overcome by pure random chance. As many more SOG optima become available in the population, the role of crossover changes from creating optima within SOGs to combining already existing good SOG solutions. In all but the extreme cases UX simultaneously does some of both the combining and creating actions. Since we are interested in structured problems rather than random search performance here, studying ruggedness when the number of bits per objective *n* is absolutely small is uninteresting.

The second, and more significant force at work in the tails of Figure 34 and Figure 35 is the concept of 'hill-standing'. The magnitude of the objective value between the local SOG optima and global SOG optimum can be important. As the height of the 'hill' between the two points gets small in comparison to the average fitness in the population, some solutions can be competitive on total fitness while having one or more SOGs that are 'standing' on the hill adjacent to the global SOG optimum. These solutions are each exactly one Hamming distance from the global SOG optimum and can 'overcome' the local optima with a one bit move. Figure 36 gives evidence of hill-standing occurring in this experiment. The figure graphs the percentage of SOG optima created by parents that are not already optimal in a sub objective. These percentages are reported by the Hamming distance between the parents that created the SOG optimum. For the $\gamma_2 = 0$ ruggedness case, SOG optima are most often created by parents that differ by exactly 2 bits. Since no parent can have the SOG optimum already represented, each parent must be exactly 1 bit from the optimal solution in that SOG. In contrast for the $\gamma_2 = 2q\text{-}2$ ruggedness cases, parents with Hamming distance of 2 or 3 apart almost never create an optimal solution until the number of SOGs exceeds 100. The presence of local SOG optima keeps parents from having components that are one bit away from the optimal solution. As a result, the GA must combine parents that differ by 4 or more bits to generate a SOG optimum. When the number of SOGs gets large, the height of the hill between the local SOG optima and the SOG optima gets small. The shorter hills allow parents to survive while 'standing' on one or more hills as evidenced in the $\gamma_2=2q\text{-}2$ ruggedness cases as m gets large. In these cases there is a marked increase in the frequency of SOG optimal solutions created by parents that differ by only 2 or 3 bits. Hill-standing suggests that it is important to keep n relatively high with respect to m if local SOG optima are intended to be difficult to overcome.

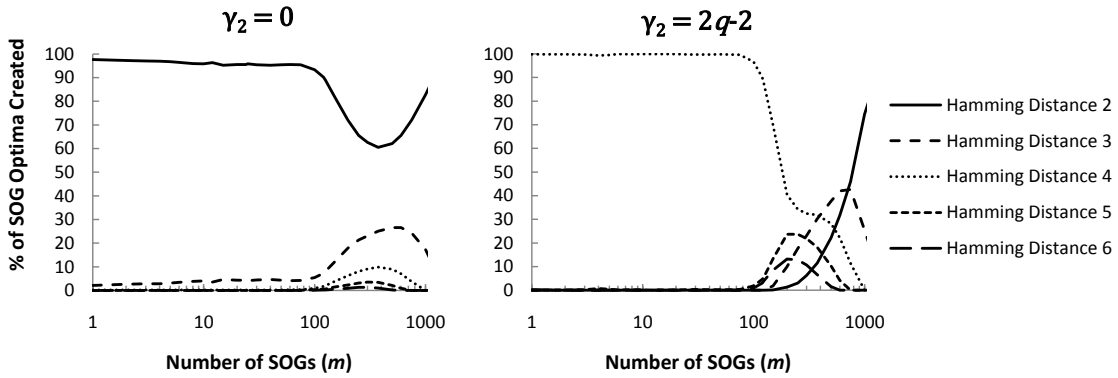


Figure 36 – Percentage of solutions created by crossover by Hamming distance between parents. This figure shows that as the number of SOGs gets large, solutions bypass local optima by surviving and mating solutions that have SOGs that are at a local maximum.

The two forces of randomly generated SOG optima and hill-standing together contribute the phenomena in the tail of Figure 34 to the previously mentioned decreasing average best fitness when moving from the m = 750 to the m = 1000 case in Figure 32. These two phenomena imply that to study an algorithm's ability to overcome local optima, it is best to avoid studying values of n that are absolutely small or smaller by more than an order of magnitude in relation to values of m.

5.4.3.3 Conclusion

With an established understanding of the problem difficulty as it relates to number of SOGs and overcoming of local optima, we can begin to study the differences between a basic GA and multi-objectivization via helpers. Because the difficulty of the problem scales linearly with the addition of parallel local optima and because the fitness function for parallel local optima can be decomposed, parallel objectives with a single local optimal value are studied in the remaining experiments.

5.4.4 Experiment 2 – Multi-objectivization on the basic problem with local optima

Jensen (2004) hypothesized that multi-objectivization helped algorithms overcome local optima present in problems. With a relatively simple problem with v = 1 and using the 2q-2

ruggedness transformation to generate a number of local optima, we can detect benefits from multi-objectivization. A number of helper methods using NSGA-II were run against the basic GA in this experiment.

5.4.4.1 Experimental setup

Practical experiments on the TSP and JSSP have used between 10 and 50 helpers through the course of the run (Jensen, 2004, Jahne et al., 2009, Lochtefeld and Ciarallo, 2011b). To cover this range, *m* in this experiment was given all values between 10 and 100 that are evenly divisible by 10. The number of bits in each objective, *n*, was set to 30 because finding the optimal value in a 2^{30} search space through random samples is highly unlikely given less than a million samples are taken per replication. The probability of finding the optimal solution in an objective for a single random sample is $9.31*10^{-10}$. Also, with an *n* value of 30 and *m* values no greater than 100, the phenomenon of hill-standing should be minimal.

Previous research indicated that 1 and 2 dynamic helpers were superior to more than 2 dynamic helpers for the problems studied (Jensen, 2004, Lochtefeld and Ciarallo, 2010). A higher number of dynamic helpers produced a significantly worse search as "when many helper-objectives are used simultaneously, the disadvantage of bad moves outweighs the advantage of escaping local optima" (Jensen, 2004). For this experiment between 1 and 10 dynamic helpers were tested.

5.4.4.2 Results and analysis

Results from this experiment are shown in Figure 37. This graph shows that multi-objectivization can be advantageous for the 2*q*-2 landscapes. The 100 SOG cases have the worst fitness, the 90 SOG cases have the next worst fitness, and so on which implies that as the number of objectives *m* gets larger, the problem increases in difficulty due to a larger search space and the addition of local optima. The 1- through 5- dynamic helper cases outperformed the basic GA for all but the easiest problem with an *m* value of 10. Out of the helper methods, the 5-dynamic

helper method was best, the 3-dynamic helper was second best, the 1-dynamic helper method was third best, and the 4- and 2-dynamic helper methods were a trailing fourth and fifth respectively. The 5-dynamic helper method was equivalent or better than any other method for all of the cases. A t-test reveals that the 5-dynamic helper algorithm performed statistically better than the basic GA at the $\alpha = 0.01$ level for all but the easiest problem.

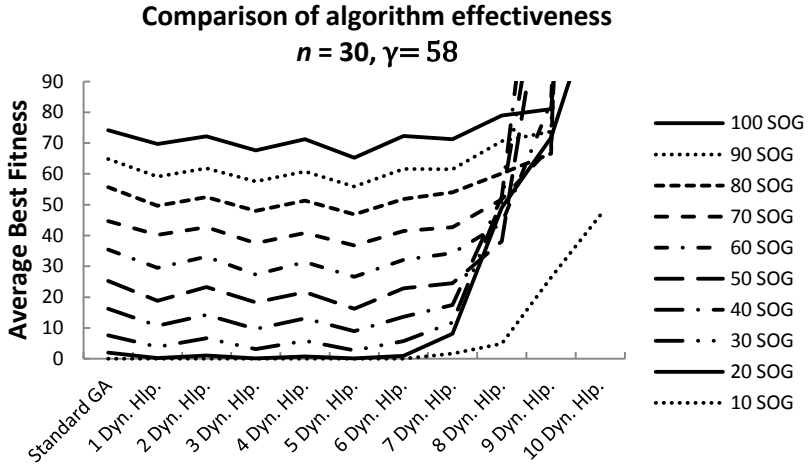


Figure 37 – Comparison of multi-objectivization on SOG local optima landscapes. This figure shows the utility of multi-objectivization in a problem with no objective conflict. The effectiveness of overcoming 10 to 100 parallel local optima is displayed for various numbers of dynamic helpers and the standard GA

With 20 SOGs in Figure 37, in this case the basic GA does not consistently overcome the 20 local optima in the allotted 6,000 generations. However, as earlier noted in Figure 30, in that case the basic GA was able to consistently overcome ~20 local optima in 350 generations. This raises the question: Why can the earlier GA overcome 20 local optima in 350 generations when the basic GA here has difficulty with 20 local optima in 6000 generations? While the search space is larger here, it is also less deceptive. In both landscapes, a 2 bit move is required to overcome a local optimum. The answer lies in the fact that in the first example, the GA positively knows when it overcomes local optima as the action always produces a solution with improved fitness. However, when SOGs are summed up as in the example here, a positive benefit found in one SOG can be unrewarded if one or more offsetting negative moves happen in other SOGs. The positive moves (signal) can be drowned-out by the negative moves (noise). Helper methods can

improve algorithm effectiveness because they are able to differentiate the signal from the noise. The recognition and survival of signal is the reason helper methods performed better than the basic GA. Noise and signal can happen at many levels through the problem and it is important to pick helpers that reward significant signal while ignoring significant noise. Signal-to-noise is a likely reason for why optimization of helper size may be important. Problems like the TSP and JSSP that use a sum-of-parts fitness function can be easily divided into SOGs and thus can benefit from improvement in signal-to-noise that is present in the standard optimization of the main objective. Of course, in a real problem things are not quite so simple and a positive move in a SOG may actually be a negative move overall. As long as there is some degree of alignment between improvements in the SOGs and main objective, multi-objectivization may be useful.

In Figure 37 it is evident that the helper methods with an odd number of dynamic helpers generally outperformed their even numbered neighbors. This phenomenon was explored but no evidence was found that linked the change in performance to a likely cause in either the TOP model or the NSGA-II algorithm.

For more than 6-dynamic helpers the algorithm becomes much less efficient. In this region too much focus is taken away from the main objective and put on trade-offs between SOGs which causes the algorithm to converge slowly toward good solutions. This phenomenon of highly diminished returns for a large number of dynamic helpers has been previously observed after only one or two dynamic helpers whereas here it occurs much later.

5.4.4.3 Conclusion

Clearly this experiment demonstrates the benefits of multi-objectivization using a small number of dynamic helpers. This experiment highlights how isolation of fitness improvements (signal) from fitness decrements (noise) can lead to improved results. However, it is unrealistic to expect zero interdependencies between objectives. These simple landscapes lack conflict present

in a more realistic problem. One way of adding interdependencies to TOP is through objective-convolution.

5.4.5 Experiment 3 – The effect of objective-convolution on multi-objectivization

Experiments 1 and 2 studied SOGs that did not have any level of conflict. In reality if we had such a problem we could simply decompose it into the SOG parts and solve each part individually. However, this is not practically possible for real problems studied by multi-objectivization because some level of conflict exists between the parts of the decomposed objective. Therefore, an important feature in multi-objectivization is the level of trade-off or conflict within and between objectives (Lochtefeld and Ciarallo, 2011b). Objective-convolution attempts to model inter-objective conflict since it transforms consecutive bit sections and the bits of objectives are interleaved in the genotype.

5.4.5.1 Experimental setup

To study the relationship between multi-objectivization and objective conflict, we need to understand the basic hardness of a problem with respect to objective-convolution. The basic GA as introduced previously with TOP parameter values of $m = 30$; $n = 30$; $\gamma_2 = 0$ and $\gamma_2 = 2q\text{-}2 = 58$ was run for selected levels of τ and η. Results for the average best fitness are shown in Figure 38. In the $\gamma_2 = 0$ cases, within an η level, the average best fitness increases relatively linearly with respect to τ. The basic problem with no epistasis always achieves the optimal solution of 0. The relative linear increase in problem hardness in TOP should allow us to simulate problems of various tunable difficulties through the use of the τ and η parameters.

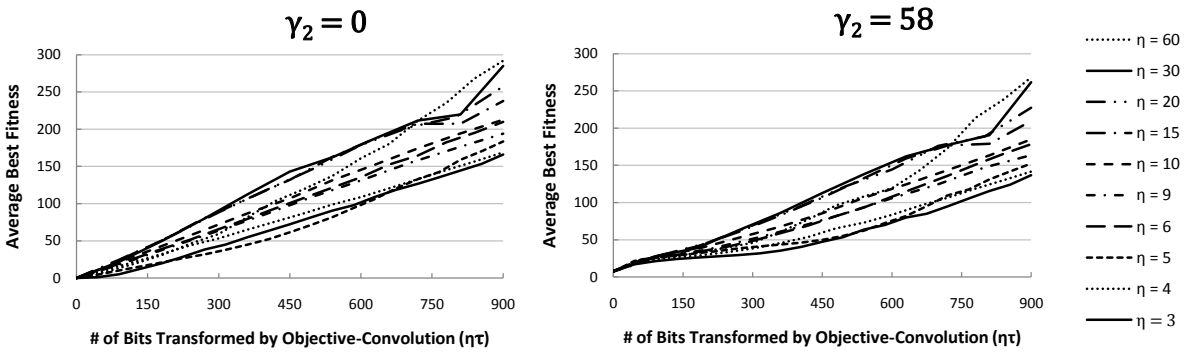


Figure 38 – Objective-convolution hardness for TOP with *n* = 30, *m* = 30. This graph illustrates how objective conflict created by the objective-convolution transformation scales with various levels of η and τ for linearly decreasing landscapes ($\gamma_2 = 0$) and landscapes with one parallel local optimum ($\gamma_2 = 58$).

For the $\gamma_2 = 2q\text{-}2 = 58$ graph each of the 30 SOGs contains local optima. These local optima cause the problem to be difficult even when objective-convolution is not active for $\tau = 0$ and as a result the easiest cases still have an average best fitness greater than 0. Increased levels of τ appear to increase difficulty in roughly the same linear fashion as the $\gamma_2 = 0$ cases. However, for the highest levels of τ, the optimization process converges to average best fitness values roughly 30 less than in the $\gamma_2 = 0$ cases. The objective-convolution transformation is causing roughly as much difficulty in both the $\gamma_2 = 0$ and $\gamma_2 = 58$ cases. Since the $2q$-2 ruggedness transformation causes the average fitness of a random solution to drop almost 30 points, the resulting "ceiling" is reduced accordingly. The epistasis transform is primarily governing the problem difficulty with higher levels of τ. Comparing the average best fitness across different ruggedness transformations can be misleading with large amounts of objective-convolution since objective-convolution is so highly disruptive with high values of τ. We know that real problems have some regularity to them – otherwise they would not be amenable to heuristics. As a result, study of the highest levels of objective-convolution may be impractical. However, because we do not have a good way to know what largest realistic objective-convolution transformations should be studied; we examined a range of transformations next.

For the n = 30, m = 30 problem, multi-objectivization was studied for various epistasis levels using the $\gamma_2 = 0$ and $\gamma_2 = 58$ ruggedness transformations. As greater than 7 dynamic helpers resulted in significantly worse results in Experiment 2, up to 7 dynamic helpers were used here. An η value of 5 was used since it was believed that a moderate amount of inter-objective interactions is a more realistic setting with respect to modeling a real problem. Values of τ were set to 0, 6, 12, 18, 24, 30, 45, 90, 135, and 180. The sequence of helpers used was random. Better helper sequences may exist since an η value of 5 does not make all objectives entirely symmetric with each other.

5.4.5.2 Results and analysis

Table 14 and Table 15 show the results of these runs. Values in bold are the average best fitness for algorithm performance for the given level of τ. Grey values in the tables indicate that the value is statistically different than the corresponding bolded value. Differences were checked using the Z-test at the $\alpha = 0.05$ level. The introduction of local optima into the SOGs (shown in Table 15) appears to make a larger number of dynamic helpers more useful as seen by the comparison of helper effectiveness between the tables.

Table 14 – Average best fitness for n = 30, m = 30, η = 5, γ2 = 0 and various levels of τ

Algorithm	τ = 0	τ = 6	τ = 12	τ = 18	τ = 24	τ = 30	τ = 45	τ = 90	τ = 135	τ = 180
Basic GA	**0**	3.1	6.6	10.3	13.6	16.6	27.0	**60.5**	**117.7**	186.1
1 Dyn. Hlp.	**0**	2.1	4.9	8.4	11.4	**15.0**	**25.1**	68.1	123.6	**182.9**
2 Dyn. Hlp.	**0**	**1.6**	**4.7**	8.2	**11.2**	15.3	27.6	76.3	135.1	195.4
3 Dyn. Hlp.	**0**	2.2	5.4	8.8	11.3	15.8	27.1	77.7	138.9	200.8
4 Dyn. Hlp.	**0**	1.7	4.9	**7.7**	12.1	16.1	29.0	83.0	147.8	206.4
5 Dyn. Hlp.	**0**	2.7	5.4	8.5	11.9	16.5	29.4	89.1	151.5	212.0
6 Dyn. Hlp.	**0**	2.4	6.3	9.5	15.5	20.1	37.8	109.1	219.4	283.8
7 Dyn. Hlp.	**0**	3.6	9.6	16.5	24.7	38.2	128.5	257.7	299.4	322.1

Table 15 – Average best fitness for n = 30, m = 30, η = 5, γ2 = 58 and various levels of τ

Algorithm	τ = 0	τ = 6	τ = 12	τ = 18	τ = 24	τ = 30	τ = 45	τ = 90	τ = 135	τ = 180
Basic GA	7.8	18.6	22.8	25.9	27.5	29.2	34.9	49.9	**90.9**	153.4
1 Dyn. Hlp.	3.7	**14.2**	20.5	23.7	26.7	28.2	34.1	**49.4**	94.8	**152.2**
2 Dyn. Hlp.	6.7	16.6	21.2	23.5	25.4	27.5	33.3	56.8	106.9	163.9
3 Dyn. Hlp.	2.9	14.2	19.3	22.5	24.6	26.9	33.3	62.4	112.1	170.2
4 Dyn. Hlp.	5.1	15.2	19.2	22.2	**24.0**	26.5	**32.7**	66.0	121.5	182.4
5 Dyn. Hlp.	**2.6**	15.1	**18.4**	**21.3**	24.3	**25.8**	32.9	72.7	126.7	190.5
6 Dyn. Hlp.	5.9	15.7	18.5	21.5	24.2	27.6	34.3	79.5	181.6	238.6
7 Dyn. Hlp.	12.4	19.3	22.9	25.1	27.4	30.0	63.1	207.6	257.9	278.3

The results in Table 14and Table 15 show that multi-objectivization can give improved results over the basic GA with low to moderate levels of objective-convolution. However, as the amount of objective-convolution passes 25 percent of the genotype, the basic GA becomes much more competitive with respect to many of the helper methods and its only significant competitor is the 1 dynamic helper method. This implies that as a problem becomes highly constrained and small moves cause many changes in the structure of the solution, helper methods will perform relatively poorly compared to the basic GA. In an SNR sense, as the level of conflict between decompositions increases, less aggressive forms of multi-objectivization are required since it becomes more difficult to identify signal from noise. Fortunately for multi-objectivization, many solution representations and crossover methods are built to maintain solution structure and properties when performing recombination and mutation.

5.4.5.3 Conclusion

This experiment demonstrated how objective conflict causes the basic GA to become more competitive with multi-objectivization as the level of conflict rises between objectives because it becomes more difficult to distinguish signal from noise. Improved SNRs may not be the only reason to use multi-objectivization. Experiment 4 explores another reason for why multi-objectivization can give improved results: multi-objectivization can reward moves that avoid interactions within a problem.

5.4.6 Experiment 4 – Breakage of multi-level epistasis

Lochtefeld and Ciarallo (2011b) theorizes that the breakage of epistasis is a major reason that helper-methods have improved performance on certain problems. To test this theory, multiple ruggedness transformations at the SO and SOG levels were used. The multiple levels allow us to model problem constraints as solutions with poor objective values.

5.4.6.1 Experimental setup

For this experiment the number of bits in a solution was set to 1800. All SOGs in the SOG layer contained exactly 30 bits with a 2*q*-2 ruggedness transformation. The number of objectives in the SOG level was always 60. These settings assure that the main objective is identical for all cases analyzed. Figure 39 shows several comparisons with various helper sequences and ruggedness transformations at the SO level. The legend in this figure shows pictorially in each case how the supporting SOs are mapped to objective values for the helpers and how these SOs are indirectly aggregated into a SOG. The Basic GA only uses the main objective and thus the cases that vary the SO layer have no impact on performance. However, for the helper methods we see a difference in performance across the different cases.

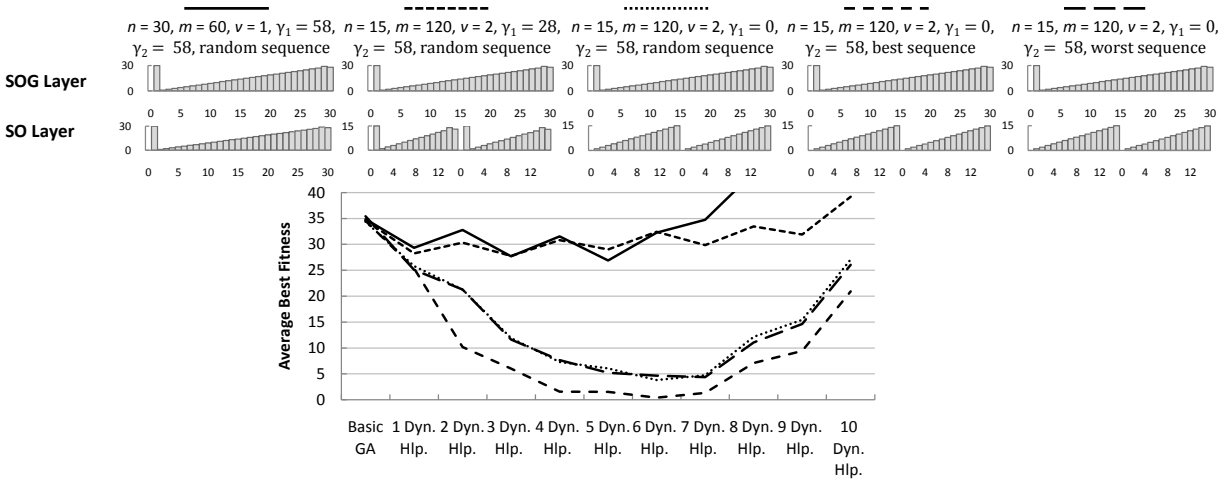


Figure 39 – Effectiveness of various methods against several different multi-level landscapes.The key illustrates the ruggedness mappings for the SOG and SO layers associated with each case.

The settings for the base case were $n = 30$, $m = 60$, $v = 1$, $\gamma_1 = \gamma_2 = 58$, and random helper sequencing. In the base case in this experiment the SO layer is exactly equal to the SOG layer. This case corresponds directly to the settings for the 60 SOG case in Figure 37 and demonstrates the signal-to-noise improvements caused by multi-objectivization. The case $n = 15$, $m = 120$, $v = 2$, $\gamma_1 = 28$, $\gamma_2 = 58$, and random helper sequencing corresponds to a situation where two local optima in the SO layer turn into one larger optimum in the SOG layer. Comparing these cases in Figure 39 we see that the additional helpers at the SO layer do not result in improvements in performance for the most viable helper methods. Similar optima to the one in the SOG are present in each SO and thus improvements in performance would be unexpected. Note that this case is relatively viable for the 8 and 9 dynamic helper cases whereas the base case is not. Comparing this case to the base case we can see that the number of helpers increases by a factor of 2 while the number of generations for the analysis stays the same. This causes more helper swaps to occur. Since helpers are more transient and the main objective is consistently an objective, adding more helpers brings the focus of the algorithm back to the main objective. Frequent helper swaps decrease their influence on the overall progress of the optimization process. Since the main objective is not swapped and remains as one constant force throughout the evolution, its relative influence increases. Note similar phenomena can also be observed in Figure 37 where more objectives make a larger number of dynamic helpers viable.

There are two general cases for how a solution could be residing at a SOG local optimum for a $2q$-2 ruggedness transformation and v value of 2. Both bits in the solution that differ from the SOG optimum could be exclusively in one SO or the bits could each be in a different SO. If the bits in a solution that differ from the SOG optimum are all in one SO, then using that SO as a helper could be highly effective because either of the one-bit moves that bring the solution closer to the SOG optimum will survive. Conversely using the helper that has neither of the two bits that differ from the SOG optimum will be ineffective since the optimum already exists for the SO that

has neither of the differing bits. When each of the bits in the SOG that differ from the optimal string are in different SOs, both of the SOs may be effective helpers if they can accept the one-bit move in their SO that moves the solution toward the optimal. This will allow the solution to be created sequentially from one-bit moves in the SO that was a helper. When both of these helpers are used concurrently, they should be even more effective as either bit move that improves the SOG will be accepted with the respective SO and thus the one-bit away solutions can be created concurrently. This leads us to the belief that helper sequence is important for certain landscapes where cooperating helpers can be identified and exploited.

Three cases were run using a linearly decreasing landscape in the SO layer. The linearly decreasing landscapes in the SO layer can be thought of as 'easy' portions of the SOG where the local optima are not present. Like all cases here, the main objective does not contain information directly in the SO layer. These cases vary only based on the type of helper sequence used in a run. In the random sequence case, helpers associated with any SO appear in a different randomly drawn order for each replication. This case assumes no information is used for determining a good helper sequence. In the best sequence, helpers appear first to last based upon their appearance in the genotype. This ensures that the two helpers associated with a SOG appear concurrently or as close together in time as possible. The worst sequence uses all of the helpers associated with the first SO in the SOGs before phasing in all helpers associated with the second SO in the SOGs. This worst case ensures that no two helpers associated with a SOG will be used concurrently. While this case would never be a goal in optimization, it may give us additional detail about the nature of TOP and helper sequences.

5.4.6.2 Results and analysis

Figure 39 also shows the results of three cases with linearly decreasing SOs. All of these three cases appear to have a natural convex-hull shape. These cases demonstrate a marked improvement over the base case indicating that if helpers with easy to find optima are present,

they can be used to overcome local optima in a more aggregate layer. In the 1 dynamic helper cases, the three linearly decreasing SO cases have an average best fitness that is statistically lower than the baseline case at the $\alpha = 0.05$ level. This demonstrates an intuitive result: if there is access to a helper that bypasses a local optimum in the main objective, it is advantageous to use that helper. In the cases with more than 1 dynamic helper, the random and worst sequencings performed nearly identically. The low probability that two helpers associated with the same SOG were randomly used together resulted in these two cases performing nearly identical. In contrast the best helper sequence had a significant improvement in average best fitness across these cases. Having both helpers associated with the SOG in use at the same time ensures the GA will survive any one bit improvement that moves the solution toward the SOG optimal. The overall dip in Figure 39 for the three linearly decreasing cases is explained by the fact that by increasing the number of dynamic helpers each helper has a longer lifetime and therefore can be more effective at guiding solution improvements that overcome the local optima in the main objective. This trend leads naturally to the conclusion that good helpers should be given more time in order to be effective.

5.4.6.3 Conclusion

Experiment 4 shows how different constraints or poor fitness areas in a problem can be overcome using multi-objectivization by rewarding solutions that improve fitness in a given area regardless of layered interactions that may be present. This evidence indicates that appropriate helpers should be defined at the appropriate level of a problem in order to break epistatic interactions. Furthermore, helper objective sequence matters - concurrent, complementary helpers assist in providing improved results.

5.5 Chapter Conclusion

General principles underlying multi-objectivization were discovered through experimentation on the TOP model introduced here. The TOP model is a derivative of the Weise

model that adds layered objectives so that a larger set of multi-objective and single-objective problems can be studied. The TOP is a general optimization problem – it can be examined using many different optimization methods, including evolutionary approaches. Several features in the TOP model make it useful in studying multi-objectivization techniques including layered objectives, and an improved objective-convolution transformation. The general principles underlying multi-objectivization provide new insight into multi-objectivization techniques and how such techniques provide improved results.

Helpers appear to assist primarily in overcoming parallel local optima as opposed to overcoming sequential local optima. The work here has proven that multi-objectivization can reduce the difficulty of overcoming local optima in at least two ways. Firstly, by using multi-objectivization, good solutions can be identified that the algorithm would have otherwise discarded. When these solutions become difficult to generate within problem sections, their improved objective values (signal) can be hard to distinguish from simultaneous negative objective value moves in other parts of the genotype (noise). Improved signal-to-noise-ratios (SNRs) enable multi-objectivization approaches to demonstrate improved results over traditional single-objective techniques. Secondly, multi-objectivization can be used to traverse regions of the search space that would have been otherwise difficult to traverse due to interactions within the problem caused by epistasis or constraints. This chapter shows that predictive measures for the amount of alignment between the decomposed objectives and good solutions in the main objective are useful for identifying helpers that can break epistasis.

From this chapter and previous works we can see there are at least several principles associated with helpers. While achieving some of these general principles in practice may be difficult due to imperfect or lacking information, these goals still hold. This chapter has demonstrated that fewer helpers appear to be best for problems with moderate or higher conflict

and that problems with low conflict may benefit from more helpers. The following principles should be considered when using helper-objectives. Helper-objectives:

- should be sequenced according to their contribution to fitness (importance) (Lochtefeld and Ciarallo, 2011b),
- should be selected by their likelihood to be unhampered by local optima that are present in the main objective,
- should be selected based on improving SNRs that are internal to the problem
- should be selected such that concurrent helpers have complementary properties, and
- should be selected for their alignment with the main objective.

The signal-to-noise issues in a problem are likely an inherent part of many sum-of-parts fitness functions when there are many parts. Selecting helpers based on SNRs implies finding problem divisions where significant improvements can be detected while noise can be isolated to other areas of the optimization.

The last principle, selecting helpers with alignment with the main objective, has not been demonstrated in this chapter but makes intuitive sense. Trade-offs in portions of a problem may actually require a poor fitness value for a helper in order to build good or optimal solutions. If a helper must get worse in fitness compared to fitness values currently represented in the population, the helper will be largely ineffective since helpers reward fitness improvements, not fitness decrements.

Additional study of multi-objectivization on the TOP may provide further insight into multi-objectivization. Because TOP results in symmetric landscapes, some concepts of multi-objectivization may be difficult to study in the problem model's current form. For instance an effective in-depth study of helper-sequencing would likely require an asymmetric objective-convolution transformation so that some objectives conflict heavily while others barely conflict at all. Asymmetry of transformations can be implemented relatively easily in TOP by removing the single parameter for a transformation and adding maximum and minimum parameter values that govern the transformation. Transformations could then be applied across the solution such that each subsequent portion of the solution would be transformed by using a parameter value that

would linearly increase from minimum to maximum parameter values depending on the location of the transformation in the solution. This would lend significant asymmetry to the TOP while maintaining relatively low complexity. Additionally, there are many other ruggedness landscapes that could be studied using multi-objectivization to determine how the optimization technique behaves with the addition of more local optima in an objective and with the addition of levels of deception. Ruggedness transformations that do not follow the suggestions set forth by Weise et al. (2008) may also prove interesting from a study perspective. Lastly, multi-objectivization in TOP can also be studied in the light of other effects not examined here such as additional neutrality and stochastic noise introduced by the overfitting and oversimplification process.

Even though additional research on abstract problems is needed, some of the research in this chapter may not directly and fully translate into conclusions about individual NP hard problems. As a result, the following two chapters study two NP hard problems. Firstly we study the TSP, and next we revisit the theoretical founding between two different decomposition methods and test theories on the JSSP.

Chapter 6 Multi-Objectivization via Segmentation on the TSP

6.1 Introduction

Multi-objectivization is a relatively new optimization technique due in part to two reasons. Firstly, Evolutionary Multi-objective Optimization EMO methods are relatively recent and prior to the introduction of EMO methods in the late 1980's few efficient techniques existed to simultaneously find many solutions on the Pareto frontier. Secondly, and perhaps just as importantly, the size of optimization problems studied has increased to a point where multi-objectivization methods can be competitive. Small problems are generally not complex enough to require multi-objectivization techniques. Some research to date has provided weak evidence that larger problems benefit from more aggressive forms of multi-objectivization (Jensen, 2004, Lochtefeld and Ciarallo, 2011b, Lochtefeld and Ciarallo, 2011c). Additional stronger evidence is required.

Multi-objectivization techniques fall into two major categories (Handl et al., 2008a). The addition of novel objectives is one major approach. Novel objectives approaches have shown improved results over single objective optimization with the addition of objectives such as solution age (Abbass and Deb, 2003), frame bar width (Greiner et al., 2007), and the first derivative of the objective function (Deb and Saha, 2010). The second major category of multi-objectivization is Multi-objectivization Via Decomposition (MVD). MVD divides the objective function into component objectives and then uses those objectives in the optimization process. MVD has been most commonly used on fitness functions that have a sum-of-parts property but has also been theorized as being useful in sum-of-product fitness functions (Jensen, 2004).

Previous works on MVD have used two major approaches. The first approach, helper-objectives, utilizes the main objective in conjunction with additional decomposed objectives (Jensen, 2004, Lochtefeld and Ciarallo, 2011b, Lochtefeld and Ciarallo, 2011c). The second approach, *pure decomposition*, does not use the original problem's main objective but instead only works on the decomposed objectives (Knowles et al., 2001,Jahne et al., 2009,Handl et al., 2008a,Handl et al., 2008b). Lochtefeld and Ciarallo (2011c) outlined several principles governing multi-objectivization via helper-objectives. Several of these principles are general and likely apply to multi-objectivization via pure decomposition. This chapter studies the general principles that govern helper-objectives using a pure-decomposition method in order to determine the applicability of the principles governing helper-objectives on pure-decomposition approaches. These principles are studied in the context of the Traveling Salesman Problem (TSP).

The remainder of this chapter is structured as follows. The background section focuses on multi-objectivization research to date. General principles of multi-objectivization are summarized. Further the TSP is described and prior research studying the TSP using multi-objectivization techniques is examined. The experiment section contains three distinct experiments. The first two experiments focus on finding and improving possible decompositions for the TSP by analysis of existing decompositions, proposed new decompositions, and empirical study of the performance of promising decompositions. The third experiment introduces MOPS and evaluates its performance against methods with a static degree of decomposition such as MOS. Finally, concluding remarks are provided that both summarize the work and recommend avenues for further research.

6.2 Background

The TSP is a classic combinatorial optimization problem that "is probably the most studied of $\mathcal{NP}$-hard problems" (Applegate et al., 2006). The goal of the optimization is to find good or optimal low-cost tours that traverse a set of cities. A tour is a route that starts and ends at the

same city and travels through each city exactly once. Frequently solutions for the TSP are defined by a permutation string which determines the sequence in which cities are visited. Each city appears in the string exactly once. TSPs have a multitude of practical applications and have been used in the past to model and solve problems related to applications in data clustering, drilling circuit boards, genome sequencing, and delivery and pickup (Applegate et al., 2006). Methods to solve large TSPs include heuristics such as Genetic Algorithms (Larrañaga et al., 1999), Simulated Annealing (SA) (Aarts et al., 2005), and tabu search (Knox, 1994), and exact methods such as branch and bound (Applegate et al., 2006) and dynamic programming.

The remainder of this background section is composed of two areas. The first section, *multi-objectivization studies on the TSP*, summarizes previous work accomplished on the TSP. The second section, *principles of multi-objectivization*, describes known principles that apply to multi-objectivization techniques. For a broader background on multi-objectivization the reader is referred to (Lochtefeld and Ciarallo, 2011b, Lochtefeld and Ciarallo, 2011c).

6.2.1 Multi-objectivization studies with the TSP

Multi-objectivization has been studied with the TSP in at least three independent efforts under the same thread of research. Knowles et al. (2001) examined the TSP using a multiple objective hill climbing algorithm. Later, Jensen (2004) applied the concepts of helper-objectives to the TSP. Finally, Jahne et al. (2009) studied the TSP and proposed a new method called Multi-Objectivization via Segmentation (MOS). These three works are discussed next.

Knowles et al. studied the TSP with multi-objectivization by pure decomposition using the Pareto-Envelope based Selection Algorithm (PESA) (Corne et al., 2000), a multi-objective hill climber. To turn the main objective into multiple objectives, two random cities, cities A and B, were selected. The decomposed objectives were the travel cost of moving from city A to city B, and the travel cost of moving from city B to city A. Since the full tour consists of going from city A, through some cities and on to city B, and then through some other cities and back to city A,

the decomposed objectives ensured all costs associated with a full tour were considered. The multi-objectivization method using PESA outperformed its single-objective counterpart on six different TSPs ranging from 20 to 100 cities in size.

The Knowles et al. approach suffered from three weaknesses. Firstly, decompositions could be degenerate if cities A and B were close to each other. Secondly, and exasperating the first weakness, only a single decomposition was used in a given run which made the run heavily based on a single problem division. Lastly, identical solutions could be incomparable in the Pareto sense if a tour were reversed in two or more different solutions (Jensen, 2004). Identical, incomparable solutions in the Pareto sense can result in inefficient tracking of solutions by an EMO algorithm.

Jensen (2004) corrected these weaknesses by explicitly assigning each city to two or more decomposed objectives. Jensen used the main objective simultaneously in conjunction with the decomposed objectives via a concept called helper-objectives. Because helper-objectives use the main objective simultaneously with the decomposed objectives, the best solution found would survive throughout the optimization. Since the objective function is based on the cost of travel between cities, each helper-objective summed the cost of incoming and outgoing links for its associated cities. This type of decomposition sums the cost of each link twice since links are shared between two cities. If two cities were adjacent in the tour and assigned to the same helper-objective, the cost of the links between the cities is added twice when calculating the objective value of the helper-objective. Similarly if adjacent cities are in different helper-objectives, the cost of the link is added to the objective value of both helper-objectives. Cities were randomly assigned to the different helper-objectives. To combat the possibility of a single, poorly-chosen decomposition, multiple random decompositions were used. These decompositions were used sequentially based on a random ordering. After a certain number of generations, a new set of helper-objectives would be used by the optimization. A more detailed description of helper-

objectives is provided in Lochtefeld and Ciarallo (2011b). Jensen studied 40 TSPs ranging from 99 to 2103 cities.

Jensen theorized that adaptive strategies using the decomposed objectives could give improved results. An adaptive strategy makes decisions about how the algorithm works based on the evolution of the population. Jahne et al. used adaptive decompositions of the cities based upon different properties of the costs of current links represented in the population (Jahne et al., 2009). The proposed MOS method uses pure decomposition, partitioning the cities into two decomposed objectives based upon a single dividing point. This dividing point is determined by examining a sample of individuals in the population to determine the representative cost of links associated with cities. Three different dividing points were considered. For instance, one decomposition used Expected Value Of Distances (EVOD) for each city to divide cities into two segments based upon above average and below average EVODs. If the represented cost of the links into and out of a city in the sample was greater than the average EVOD for all cities in sample, the city was assigned to the first segment. Conversely, if the represented cost was lower than the average EVOD in the sample, the city was assigned to the other segment. Decompositions that used more than two divisions were not studied because of empirical evidence gathered by Jensen (2004) on the Job Shop Scheduling Problem (JSSP) that indicated the smallest (most basic) decompositions were the most competitive.

EVOD uses the expected value of the distance scores represented in a sample. A given distance score for a city is defined by summing all of the represented distances into and out of that city for the individuals in the sample. Suppose we need to calculate the distance score for city α for a sample of the population defined by the set I that contains ρ individuals. Let the function $P(i, \alpha)$ return the city immediately preceding city α in solution i. Similarly, let the function $S(i, \alpha)$ return the city immediately following city α in solution i. Let the function $D(\alpha, \lambda)$ return the Euclidian distance between cities α and λ. An EVOD score for a city is then defined as

$$EVOD(I,\alpha) = \left\{ \sum_{i \in I} D\left(\alpha, P(i,\alpha)\right) + D\left(\alpha, S(i,\alpha)\right) \right\} / \rho \quad (3)$$

Jahne et al. (2009) studied MOS using three different adaptive problem divisions – the EVOD, the standard deviation of distances, and the expected value of different neighbors. Each of these methods uses a sample of the population to divide cities into two segments associated with a given decomposition. These three segmentation methods were compared against the 'traditional' GA and the previously mentioned methods proposed by Jensen and Knowles et al. MOS was found to be superior for the majority of the 12 TSPs studied. Among the three segmentation methods studied, the EVOD was the best method when no special operator was used to preserve additional diversity. After finding the scores for each city using (3), EVOD computes an average score for all cities. If an individual city has a score above the average score, the city is assigned to a first segment, otherwise the city is assigned to a second segment. The sample set of cities, I, was defined as all individuals in the population, e.g. the sample was exhaustive. A formal description of EVOD and more detailed descriptions of the other two segmentation methods are given in Jahne et al. (2009).

6.2.2 Principles of multi-objectivization

Several principles related to helper-objectives have been identified in the past (Lochtefeld and Ciarallo, 2011b, Lochtefeld and Ciarallo, 2011c). We theorize that the many of the principles associated with helper-objective optimization also apply to pure decomposition methods. These principles are as follows.

- Helper-objectives should be defined based upon their likelihood of avoiding local optima present in the main objective.
- Concurrent helper-objectives should have complementary properties.
- Helper-objectives should be selected for their alignment with the main objective
- Helper-objectives should be selected to improve Signal-to-Noise Ratios (SNRs) created by solution generation.
- Helper-objectives should be sequenced according to their contribution to fitness.

Defining helper objectives based upon their likelihood of avoiding local optima present in the main objective is likely the most difficult principle to apply to a real problem. Certainly knowing the place, size, and other properties that define each local optimum in a problem eliminates the need for a heuristic since the optimal value must be known. Some efforts have attempted to understand the problem landscape and problem difficulty through sampling techniques and the concept of epistasis (Davidor, 1991,Naudts et al., 1997,Reeves, 1999). These techniques must always assume some level of error. The principle of overcoming local optima is not explicitly focused on by this chapter since current sampling techniques are fraught with error unless large samples are used. Large samples of a problem landscape can be cost-prohibitive to obtain for large TSPs as the problem gets combinatorial more complex with the addition of cities.

The principle that concurrent objectives should have complementary properties has been focused on heavily by past research. For instance, picking cities A and B that were far apart attempted to give complementary and balanced objectives in Knowles et al. (2001) By dividing cities based on their expected incoming and outgoing link cost, Jahne et al. attempted to create a broad Pareto frontier where many Pareto efficient solutions were possible (Jahne et al., 2009). This chapter explores complementary decompositions only in the context of reducing SNRs.

Selecting decomposed objectives that are in alignment with the main objective is natural to most multi-objectivization approaches. For a minimization problem like the TSP and JSSP, it is important that the decomposed objectives be minimized (to some extent) or the multi-objectivization process will fail to reward solutions with good properties. Maintaining alignment with the main objective has been primarily accomplished by using relatively large decompositions because as decompositions get large, the likelihood that the decomposition must be minimized increases. Since the objective in these problems is to minimize a sum of parts, when more parts are used in a decomposition there is a higher probability the decomposition requires minimization in order to generate good solutions. However, as the problem gets larger by

adding more cities or operations, more decompositions should be viable because each part of the fitness function contributes to a smaller proportion of overall fitness and thus decompositions will remain relatively large even though there are more of them. These large decompositions will maintain alignment with the main objective due to their size. This implies that larger problems can benefit from finer objective divisions. Some weak evidence of this has been reported in Lochtefeld and Ciarallo (2011b) where the largest problems sometimes benefited from the use of more helper-objectives. Also some evidence to indicate that more aggressive multi-objectivization is useful on larger problems has been shown in Lochtefeld and Ciarallo (2011c) on a well-structured problem where all information about the problem landscape was known *a priori*. Understanding the driving forces in selecting an appropriate level of multi-objectivization may assist us in generating better algorithms.

Little work to date has explicitly focused on improving SNRs during solution generation as a principle for multi-objectivization. This SNR principle states that since evolutionary search works simultaneously on different parts of a solution, a potential improvement in the fitness of a part of a solution (signal) may be wrongly ignored if the improvement was created simultaneously with some fitness decrement (noise) in another part of the solution. We theorize that for algorithms that employ recombination, a strategy that isolates signal from noise can enable the multi-objectivization process to work more efficiently.

The last principle is that helper-objectives should be used in an order that is based on fitness importance. This principle was shown in two works on the JSSP with overwhelmingly better results than the previous random ordering (Lochtefeld and Ciarallo, 2010,Lochtefeld and Ciarallo, 2011c). This principle applies to single decompositions with a large number of splits where the splits are used sequentially over time.

In addition to the principles of helper-objectives for pure decomposition, two other more general principles apply. Decompositions should divide all components of the main objective and

decompositions should be non-degenerate. Dividing all components of the main objective ensures all component parts are considered in equal weighting to how they are considered in the main objective. For example, in the MOS for the TSP, each city should be assigned to exactly one objective. If several cities were excluded in the optimization or some cities appeared more than once, the decomposition fails to reflect the properties of the original problem. Decompositions should be non-degenerate to ensure that the algorithm does not spend an inordinate amount of time handling the multi-objective (Pareto ranking) portions of the search on objectives that have zero or few possible trade-offs. For instance, in the case of the TSP a degenerate decomposition is one with no assigned cities. Arguably, a decomposition with only a few cities may also be minimally productive as the number of possible Pareto efficient solutions can be small. One method of minimizing the creation of degenerate decompositions is to divide the objective components into evenly sized segments.

This chapter focuses on attempting to create segmentations based on the SNR principle using equally sized segments. We also introduce an extension to MOS that provides a phased approach to multi-objectivization.

6.3 Experiments

The goals of the experiments are to use and extend MOS to:

- determine if objective decompositions inspired by the use of signal-to-noise are competitive with EVOD,
- identify the mechanism that makes EVOD a competitive division and how well EVOD extends into decompositions with a larger number of splits,
- identify other decompositions that may work better than previous decompositions
- determine how levels of multi-objectivization change convergence, and
- evaluate a new algorithm that utilizes progressively stronger decompositions

To achieve these goals, TSP instances examined range from 107 to 395 cities. Data for the TSPs in the experiments were obtained from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ and http://www.tsp.gatech.edu/vlsi/index.html. Experiments 1 and 2 focus heavily on developing methods based on the SNR principle. Experiment 3 introduces

a variant of MOS which combines the strengths of different decomposition levels into a single run. The experiment also evaluates the performance of the new algorithm relative to MOS.

6.3.1 Algorithms, settings, and operators

Non-dominated Sorting Genetic Algorithm version II (NSGA-II) (Deb et al., 2002) was used to perform the multi-objective portions of the search. This algorithm was used by both Jahne et al. (2009) and Jensen (2004). NSGA-II uses Pareto dominance as the first order of fitness evaluation and a crowding mechanism to differentiate between Pareto incomparable solutions. The crowded comparison method rewards diversity along the Pareto frontier. One weakness of a pure decomposition method such as MOS is that there is a chance that the best found solution will not always survive if the non-dominated front size of all candidate solutions becomes larger than the population size. To address this weakness, the MOS implementation here archives the best found solution at the end of every generation.

Similar to the previous two works on multi-objectivization for the TSP (Jensen, 2004, Jahne et al., 2009), this chapter uses the improved edge recombination operator, tournament selection, single city mutation, and the 2-opt local search procedure. A solution is encoded as a list of integers representing the different cities. Adjacent cities in the genotype are adjacent in the tour with the final city also being adjacent to the initial city. Parent selection is accomplished by fitness tournaments with a tournament size of two. The mutation operator is performed only when recombination is not performed. Mutation creates a new individual by creating a copy of a single parent, removing a city from the genotype, and inserting that city at a random position in the genotype. The algorithm does not consider infeasible solutions; feasibility is maintained by the mutation, recombination, and local search procedures.

The improved edge recombination attempts to maintain edges that existed in the parents rather than focus on the order that cities appear in the genotype. The improved edge recombination operator creates an edge table that indicates which cities are adjacent to a given

city and also flags edges shared by both parents. Edges are selected from the edge table using a greedy selection strategy building a solution one edge at a time. The edge selection strategy attempts to preserve common subtours while also minimizing dead-end assignments. Whitley et al. (1991) provides a more detailed description of the operator.

After a new individual is created either through recombination or mutation, the individual is improved using 2-opt. 2-opt is a powerful, but somewhat computationally prohibitive local search heuristic. Naive implementations of 2-opt are $O(N^2)$ where N is the number of cities in the TSP (Codenotti et al., 1993). Methods that sort on the distance of candidate improvement cities typically have improved performance. The distance of improvement candidates are sorted into a list for each city, L_c. The 2-opt algorithm in these experiments only considers the first 25% percent of cities in each L_c. Exploring a limited number of cities for improvement can increase computational performance without a compromise on solution quality since the probability of a far city contributing to a local fitness improvement is low. This implementation of 2-opt is consistent with experiments in Jahne et al. (2009).

Many parameter settings were consistent with the analysis in Jensen (2004) and are also consistent with the experiments in Jahne et al. (2009). Jensen ran experiments to determine the good settings for population size, recombination rate, and the number of objective decompositions (Jensen, 2004). Like both previous works, a population size of 100, recombination rate of 0.7, and 10 objective decompositions are used here. The swapping of various decompositions was accomplished according to the swapping of helper-objectives for the TSP experiments in Jensen (2004). For the adaptive segmentation methods here, like the prior work by Jahne et al. (2009), these experiments used the entire current population as the sample for determining segmentation.

Relative run times between the methods tested were very similar. The time required to generate and swap segments and to perform multi-objective survival selection is vastly dominated

by the time required to create new solutions through edge recombination and 2-opt. The fact that solution generation dominates the time required to complete a run implies that using the number of fitness evaluations as a computational budget is appropriate for direct comparisons of different MOS methods.

6.3.1.1 Decompositions supporting signal-to-noise isolation

Recombination methods for the TSP generally attempt to maintain some properties of the tours associated with the parents. Since the properties of a tour contain relatively local information, good recombination methods generally attempt to keep local neighborhoods in tact while generating a new solution. If we can isolate a neighborhood from the remainder of a solution, it is possible that we can identify improvements in that neighborhood (signal) regardless of other simultaneously poor moves made elsewhere in the solution (noise).

In MOS, a given decomposed objective had an assigned set of cities and the decomposed objective was called a segment. For MOS, only two segments were used in a given decomposition, but it is easy to extend the MOS concept for more than two segments. To do so, we define a segment tuple as the set of segments associated with a given decomposition. Since a given decomposition may be subject to bias, an optimization will typically create multiple segment tuples for a given replication. Let K be the number of segments in a given segment tuple and S be the number of segment tuples. We refer to K as the degrees of multi-objectivization and S the number of decompositions. K and S must follow the constraints $K \geq 2$ and $S \geq 1$ to be a MOS approach. If $K = 1$ and all cities are included in the decomposition, the optimization is essentially a 'traditional' single-objective optimization. Similar to MOS, the decompositions presented here assign a number of cities to different segments. Fitness for a given segment is then the sum of all incoming and all outgoing links for all cities in a given segment. Since a given link is shared between two cities, every link is counted twice when evaluating the fitness of all

segments. This ensures an unbiased and yet all inclusive decomposition of the problem. (Jensen, 2004) provides additional motivation regarding this type of decomposition.

6.3.1.2 Radial decomposition

We would like to improve SNRs by isolating local neighborhoods from other neighborhoods of the problem. This may be accomplished by dividing the problem into multiple spatially oriented segments. There are many possible spatially oriented segmentation methods. The goals of this spatial decomposition are to create splits that minimize the division of important neighborhoods, create many possible sets of objectives for the optimization, create relatively balanced divisions of the main objective, and include each city in exactly one segment in each decomposition. An unbiased method that selects segments based on balanced divisions of the problem can delay and/or prevent the production of degenerate segments as S increases. A scalable method allows us to examine more degrees of decomposition since there is some evidence that in some cases problems can benefit from more aggressive decomposition. Finally, a method that produces no disjoint areas attempts to keep as many neighborhoods intact as possible. One possible decomposition method that meets these requirements is a method inspired from the idea of dividing the problem into 'wedges of a pie'.

Isolating signal from noise may be as easy as generating spatially sensitive decompositions by creating 'wedges' originating at a central point of the data that would divide the cities into wedges. For the most basic decomposition, drawing a line through the centroid of the city locations provides a way to assign cities into two objective sets. Such decompositions allow fitness improvements in one half of the cities to be captured regardless of simultaneous fitness decrements that occur in the other half of the cities. Dividing the problem spatially creates decompositions with a relatively even number of cities if the cities are somewhat spatially symmetric about the centroid of the data. For larger degrees of decomposition beyond 2, rays radiating from the centroid can be used to define the 'wedges of the pie'.

A segment can be defined as all cities in a wedge formed between two rays originating at the centroid that are $\Delta\theta_K = 2\pi/K$ radians apart. We would like to create *S* segment tuples. Each segment should be unique throughout all segments in all segment tuples. Thus there should be a total of *KS* unique segments. Since multiple unique tuples are desired, the starting angle for determining these rays must change as the segment tuples are created. The starting angle of the initial ray that partially defined the first segment must change by $\Delta\theta_S = 2\pi/KS$ radians for each tuple for the tuples to be unique. To prevent the algorithm from having a bias due to the initial starting angle, in each replication we create a new random starting angle between 0 and 2π radians defined as θ_{start}. The angle for a given ray is then defined as $\theta_{ij} = \theta_{start} + j\Delta\theta_K + i\,\Delta\theta_S$ where *i* is an index for the number of segment tuple and *j* is an index for the number of segment within a given tuple. By rotating the cities around the centroid point by the angles that define the ray, we can determine the cities between the rays θ_{ij} and θ_{ij+1}. A city g, at location (x_g, y_g), belongs to segment T_{ij} iff $(y_g - \bar{y})\cos(\theta_{ij}) + (x_g - \bar{x})\sin(\theta_{ij}) \geq 0$ and $(y_g - \bar{y})\cos(\theta_{ij+1}) + (x_g - \bar{x})\sin(\theta_{ij+1}) < 0$. Segment tuples defined in this way are as far apart from each other as possible and non-overlapping. To prevent potential rounding errors, *K*-1 segments were created using this method and the remainder segment was assigned those cities that had not been assigned a segment. Figure 40 shows a division of a notional TSP for *K* =3 for the i^{th} tuple.

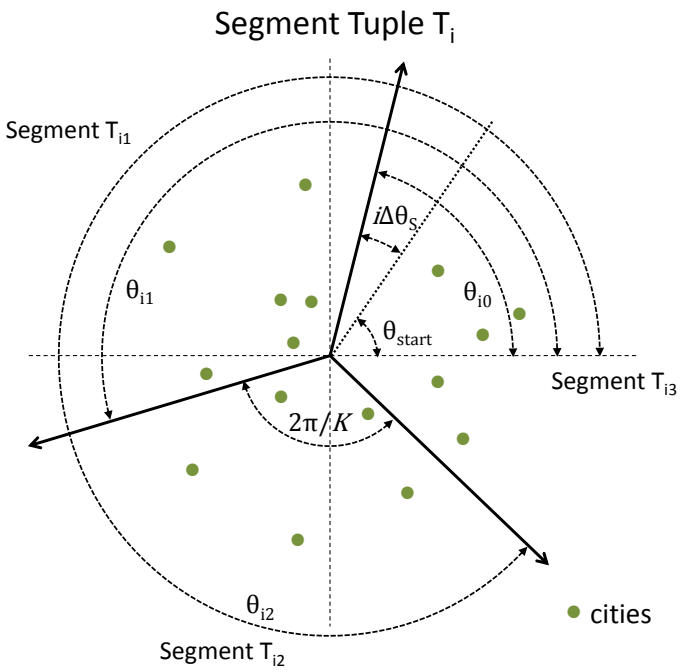


Figure 40 – Radial segmentation of cities in a TSP

Previous work established that the sequencing of decomposed objectives is important (Lochtefeld and Ciarallo, 2011b). The segment tuples generated by the radial segmentation method can be sequenced in different orders during the run. Two sequences were considered. The first sequence places the segments in the order $i = \{1,2,3,\ldots,S\}$ which generates a gradual change in the decomposed objectives as few cities will switch to different segments when a new segment tuple is used. This radial decomposition using the gradually changing sequence is further labeled as 'radial ordered' for the experiments here. The second sequence considered places the segments in the order $i = \{1, \frac{S}{2}, 2, \frac{S+2}{2}, 3, \frac{S+4}{2}, 4, \ldots, \frac{S}{2} - 1, \frac{S+S}{2}\}$. This decomposition and sequencing, labeled further as 'radial phased', swaps in segments that are as different from the currently used segments as possible. Since previous multi-objectivization works on the TSP has indicated that maintaining diversity in the TSP is important (Jensen, 2004, Jahne et al., 2009), this method may give better results than radial ordered sequencing.

6.3.1.3 k-means decomposition

Radial based decompositions have weaknesses. Neighborhoods on the edges of the rays will become divided. Neighborhoods near the centroid of the cities, will be divided frequently while neighborhoods far from the centroid will be divided less frequently. If keeping neighborhoods intact is critical to these segmentation methods, a k-means clustering algorithm that assigns the cities into exactly K segments may give better results than the radial segmentation methods. Since k-means clustering can give different outcomes depending on the seed conditions, it has the potential to still generate many unique segment tuples.

A simple clustering technique such as Lloyd's algorithm (Lloyd, 1982) was used here. Lloyd's algorithm iteratively assigns cities to clusters based upon their distance to K centroids. Cities are assigned to the cluster associated with the closest centroid. This process is repeated until the centroids of the clusters do not change. For the work here, the initial clusters were assigned using the radial methods outlined above and thus we examine two k-means methods. 'k-means ordered' uses the radial ordered segments for their initial seeded clusters. 'k-means phased' uses the radial phased segments for their initial clusters.

6.3.2 Experiment 1

The first experiment was performed on the set of problem instances pr107, pr124, pr152, pr299, rat195, d198, xqf131, xqg237, pma343, pka379, bcl380, and pbl395. This set of problem instances corresponds to all problem instances with reported output in Jensen (2004) with less than 400 cities. Problem instances pr107, pr124, pr152, and xqf131 were also studied in Jahne et al. (2009).The six instances xqf131, xqg237, pma343, pka379, bcl380, and pbl395 are Very Large-Scale Integration (VLSI) instances while the other seven are non-VLSI instances. The maximum number of fitness evaluations performed, E, scaled according to $E = m\sqrt{N^3}$. (Jahne et al., 2009) used this formula for their computational budget with $m = 15$. m was set to a value of 5 for this experiment and therefore we are using exactly one-third of the number of fitness

evaluations examined by the previous MOS study. 200 replications were run for each experimental setting.

The concept behind EVOD is to divide the cities into a high-cost and a low-cost segment based on their represented costs in the population. The EVOD method cannot assign cities to more than two segments since it is based on a single dividing point (the average cost of links in a sample). To extend EVOD into higher dimensions a more general form is required. The extended version of EVOD is further called the Percentile of Distances (POD). POD can produce segmentations for $K \geq 2$. Like EVOD, the costs associated with a given city are calculated by summing the incoming and outgoing links for that city for each individual in the sample. In POD, these cities are sorted from highest to lowest cost. This list of cites is then divided by percentiles based on the desired number of segments: all cities associated with a given percentile are assigned to the same segment. For example if the number of desired segments is 4, the sum of the costs of represented links for each city are calculated and the cities associated with the first quartile of costs is assigned to the first segment, the cities associated with the second quartile of costs is assigned to the second segment, and so forth. For 3 segments per segment tuple, this method would create one segment containing the high cost cities, one segment containing the medium cost cities and one segment would containing the low cost cities.

Table 16 details the overall results from Experiment 1 as reported by the average percentage from the optimal value for the 200 replications. A value of 0.000 corresponds to all 200 replications finding an optimal solution. For EVOD, equal or marginally greater average percentages from optimal values were found for the problem instances that overlap reported results in Jahne et al. (2009). Considering the number of evaluations used in this experiment was one-third of what was previously used, these marginally worse to equivalent results indicate that MOS was implemented in a consistent manner with the prior work.

Table 16 – Experiment 1 - Average percentage from the optimal solutionfor various segmentation types degrees of decomposition

	***K* = 2**					***K* = 3**				
	EVOD	**Radial Ordered**	**Radial Phased**	**K-means Ordered**	**K-Means Phased**	**POD**	**Radial Ordered**	**Radial Phased**	**K-means Ordered**	**K-Means Phased**
pr107	0.0035*	0.0024*	*0.0022**	0.0026	0.0028	0.0368*	0.0008+	0.0011+	0.0013+	***0.0003+***
pr124	0.0008	***0.0000***	0.0005	0.0008	0.0009	***0.0000***	0.0177*-	0.0237*-	0.0215*-	0.0082*-
pr136	0.1359*	0.1418*	***0.1169+***	0.1192+	0.1269*	0.1816*	0.1359*+	0.1323*+	0.1400*+	*0.1217+*
pr152	0.0106	0.0062	***0.0060***	0.0193*	0.0254*-	0.0321*	0.0493*-	0.0283*	0.0129+	*0.0074+*
pr299	0.1569	0.1591	***0.1443***	0.1809*-	0.1907*-	0.2611*	0.2134*+	*0.1685*+*	0.2270*+	0.2090*+
rat195	0.5415	0.5466	***0.5155***	0.5877*-	0.5566*	0.5780*	0.5691*	*0.5293+*	0.5360+	0.5596*
d198	0.1937*	0.1736+	***0.1637+***	0.1997*	0.1923*	0.3012*	0.3456*-	*0.2907**	0.4459*-	0.4414*-
xqf131	***0.0258***	0.0445*-	0.0437*-	0.0552*-	0.0481*-	0.0294	*0.0276*	0.0303	0.0347	0.0303
xqg237	*0.3215*	0.3230	0.3299	0.3467*	0.3432*	0.4586*	***0.2988+***	0.3131+	0.3319+	0.3013+
pma343	0.1745	***0.1532***	0.1543	0.1800*	0.1686	0.4324*	0.2186*+	*0.1682+*	0.1929*+	0.1899*+
pka379	0.1784	0.1864	***0.1698***	0.1837	0.1830	0.5150*	0.1973*+	0.2003*+	0.2230*+	*0.1954*+*
bcl380	***0.1293***	0.2719*-	0.2328*-	0.2517*-	0.2558*-	*0.1575*	0.1600	0.1624	0.1618*	0.1668*
pbl395	*0.2040*	0.2624*	0.2848*-	0.2954*-	0.2612*	0.2467*	0.1804+	***0.1718+***	0.1722+	0.2020
Average of VLSI TSPs	***0.1723***	0.2069	0.2025	0.2188	0.2100	0.3066	0.1805	*0.1744*	0.1861	0.1810
Average of Other TSPs	0.1490	0.1471	***0.1356***	0.1586	0.1565	0.1987	0.1903	*0.1677*	0.1978	0.1925
Average of All TSPs	***0.1597***	0.1747	0.1665	0.1864	0.1812	0.2485	0.1857	*0.1708*	0.1924	0.1872

Values that are **bold** indicate the best division for all cases on a given problem instance. Values marked with a '*' are statistically worse than the overall best division of the problem instance. Values in *italics* indicate the best value for a given problem instance for a given value of *K*. For cases with *K* = 2, values identified with a '+' have a statistically lower average percent from optimal value than the EVOD case indicating the case outperformed EVOD. Conversely, for K = 2, values marked with a '-' have a statistically greater average percent from the optimal value than the EVOD case. For cases with *K* = 3, values identified with a '+' have a statistically lower average percent from optimal value than the POD case indicating the case outperformed POD. Conversely, for K = 3, values marked with a '-' have a statistically greater average percent from the optimal value than the POD case. All statistical tests used the student's t-test with an $\alpha = 0.05$ level assuming unequal variances.

From Table 16, two segmentation methods, EVOD and radial phased segmentation with $K = 2$, have the best resulting performances. Radial phased segmentation with $K = 2$ had the best performance across all problem instances and settings in 6 of the 13 problem instances while EVOD was the best performer in just 2 of 13 problem instances. However, for average performance across all the problem instances, EVOD outperformed radial phased segmentation. Relative to EVOD, the radial phased method with $K = 2$ performed worst in the bcl380 and pbl395 problems. Averages across all VLSI problem instances indicate that EVOD had better performance for these instances. Compared with EVOD, radial phased segmentation with $K = 2$ had better performance on the non-VLSI problem instances.

For the $K = 3$ cases, averaged across all problem instances, POD is the poorest performer relative to the other segmentations methods. This indicates that EVOD does not likely extend well into higher degrees of decomposition. The $K = 3$ cases generally underperformed the $K = 2$ cases. While some evidence in other problems has indicated that as a problem gets larger, it will be more amenable to more aggressive decompositions, only the bcl380 and pbl395 instances seem to support that theory here. It is more likely that the budget as it relates to problem size is the issue that has caused some past larger problems to be more amenable to higher degrees of multi-objectivization. More intuition on this topic is presented in Experiment 3.

The concept of sequencing different problem decompositions to change the diversity properties of an EMO has been introduced here. Such diversity control through ordering the decompositions has not been done in prior MVD works. In both the radial and k-means cases, the phased sequencing of segment tuples generally outperformed the ordered sequencing implying that segmentation sequencing with stronger diversity give better results for these instances. This is not altogether surprising given that previous works used additional operators to attempt to preserve diversity (Jensen, 2004, Jahne et al., 2009).

From the signal-to-noise argument outlined above, we conjectured that the differentiation between neighborhoods can result in improved performance. A visual inspection of the nature of neighborhoods in these problem instances gives us some explanation for performance of the different segmentation methods as it coincides with the input set. Insight into algorithmic performance can assist us in developing even better methods and can assist us in selecting the appropriate segmentation methods when studying other TSP instances.

6.3.2.1 Understanding the mechanism behind EVOD's and POD's performance

Since 2-opt is used on each solution, we can assume that local neighborhoods of cities will have relatively efficient paths represented in the population. In a population with efficient neighborhoods, many of the high cost links are likely the links between neighborhoods rather than the links within neighborhoods. After all, the spatial definition of neighborhood implies cities are close together relative to other cities. Logically from this argument, EVOD has a good chance of separating inter-neighborhood links from intra-neighborhood links since cities with above average cost links are assigned to one segment and the other cities are assigned to the remaining segment. Since methods like EVOD and POD are adaptive, as the efficient neighborhoods in the population change, the segmentation method will reassign some cities to different segments.

From Table 16, we can see that POD gave the worst average performance across all methods and values of K. This implies that EVOD does not generally extend well into decompositions with higher dimensionality because the additional decomposition does not maintain local neighborhoods well. However, for the pr124 problem, POD was tied as a top performer, finding the optimal value in all 200 replications. This evidence seems counter to the idea that EVOD will not extend well in higher dimensions. The spatial layout of cities for pr124 input data shown in Figure 41 reveals an explanation for why POD is competitive here. As indicated in the figure,

many of the neighborhoods in the pr124 contain uniform-cost efficient paths. There are a variety of low-cost efficient path neighborhoods and medium-cost efficient path neighborhoods. Since the efficient costs are uniform within neighborhoods and there are a variety of neighborhoods with different costs between cities, POD is able to differentiate between the low- and uniform-cost efficient path neighborhoods, the intermediate- and uniform-cost efficient path neighborhoods, and the cities that are either between neighborhoods or in high-cost neighborhoods. Other problem instances in the test set did not contain a similar distribution of uniform-cost efficient path neighborhoods, which is why POD had poor performance on other instances.

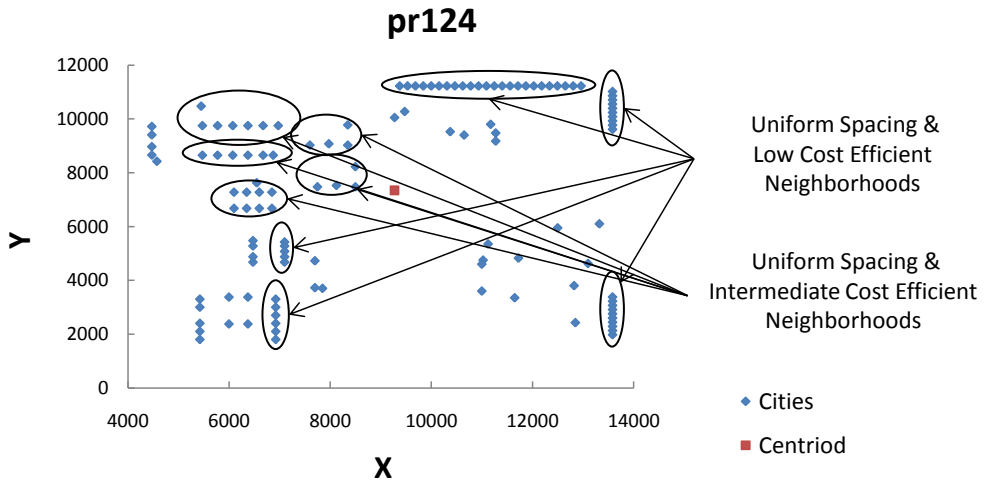


Figure 41 – City configuration for the pr124 instance

In the pr124 instance, some of the less dense neighborhoods are likely interpreted as not being in a neighborhood at all. These less-dense neighborhoods may have above average costs in distance in comparison to nearest neighbors in other areas of the problem and as a result they fall into the high-cost segment. While assigning a whole neighborhood to the intra-neighborhood segment may seem bad, it keeps the neighborhood intact. A potential weakness of EVOD and POD arises when problems have neighborhoods of various densities. In this case, EVOD and POD may consider cities in the sparse neighborhoods as between neighborhoods rather than

within. Neighborhoods with variable densities may have their high-cost cities assigned to a different segment than the other cities in the neighborhood. EVOD and POD both have a high potential to split many variable density neighborhoods into different segments. A decomposition method that uses some local neighborhood information to estimate densities of the neighborhood may be superior to the basic EVOD and POD methods because it may avoid some splitting of variable density neighborhoods.

EVOD differentiates between those cities within a neighborhood and those between neighborhoods. This differentiation allows the algorithm to identify and survive good solutions. In contrast, the radial segmentation and k-means segmentation methods divide the objective space based on the spatial location of cities. Spatially oriented segmentation methods do not explicitly separate cities that border neighborhoods from cities that are contained within neighborhoods but rather attempt to divide neighborhoods from each other. While the methods decompose the objective in very different ways, both spatial decompositions and intra/inter neighborhood decompositions attempt to improve the SNR of a candidate solution by isolating fitness decrements (noise) from fitness improvements in a local area (signal).

6.3.2.2 Understanding radial segmentation methods

Radial segmentation is based on the concept that neighborhoods are local and therefore dividing a problem into different localities should result in isolating a subset of neighborhoods from each other. The method will divide some neighborhoods between different segments. We theorize that the more the spatial segmentation method splits-up neighborhoods, the less likely it will have good performance. One problem where the radial segmentation methods were significantly outperformed by EVOD was in the bcl380 instance.

Figure 42 shows the city layout for bcl380. This instance appears to have many neighborhoods that have strong vertical orientations. Having many neighborhoods with vertical orientations is problematic for radial segmentation as radial segmentation will frequently divide

these neighborhoods when it creates segment tuples. This is reflected in the relatively poor performance of the radial segmentation methods in comparison to EVOD's performance on bcl380 as seen in Table 16. TSPs often have neighborhoods with a strong unidirectional orientation due to the nature of chip design and layout. Neighborhoods with a primary orientation that is not the same as the orientation of the segmentation method can cause many neighborhoods to be divided. Radial segmentation methods generally performed better on the non-VLSI problems. VLSI problems may be better suited for a different type of segmentation that better avoids dividing strongly vertical neighborhoods.

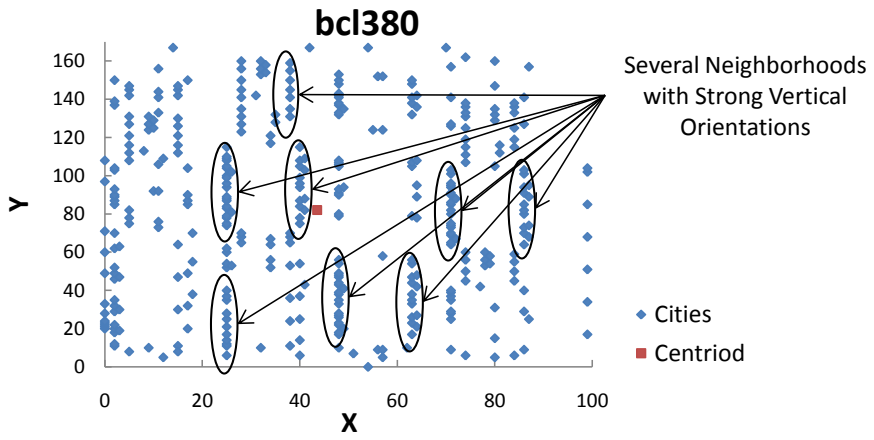


Figure 42 – City configuration for the bcl380 instance

6.3.2.3 Understanding k-means decompositions

The k-means methods examined in this experiment were used to attempt to repair some of the neighborhood divisions that may have been created by radial segmentation. The k-means methods on average performed worse than their radial segmentation counterparts. Even though Lloyd's algorithm has a low probability of splitting neighborhoods, it must have other properties that are causing worse performance than the radial methods. The k-means methods are believed to suffer from two major flaws. Firstly, the clusters are not required to maintain some balance of cities between clusters and thus one cluster may degenerate into only a few cities. We believe this is unlikely the cause given that many of problem instances here did not have extremely spatially

skewed distributions of cities. The second flaw of k-means clustering is that the clustering method may converge to the same solution regardless of the initial seeded conditions. For the instances studied here, k-means generates relatively poor performance when compared to the other methods.

The pr107 instance shows an example of where Lloyd's algorithm does not work well for the $K = 2$ cases. Examining the x dimension in pr107 we can see that cities are distributed in a bi-modal fashion with little middle-ground. Since Lloyd's algorithm is iterative, assigning cities to the nearest centroid until the new centroid has not changed, there is a high probability that all segments in the pr107 problem will be divided by the 'void' between the cities near $x = 8{,}500$ and the cities near $x = 16{,}000$. For pr107 Lloyd's algorithm does not allow MOS to explore different decompositions. In the $K = 3$ cases, Lloyd's algorithm must divide one side of the pr107 into two segments. In conjunction with the radial phased seeding approach, the k-means algorithm outperforms the other methods for pr107. Seeding the clusters with the radial phased approach alternates which side of the pr107 problem is divided. Since each side has a similar number of cities, this results in one segment that contains roughly half the number of cities and two other segments that contain the other cities. For $K = 3$ and radial phased seeding, the k-means phased method gives relatively 'even' splits of the cities, no splitting of neighborhoods, and alternating splits that result in stronger diversity. As a result, k-means segmentation for the $K = 3$ case was the top performer for the pr107 instance.

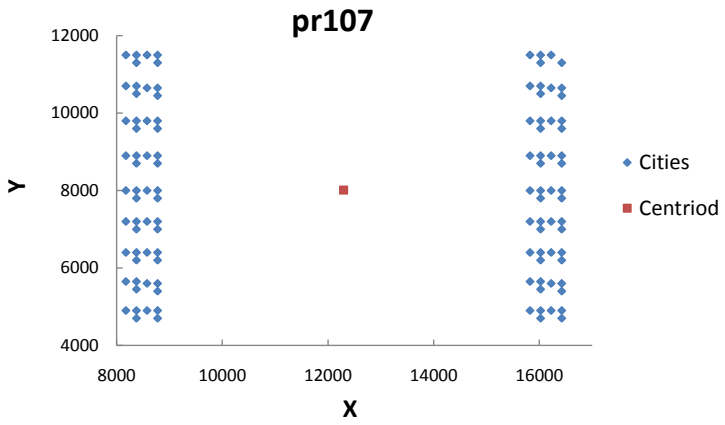


Figure 43 – City configuration for the pr107 instance

6.3.3 Experiment 2

We have seen in Experiment 1 that both inter- and intra-neighborhood decompositions (EVOD segmentation) and spatially oriented decomposition (radial segmentation) can result in improved performance. However, radial segmentation performs poorly on problems with city orientations that conflict with the segmentation method. Furthermore, inter- and intra-neighborhood methods do not generally extend well into higher dimensions.

One spatially oriented method and one inter- and intra-neighborhood method were considered in Experiment 1. Experiment 2 introduces a second spatially oriented method and a second inter- and intra-neighborhood method in order to confirm hypotheses drawn in Experiment 1. These new methods are compared with the prior methods to determine their suitability. Additionally, hybrid methods that include both inter- and intra-neighborhood and spatial decompositions are also introduced and analyzed. Other than the segmentation specific methodologies, Experiment 2 used algorithms and parameter settings consistent with Experiment 1.

The goals of Experiment 2 are: 1. to confirm that using an appropriate spatial segmentation method for the type of problem produces improved performance and 2. to determine when and if hybrid methods are better than existing segmentation methods.

6.3.3.1 Nearest Neighbor Ratio of Distances (NNROD) decomposition

We theorize that EVOD works well because it divides cities that are within neighborhoods from cities that are between neighborhoods. However, if neighborhoods are of various densities EVOD seems to misidentify cities within low density neighborhoods as being between neighborhoods. A simple and direct method of estimating the density of a given neighborhood may help to more precisely differentiate those cities that are actually between neighborhoods from those cities that are contained within neighborhoods. This is the inspiration behind the NNROD method.

Like EVOD, the NNROD method uses the costs of links represented in the population to determine which cities should be considered for the different segments. However, unlike EVOD which only uses the information in the population to determine the segments, NNROD also uses the information of relative city locations as defined by the input data. NNROD uses an estimate of the density of the nearby neighborhood to determine if a city is "between" or "within" the neighborhood. The parameter d is used in the process to estimate the density of a given neighborhood. As in EVOD, NNROD uses the average cost of links associated with a city by averaging the cost of incoming and outgoing links for the city in each solution in the population. In NNROD this average cost is then divided by the average of the distance from the city of interest to the d closest cities as defined by the instance data. The result of the division is an estimate of how often individuals in the population connect that city with its closest neighbors.

More formally, the NNROD score for city α and population sample I is calculated as follows.

$$NNROD(I,\alpha) = \frac{\{\sum_{i\in I} D(\alpha, P(i,\alpha)) + D(\alpha, S(i,\alpha))\}/\rho}{\{\sum_{j=1}^{d} D(\alpha, W_j)\}/d} \quad (4)$$

Define the sorted list W to contain all cities except city α. The cities in W are sorted in ascending order according to the distance cities are from city α. If the NNROD score defined by

(4) is relatively large, then city α is likely between neighborhoods in many of the population solutions because the cost of links to and from the city in the sample will be greater than the average cost of all links to the d nearest cities. Conversely, if the NNROD score is small, then city α is likely within a neighborhood for many of the solutions.

Similar to EVOD, NNROD does not extend well into decompositions with a larger number of objective splits since the method attempts to differentiate between cities that are within and between neighborhoods (a binary classification). Like POD, for more than two objective splits, NNROD uses percentiles to divide the cities among the segments such that an approximately equal number of cities are assigned to each segment.

NNROD adds a parameter to the optimization for estimating the density of neighborhoods. Since d supports the estimate of the density of a neighborhood, the value of d should be based on the size of neighborhoods. There is no good way to know the optimal value of d *a priori* as there are many definitions of a neighborhood that likely apply in a given optimization. The optimal value of d may also change over the course of the run. We assign a value of d in relation to K based upon the following logic: The segmentation methods here roughly assign the same number of cities to each segment to prevent the creation of degenerate segments. For NNROD, like in EVOD, we propose that one segment generally holds all of the links between neighborhoods while the other segments contain the links within neighborhoods; thus we pick an estimate of the average number of cities per neighborhood based upon the number of segments per segment tuple. Since each city in the TSP is only visited once, for every neighborhood there will be a city that starts the tour of the neighborhood and a second city that ends the tour. We consider these two cities the intra- neighborhood cities while the other cities in the neighborhood are the inter-neighborhood cities. Since we restrict attention to equal sized segments in order to prevent decompositions from becoming ineffective, we make an implicit assumption about the size of neighborhoods. Knowing the proportions of intra-neighborhood cities and inter-neighborhood

cities allows us to estimate the size of the neighborhoods since, in good solutions, one city typically begins and a second city ends the tour of the neighborhood. For POD when there are two segments per decomposition (K = 2 cases), one segment holds the inter-neighborhood cities while the other holds an equal number of intra-neighborhood cities. This implies for every 2 cities between neighborhoods, there must be 2 within the neighborhood for a total of 4 cities in an average neighborhood and an associated value of 3 for d. For the K = 3 decompositions, to maintain equal sized segments there must be 2 intra-neighborhood cities for every 4 inter-neighborhood cities for an implied value of 5 for d. More generally, if we assume all segments have a roughly equal number of cities and that only one segment contains those cites between neighborhoods, d must equal 2K-1. d was set to 2K-1 for this experiment.

6.3.3.2 Vertical decomposition

Radial decomposition's performance is not as good on VLSI applications where a large number of neighborhoods have strong vertical orientations. Furthermore, the relatively good performance of EVOD indicates that consolidation of neighborhoods that are far from each other into a single segment can be a competitive approach. This implies that decompositions with a small number of spatially disjointed segments may be worth examination. Since many neighborhoods in VLSI applications have a generally vertical orientation, a decomposition method that divides the cities into vertical sections may give better performance than the radial decompositions. A vertical decomposition on these problems has potential to cause less splitting of neighborhoods between decompositions and thus give improved SNRs.

In vertical segmentation, K vertical lines are used to define K segments where each segment contains all cities between a pair of lines. Similar to the case in radial segmentation, we would like to create S segment tuples and each tuple should be unique, so we need to create KS unique segments. This implies that between segment tuples the starting position for the first vertical line must change by $\Delta x_K = \frac{x_{max} - x_{min}}{K}$ where x_{max} is the largest x value for any city and x_{min} is the

smallest x value for any city. $\Delta x_S = \frac{x_{max} - x_{min}}{KS}$ is the change in the x position for the lines in adjacent segment tuples. To prevent a bias from the initial starting position of decompositions, a random variable x_{start} is uniformly drawn between the range of 0 and $\frac{x_{max} - x_{min}}{K}$. The location of a given vertical line that partially defined a segment $x_{ij} = (x_{start} + j\Delta x_K + i\,\Delta x_S)\%(x_{max} - x_{min})$ where % is the modulus function, i is the segment tuple number and j is the segment number. A city g at location (x_g, y_g), belongs to a segment T_{ij} under two possible conditions. If $x_{ij} < x_{ij+1}$, city g belongs to segment T_{ij} if $x_g > x_{ij}$ and $x_g \leq x_{ij+1}$. If $x_{ij} \geq x_{ij+1}$, city g belongs to segment T_{ij} if $x_g \leq x_{ij}$ or $x_g > x_{ij+1}$. Figure 44 illustrates the basic method used to perform vertical segmentation on a notional TSP for K =3 for the i^{th} tuple.

Like the radial decompositions, the vertical decompositions can be ordered for gradual changes or more dramatic changes in the objectives. The first sequence places the segments in the order $i = \{1,2,3,\ldots,S\}$. This sequence generates a gradual change in the decomposed objectives as few cities will switch to different segments when a new segment tuple is used. This sequence was not studied in the experiments here due to the uncompetitive peformance of similar sequences. The most dramatic sequence considered places the segments in the order $i = \{1, \frac{S}{2}, 2, \frac{S+2}{2}, 3, \frac{S+4}{2}, 4, \ldots, \frac{S}{2} - 1, \frac{S+S}{2}\}$. Vertical decompositions using this sequence were studied and associated cases are further labeled as ‘vertical phased’.

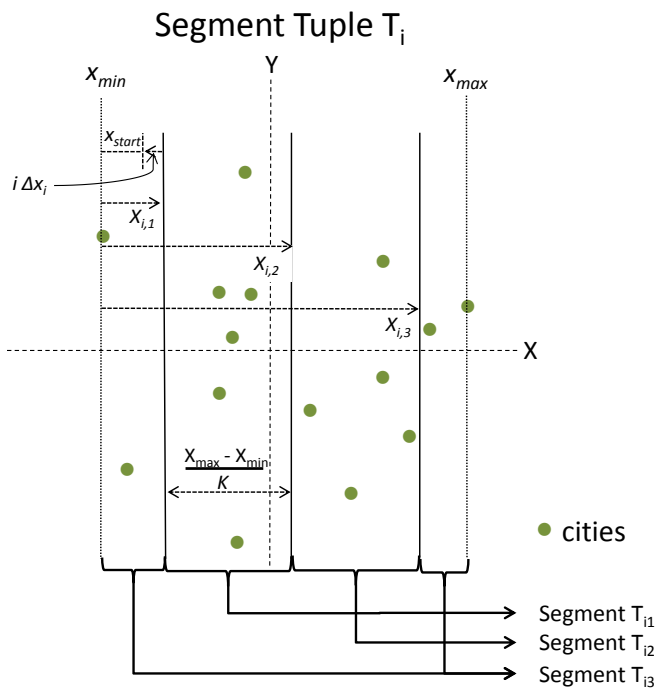


Figure 44 – Vertical segmentation of cities in a TSP

6.3.3.3 Hybrid decompositions

Since EVOD and NNROD do not extend well into higher dimensions, because radial decompositions are competitive with EVOD, and because spatial methods cannot distinguish between cities within and between neighborhoods, a combination of the inter- and intra-neighborhood decomposition methods with spatially oriented decompositions may give better performance than either of the two types of decompositions could produce alone. These hybrid methods blend spatial decompositions with intra-inter neighborhood decompositions. Spatial and intra- and inter-neighborhood segmentation methods each require a minimum of two segments. The hybrid methods construct one segment for intra-neighborhood cities while the remaining segments are designated for cities divided by spatial decomposition. Therefore, hybrid methods that combine these two segmentation methods must create at least three segments – at least two spatial segments and exactly one intra-neighborhood segment. First each city is assigned to one of *K*-1 segments through the spatial segmentation method. Then, 1/*K* cities associated with the cities between neighborhoods are removed from the populated segments and are added to the remaining

single intra-neighborhood segment using either the POD or NNROD methods. After a hybrid decomposition is complete, cities between neighborhoods should be assigned to a single segment while cities within neighborhoods are assigned to two or more different segments based on the spatial segmentation method. Hybrid methods tested in this experiment include POD with radial phased segmentation, NNROD with radial phased segmentation, POD with vertical phased segmentation, and NNROD with vertical phased segmentation.

6.3.3.4 Results

This experiment compared four new hybrid decomposition methods with 3 segments per segment tuple and two new decomposition methods with the two most competitive methods in Experiment 1. The results of this comparison are outlined in Table 17. NNROD achieved the best performance in 7 of the 13 instances. No other method obtained the best performance in more than 2 of the 13 instances. Furthermore, NNROD had the best overall performance when averaged across all problem instances. NNROD appears to do particularly well on non-VLSI applications. EVOD had a better average performance than NNROD on the VLSI instances. Many of the VLSI TSPs have neighborhoods with relatively similar densities. Since the benefit of NNROD is in differentiating between neighborhoods with differing densities, NNROD's did not achieve any performance advantage compared to EVOD in these VLSI problems. The relative performance of NNROD to EVOD strongly confirms the weaknesses in EVOD's method of addressing variable density neighborhoods. NNROD provides a mechanism for addressing such vulnerabilities.

Table 17 – Experiment 2 - Average percentage from the optimal solution for various segmentation types and sizes

	Pure Methods (K = 2)				Hybrid Methods (K = 3)			
	EVOD	**NNROD**	**Radial Phased**	**Vertical Phased**	**POD & Radial Phased**	**POD & Vertical Phased**	**NNROD & Radial Phased**	**NNROD & Vertical Phased**
pr107	0.0035*†	**0.0003**	0.0022*	0.0045	0.0331*‡	0.0064*	0.0293*	0.0230*†
pr124	0.0008	0.0014	0.0005	**0.0000**	**0.0000**	0.0013	0.0032*†	0.0048*†
pr136	0.1359*†	**0.0955**	0.1169*	0.1204*	0.1560*	0.1662*	0.1782*†	0.1792*†
pr152	0.0106	0.0261*†	**0.0060**	0.0154*‡	0.0602*†	0.0582*†	0.0308*	0.0416*
pr299	0.1569*†	**0.1142**	0.1443*	0.1498*	0.2836*	0.2689*	0.3359*†‡	0.3019*†
rat195	0.5415*†	**0.4954**	0.5155	0.4982	0.5503*	0.5311	0.5276	0.5367*
d198	0.1937*†	**0.0875**	0.1637*	0.1509*	0.2884*	0.3269*‡	0.4133*†	0.4495*†‡
xqf131	0.0258*	0.0526*†	0.0437*	0.0472*	**0.0116**	0.0196	0.0249*†	0.0241*
xqg237	0.3215*	0.3127	0.3299*‡	**0.2845**	0.3689*	0.3713*	0.3985*	0.3753*
pma343	0.1745*†	**0.1098**	0.1543*	0.1359*	0.4570*‡	0.3721*	0.4878*‡	0.4125*†
pka379	0.1784*†	**0.1513**	0.1698*	0.1611	0.6180*‡	0.4772*	0.5893*‡	0.4731*
bcl380	0.1293	0.2254*†	0.2328*	0.2148*	0.1128	**0.1076**	0.1535*†	0.1435*†
pbl395	0.2040	0.2675*†	0.2848*	0.2271*	0.2256*†	0.2197*	**0.1734**	0.1918
Average VLSI TSPs	**0.1723**	0.1865	0.2025	0.1785	0.2990	0.2613	0.3046	0.2700
Average Other TSPs	0.1490	**0.1172**	0.1356	0.1342	0.1960	0.1941	0.2169	0.2195
Average All TSPs	0.1597	**0.1492**	0.1665	0.1546	0.2435	0.2251	0.2574	0.2428

Values in **bold** denote the best value found for the problem instance across the cases reported. Values with a '*' denote runs with a statistically higher average percentage from optimal in comparison to the best for the problem instance. Values with a '†' indicate a statistically higher average percentage from optimal when comparing EVOD to NNROD, POD and radial phased to NNROD and radial phased, and POD and vertical phased to NNROD and vertical phased. Values with a '‡' indicate a statistically higher average percentage from optimal when comparing radial phased to vertical phased, POD and radial phased to POD and vertical phased, and NNROD and radial phased to NNROD and vertical phased. All statistical tests used the student's t-test with an $\alpha = 0.05$ level assuming unequal variances.

The theorized shortcoming of radial phased methods in splitting a large number of vertically oriented neighborhoods led to the development of a vertical segmentation method outlined above. We theorized vertical segmentation should have relatively good performance on VSLI instances. The average performance of the vertical phasing on VLSI instances was considerably better than the radial phased methods. This result provides additional evidence that spatially oriented segmentations should attempt to minimize disruption of neighborhoods. Furthermore, vertical phased segmentation had roughly equivalent to slightly improved performance on non-VLSI applications when compared to radial phased segmentation with a difference of 0.0014% between the methods. This indicates that spatial methods that 'splice' two disjoint areas of the problem into a single segment do not seem to have a large weakness versus methods that prevent such a splice from occurring. Averaged over all problem instances radial phased performed worse than vertical phased.

The hybrid methods had relatively disappointing performances. It was expected that these methods would outperform the purely spatial segmentation methods with a similar number of segments in Table 16. This was not the case. The intra- and inter-neighborhood segmentation methods like EVOD and NNROD, enable the algorithm to capture improvements in how neighborhoods are sequenced or traversed independent of each other and their locations but intra- and inter-methods do not easily capture improvements from creating a new entry point into a neighborhood as the cities associated with the new entry point may be in a sub-optimal segment. Spatially oriented methods can address this weakness since creating a new entry point into a neighborhood may be easily recognized as long as the new links are contained within a segment. But, spatially oriented methods may not easily capture new linkages between neighborhoods if the cities associated with the new link are in different segments. We theorize that the combination of intra- and inter-neighborhood segmentation and spatial segmentation brings the worst of these two decompositions together. These hybrid methods have difficulty in both recognizing new

entry points into a neighborhood as well as new ways to link existing neighborhoods with existing entry points.

Experiments 1 and 2 focused on how utilizing decompositions that give improved SNRs can result in better performance. Experiment 3 explores how the SNR changes over time in the optimization and analyzes a segmentation method inspired by the understanding that signal decreases as the optimization progresses through generations.

6.3.4 Experiment 3

SNRs likely change over the course of an optimization. We believe that SNRs generally decrease as a run progresses because improvements in solutions become more and more difficult to generate. When SNRs get lower, the problem can be addressed efficiently with a more aggressive multi-objectivization method. These SNRs also change with a number of problem properties including problem size. In the previous two experiments, problem features were not always similar between the different instances of various size. Since the properties are different, we may not be able to strongly show how a multi-objective method changes with larger problem sizes. In order to study the relationship between problem size and level of multi-objectivization, it is useful to study problems that have as many similar characteristics as possible while still varying problem size.

The 8th DIMACS Implementation Challenge generated a program called ‘portgen’ that creates TSP instances by placing *N* cities with independent random uniform X and Y coordinates in a 1000000 by 1000000 grid. This utility is available for download at http://www2.research.att.com/~dsj/chtsp/download.html. Because the cities are generated using random process, it is likely that the problems will have similar characteristics as the size of the problem increases. Portgen uses a pseudo random number generator and as such produces repeatable problem instances. Seed number 1 was used to generate problem instances with

different numbers of cities. Since the portgen utility uses pseudo-random numbers to generate cities, problems generated with the same random seed numbers will have many cities in common.

We used portgen to create 3 interrelated TSP instances using the seed number 1. Instances of 100, 200, and 400 cities were generated. These problem instances are further indicated by their generator name, number of cities and seed number. For instance 'portgen-200-1' indicates the TSP instance generated with 200 cities using random seed number 1. To understand the convergence of different algorithms, these three problems were run for 1000 generations with various levels of *K*. Vertical segmentation was used because it extends well into higher dimensions and has a competitive performance to other methods. All other algorithmic settings were consistent with the previous two experiments. The average best solution by generation for each level of *K* is plotted in Figures 45-47 for the three instances. The cases with *K* values of 1 are single objective GAs since a single decomposition includes all cities. The best known fitness value found here for portgen-100-1 is 7,492,995; for portgen-200-1 is 10,639,978; and for portgen-400-1 is 14,767,181. No previously published best known fitness values were found for these problems. The best known fitness value for each problem is displayed in its respective graph. A common trend exists in all three graphs that is consistent with previous observations of helper-objectives in previous works (Jensen, 2004, Lochtefeld and Ciarallo, 2011b). These results show that the less aggressive multi-objectivization methods converge faster but can prematurely converge to a sub-optimal solution during the process. As the degrees of multi-objectivization increase with rising levels of *K*, early convergence is slowed. Excluding *K* values of 4 and 5 that did not converge in the allotted 1000 generations, at the end of the runs larger values of *K* generally resulted in better performance than smaller values of *K*. This strongly indicates that as the budget of fitness evaluations for a given problem increased, the problem is more amenable to higher degrees of multi-objectivization.

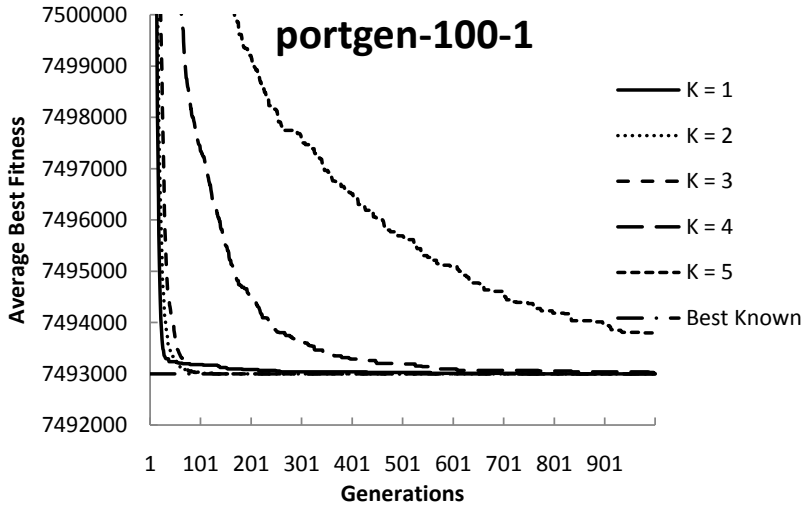


Figure 45 – Average best fitness by generation and level of K for portgen-100-1

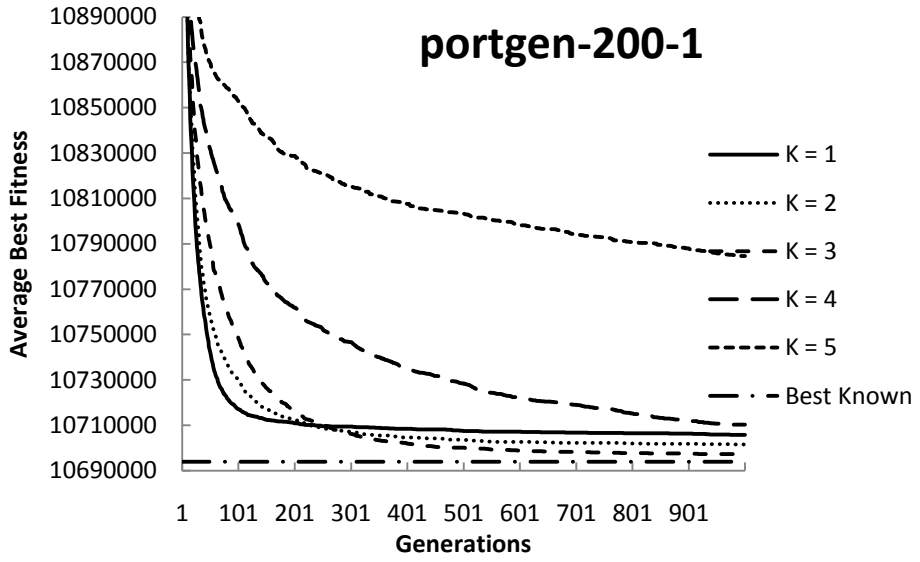


Figure 46 – Average best fitness by generation and level of K for portgen-200-1

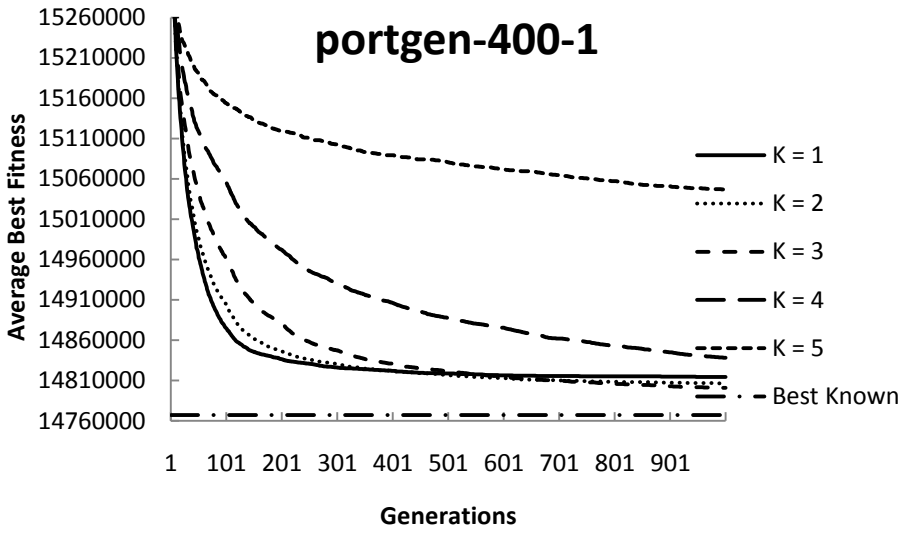


Figure 47 – Average best fitness by generation and level of K for portgen-400-1

It may be possible to combine the different levels of multi-objectivization in a way that gives a robust algorithm that works for any budget of fitness evaluations. Such an algorithm could start with a low *K* value and increase the level of multi-objectivization over time. In principle, we would move to the next higher level of *K* when the rate of fitness improvement in the current level of *K* is less than or equal to the rate of fitness improvement in the next higher level of *K*. With appropriate switching between the levels of decompositions, we could generate a more robust and general method that works well for a variety of computational budgets. Of course switching between levels of *K* in theory is different than in practice. The theory assumes that we already know the convergence profiles but we won't have this information in a practical optimization setting.

6.3.4.1 Progressive Segmentation

MOS used three parameters to govern the definition and switching of various decompositions. The value of *K* determined the fixed number of segments per segment tuple. In MOS the value of *K* did not change in a run and in previous work it was always set to a value of 2. The parameter *h* governed the number of unique decompositions used in a given run. Related

to *h* and the number of generations *G*, a derived parameter *d* determined the number of generations before switching to a new decomposition.

The Multi-Objectivization via Progressive Segmentation (MOPS) algorithm extends MOS to vary the level of multi-objectivization over time. In addition to the parameters associated with MOS, MOPS needs parameters to govern the switching rules to higher levels of *K*. MOPS starts with a K_{min} level decomposition. A K_{min} level of 1 corresponds to a single objective genetic algorithm. The algorithm can increment *K* at the end of any generation according to a switching rule until a maximum level of *K* is reached or the budgeted *G* generations are reached. The switching rule proposed here utilizes two parameters *b* and *f*. *K* is incremented based on a consecutive number of generations that have passed without finding an improvement in the best known solution. The number of generations without improvement, g, before switching to a higher level of *K* follows the relationship $g = bf^{K_c-1}$ where K_c is the current value of *K*. As more generations are processed, it becomes more difficult to find new best solutions and so each additional degree of multi-objectivization should be given more time than the previous degree. As a result good values for the parameter *f* are likely greater than 1. In MOS the parameter *h* was selected by the analyst and the parameter *d* was set to *G/h*. We use the same procedure here and use an *h* value of 10 as in the previous experiments above. A summary of the parameters required for MOPS are outlined in Table 18.

Table 18 – Parameters of MOPS

Parameter	Description
K_{min}	The smallest value for *K* that the optimization can utilize
K_{max}	The greatest value for *K* that the optimization can utilize
b	The number of generations without a best fitness improvement before incrementing *K* the first time
f	The factor to increase the number of generations without fitness improvement for determining when further increments of *K* occur
d	The number of generations per segmentation before a new segment is used in a given level of *K*
h	The maximum number of unique decompositions per level of *K*

When $K_{min} = K_{max} = 1$ every decomposition includes all portions of the main objective in a single objective. When the parameter values are set such that $K_{min} = K_{max}$, the parameters *b* and *f*

become irrelevant as they govern the switching to higher degrees of decomposition. In the K_{min} = K_{max} = 1 case, parameters *h* and *d* are also irrelevant. In this case, every decomposition is identical because a given decomposition must cover every element of the main objective and because there is only one segment in a given decomposition. Since algorithms such as NSGA-II generally reward Pareto dominant solutions before considering the distribution of solutions within a Pareto frontier, these algorithms working on a single objective will reward the highest fitness solutions, similar to how a more standard GA would reward high fitness solutions. Thus when K_{min} = K_{max} = 1 the MOPS algorithm behaves as a single objective GA.

More generally when K_{min} = K_{max} there is a fixed number of degrees of decomposition. This means the algorithm either becomes a single objective genetic algorithm (in the special case discussed before) or a multi-objectivization algorithm identical to MOS. Thus, MOPS can be as simple as a single objective GA, an algorithm with a constant degree of multi-objectivization (such as MOS), or a more sophisticated multi-phased multi-objectivization method.

6.3.4.2 Analysis of MOPS

To test progressive segmentation, we run the same instances studied in Experiments 1 and 2 and compare the relative performances of MOS with fixed levels of *K* against MOPS for three different computational budgets. This experiment used the same number of fitness evaluations as in the previous experiments. K_{min} was set to a value of 1, which starts the optimization process as a single objective genetic algorithm. K_{max} was set to a value of 3 because in Figures 45 – 47, *K* values of 4 and 5 were not competitive in the 200 and 400 city problems. Values and *b* and *f* were set according to preliminary experiments on the rat195 and pbl395 instances discussed in the following paragraph. The number of fitness evaluates for this experiment was varied based on the same formula as above using m values of 5, 10, and 15. All other settings were consistent with the previous experiments including the number of fitness evaluations.

To find good values of *b* and *f*, an exploration of different values was accomplished on the rat195 and pbl395 instances. A full factorial experiment for *b* values of 1, 2, 5, 10, 15, 20 and f values of 2, 3, 5, 10 was run for 200 replications per experimental setting and problem instance using the a computational budget based on an *m* value of 5. Also cases were run for $K_{min} = K_{max}$ for *K* values of 1, 2, and 3 – these cases correspond with the single objective genetic algorithm and the MOS algorithm for $K = 2$ and $K = 3$. An average best found value for each generation was recorded. Since the goal is to create a robust algorithm that gives good results for different budgets, we used a measure of dominance to determine which parameter settings to use. At each generation a percentage from the average best value for the best case in that generation was used to calculate how far from the best method in that generation the other algorithms were. These results were summed across all generations giving a single score for each method in terms of their dominance relative to the other methods. Based upon these scores we selected parameter values of $b = 15$ and $f = 2$. These two settings gave the most dominant result of any method for the pbl395 instance and a near-best result for the rat195 problem.

Three separate budgets against the set of instances used in Experiments 1 and 2 were examined to determine the robustness of the MOPS algorithm. The maximum number of fitness evaluations performed, *E*, scaled as the pervious experiments according to $E = m\sqrt{N^3}$. Budgets with *m* values of 5, 10 and 15 were tested. Constant *K* values of 1, 2, and 3 were considered in addition to the variable *K* value used by progressive segmentation. Table 19 contains the average best fitness as a percentage distance from the optimal solution for 200 replications for the various cases outlined above. Out of the static methods where *K* was constant, for the $m = 5$ budget the single objective GA was best on average across all instances. For the two larger computational budgets and constant *K* values, the $K = 3$ cases were best on average across all instances. These results confirm that as computational budget increases, problems are more amenable to higher degrees of decomposition.

Table 19 – Experiment 3 - Average percentage from the optimal solution for various methods and computational budgets using radial segmentation

	***m* = 5**				***m* = 10**				***m* = 15**			
	***K* = 1**	***K* = 2**	***K* = 3**	***K* = 1 to 3**	***K* = 1**	***K* = 2**	***K* = 3**	***K* = 1 to 3**	***K* = 1**	***K* = 2**	***K* = 3**	***K* = 1 to 3**
pr107	**0.0000**	0.0044*	0.0003	0.0007	**0.0000**	0.0009	**0.0000**	0.0003	**0.0000**	**0.0000**	**0.0000**	**0.0000**
pr124	**0.0000**	**0.0000**	0.0086*-	0.0004	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
pr136	**0.1014**	0.1204*-	0.1344*-	0.1077	**0.0551+**	0.0725*	0.0592+	0.0704*	0.0420*-	0.0474*-	**0.0311**	0.0341
pr152	0.0279*	0.0154	**0.0091+**	0.0246*	0.0213*-	0.0049	**0.0012**	0.0049	0.0093*-	0.0010	**0.0001**	0.0011
pr299	**0.1204+**	0.1498*	0.1809*-	0.1462*	0.1177*	0.1168*	**0.0915+**	0.1169*	0.1204*-	0.1013	**0.0834**	0.0931
rat195	0.5004	**0.4982**	0.5043	0.5082	0.4669*+	0.3954*	**0.3487**	0.3702	0.4572*-	0.3743*-	**0.2622+**	0.3175*
d198	**0.0829**	0.1509*-	0.4441*-	0.0861	**0.0597+**	0.0714+	0.3739*-	0.0919*	**0.0414+**	0.0490*+	0.3496*-	0.0820*
xqf131	0.0479*	0.0472*	**0.0301**	0.0426	0.0461*	0.0417*	**0.0089+**	0.0408*	0.0381*-	0.0381*-	**0.0018+**	0.0186*
xqg237	0.3106	**0.2845**	0.2866	0.2978	0.2444*	0.2193	**0.2071**	0.2262	0.2439*-	0.1938*	**0.1619**	0.1703
pma343	**0.1224**	0.1359	0.1345	0.1202	0.0998*	0.0987*	**0.0683+**	0.0852*	0.0837*-	0.0805*-	0.0705	**0.0621**
pka379	**0.1396**	0.1611*	0.1678*-	0.1475	0.1325	0.1242	0.1239	0.1265	0.1197	0.1175	**0.1137**	0.1201
bcl380	0.2400*-	0.2148*-	**0.1570**	0.1777	0.1882*-	0.1598*-	**0.0713**	0.0885	0.1579*-	0.1200*-	**0.0524+**	0.0756*
pbl395	0.2440*	0.2271*	**0.1585+**	0.2619*	0.2088*	0.1335*	**0.0504+**	0.1671*	0.1717*	0.1120*	**0.0265+**	0.1327*
Average	0.1490	0.1546	0.1705	**0.1478**	0.1262	0.1107	0.1081	**0.1068**	0.1143	0.0950	0.0887	**0.0852**

Values that are **bold** indicate the best performance for a give value of *m*. Values marked with a '*' are statistically worse than the overall best performing case for that value of *m*. Values identified with a '+' in a given level of *m* have a statistically lower average percent from optimal value than the MOPS case indicating the case outperformed MOPS. Conversely, values marked with a '-' have a statistically

From Table 19 observe while progressive segmentation was rarely the best for a given problem instance, the average performance across all instances was better than a constant degree of decomposition. This was the case for all three computational budgets. MOPS, with variable strength decomposition, is a robust method that gives the best overall performance across a large variety of problems and budgets. In cases where the computational budget is unknown *a priori* or where little is known about the individual problem instances performance across various levels of *K*, progressive segmentation is a good choice for optimization. While progressive segmentation introduces several parameters, it also relaxes the need to fix other parameters such as computational budget and the degree of multi-objectivization.

Figure 48 demonstrates how the convergence of progressive decomposition differs from the methods with a constant degree of decomposition. In approximately the first 50 generations the method tracks with the single objective genetic algorithm. It then slows its convergence some in comparison to the single objective method. This slowdown may be caused by the disruption to the evolutionary process from changing the degree of decomposition. The method overcomes the single objective GA around 190 generations and then tracks relatively closely to the $K = 3$ method. This demonstrates the robustness of the method – the early convergence from the single objective GA is mixed with avoidance of premature convergence by using increasingly more aggressive multi-objectivization.

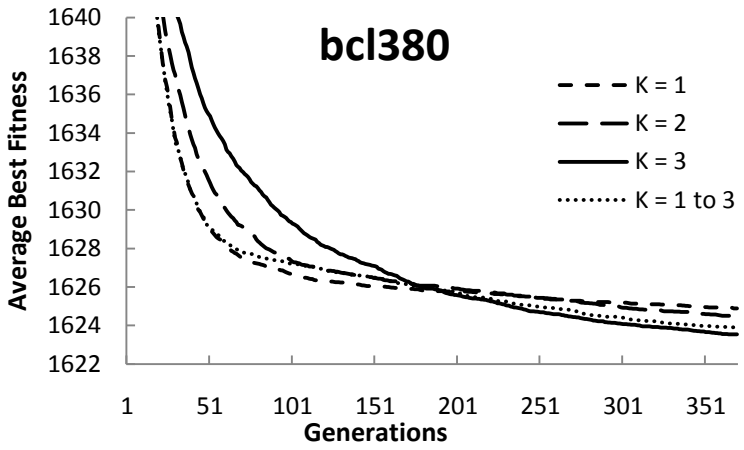


Figure 48 – Best average fitness by generation for the bcl380 instance with computational budget of m = 5

6.4 Chapter Conclusion

Based on an understanding of the principles of multi-objectivization, we provided deepened insight into the mechanisms that make MOS decompositions successful. We show why the EVOD method works – it distinguishes between cities within and cities between neighborhoods. We proposed three new decompositions: NNROD, radial segmentation, and vertical segmentation. NNROD is an inter- and intra-neighborhood decomposition method that uses a new measure of neighborhood density to assign cities to various decompositions. The NNROD had superior performance compared to EVOD on the majority of instances examined. The new radial and vertical decomposition methods are spatially oriented decompositions. These two methods were inspired by the signal-to-noise ratio principle that indicates local improvements should be isolated from simultaneous negative moves in other areas of the solution. These two methods had performance comparable to EVOD and NNROD for lower degrees of decomposition but also appear to be more extendable into higher degrees of decomposition. Goals of spatial decompositions should be to maintain neighborhoods and balanced decompositions.

In addition to analysis of the type of segmentation, the degree of multi-objectivization was examined as it relates to computational budget. We conclude that as the computational budget

increases, more aggressive forms of multi-objectivization should be considered. A new method, Multi-Objectivization via Progressive Segmentation (MOPS) is proposed. The MOPS method is capable of modeling a single objective GA, a multi-objectivization method with a constant degree of multi-objectivization, or a multi-objectivization method with an increasing degree of multi-objectivization. Progressive segmentation was compared to the single objective GA and the two most competitive degrees of multi-objectivization for various computational budgets. Progressive segmentation was found to be a robust alternative to the constant degree methods in the instances tested.

There are numerous follow-on efforts that are envisioned. Additional work can be accomplished to understand how much disruption is generated when switching to a higher degree of multi-objectivization. New switching rules can be examined to determine which rules are best applied and when they should be applied. MOPS can also be tested on other problems to more strongly determine its value. Furthermore, the study of other problems beyond the TSP naturally leads to the definition of new segmentation methods. Imbalanced segmentations could also be considered in the future since we know that some parts of a problem are more critical than other portions.

Chapter 7 A comparison of helpers and complete decomposition

The two major methods used in MVD to date are helper-objectives (Jensen, 2004) and segmentation (Jahne et al., 2009). The only empirical evidence directly comparing these methods to date was accomplished on the TSP in Jahne et al. (2009). We explore more deeply the relationship between helper-objectives and segmentation methods. We explore analytically how Pareto efficient frontiers interrelate between the two methods for additive fitness functions. An empirical study on the JSSP is also used to explore the relationship between complete decomposition and helper methods.

7.1 Related background review

Helper-objectives were proposed by Jensen (2004). Helper-objectives are additional objectives that are used simultaneously with the main objective because the "additional objectives [can guide] the search" (Jensen, 2004). In the work, both the TSP and the JSSP were studied. Jensen used objectives that were decomposed parts of the main objective as the helper-objectives. Helper-objectives are further called helpers throughout this document.

Helpers on the JSSP were later studied in Lochtefeld and Ciarallo (2011b). In previous helper related research, the sequence of helpers was chosen randomly from all possible jobs. Lochtefeld and Ciarallo (2011b) used a deterministic order of helpers to obtain better results using the Shortest Job First (SJF) rule. The premise of the SJF rule is that it is better to delay a long job by a short job's time than vice-versa, when optimizing total flowtime.

Subsequent to helper objectives, Jahne et. al (Jahne et al., 2009) proposed Multi-Objectivization via Segmentation (MOS). Unlike helper methods which use the main objective simultaneously with a portion of a decomposed objective, MOS uses all parts of the decomposed

objectives simultaneously and does not explicitly use the main objective. Each part of a decomposition was termed a segment. We term this type of MVD *complete decomposition*. As the name implies, complete decomposition takes all component parts of the main objective and divides them into a number of new objectives such that each component of the main objective is represented an equal number of times and all decomposed parts are assigned to an objective. Equal representation of the objective components preserves the original problem as it is decomposed. Unlike previous approaches, several of the decompositions in MOS were adaptive; the decompositions were based on properties of solutions in the current population. The complete decompositions used in MOS generally outperformed the helper methods on the TSP instances studied. Lochtefeld and Ciarallo (2011a) studied MOS and proposed a more general version of MOS called Multi-objectivization via Progressive Segmentation (MOPS). When progressive segmentation is not enabled in MOPS, MOPS is identical to MOS. Progressive segmentation increases the degree of decomposition over time as the evidence provided indicated that using more aggressive multi-objectivization later in the run works best.

One popular MOEA used in multi-objectivization is the Non-dominated Sorting Genetic Algorithm version II (NSGA-II) which is described in Deb et al. (2002). NSGA-II uses Pareto dominance relationships between solutions to determine the first order of magnitude of fitness for the solution. Then, between solutions that are within the same front, NSGA-II uses a hyper-boxed based penalty function to reward solutions that are far apart so that solutions, hopefully, do not cluster in the same area on the Pareto front.

The dominance relationship in NSGA-II is determined as follows. All solutions in the population are searched to find those solutions that are non-dominated, that is, those solutions that are not strongly dominated by any other solution in the population. These non-dominated solutions are all labeled with the front number 1 and are removed from further consideration. Then the remaining population is again searched for non-dominated solutions. These solutions are

labeled with the front number 2 and are removed from further consideration. This process proceeds until all individuals in the population have been assigned a front number. In NSGA-II, any individual with a lower front number is considered a more-fit individual than an individual with a higher front number.

7.2 Relating Helpers and Complete Decomposition

In practice, helpers in Jensen (2004) and Lochtefeld and Ciarallo (2011b) and complete decomposition in Jahne et al. (2009) and Lochtefeld and Ciarallo (2011a) seem to share more similarities than differences. The two methods use the same strategy for mitigating a poor decomposition by using new decompositions over time. The methods both swap in new decompositions using the same logic. Also, both methods are MVD methods, dividing the main objective into component parts. The only major difference between the methods is in the definition of the decomposition itself. In helpers, the main objective is used in place of the final decomposed part that would have been used in complete decomposition. Thus, in helpers, the main objective is explicitly an objective while in complete decomposition the main objective is implicit. A complete decomposition can be created from a helper by paring down the main objective to only those objective components that are not in the helpers. With this in mind, we embarked on a careful direct comparison of complete decompositions versus helpers. For simplicity in the analysis below we assume a minimization problem of an additive and decomposable objective function. Similar analysis holds for decomposable objective functions associated with an additive maximization problems and additive minimization problems.

The analysis below explores general relationships that hold between solutions that are compared in the Pareto sense both in helper decompositions and complete decompositions. Building on these fundamentals we show a mapping of Pareto efficient solutions in helpers to corresponding Pareto efficient solutions in complete decompositions. This leads to the

conclusion that solutions in the i^{th} Pareto frontier in helpers appear on the i^{th} or lower frontiers in complete decomposition.

We use several definitions of Pareto comparison operators in the work below. Solution x strongly dominates solution y is represented by the notation $x \succ y$. Solution x is as-least-as-good-as, (i.e. weakly dominates), solution y is represented by $x \succcurlyeq y$. Solutions x and y are Pareto incomparable is represented by $x \approx\!\!\!\!\!\!\cdot\ \ y$. Solutions x and y are Pareto identical is represented by $x \approx y$. Solution y is not strongly dominated by x is represented by $x \not\succ y$. The analysis of helpers and complete decomposition methods can lead to multiple Pareto comparisons because the decompositions have different objectives, so when needed for clarity in the following development the Pareto operators are augmented with H and C respectively. For example $x \overset{\succ}{}_{H} y$ denotes that solution x strongly dominates solution y based on a helper-objective structure.

We use the following notation to represent how solutions are evaluated with the objectives under helper and complete decompositions. Let x_k denote the k^{th} objective value associated with solution x. In our work, both helpers and complete decomposition methods contain exactly m simultaneous objectives to be minimized. For both methods m indicates the degree of decomposition. For complete decomposition $k \in \{1,2,\dots,m\}$. For helpers $k \in \{1,2,\dots,m-1,M\}$ where M is the main objective and $m-1$ is the number of helpers.

7.2.1 Definitions:

The main objective M is the sum of all decomposed objectives.

$$x_M = \sum_{k=1}^{m} x_k \tag{5}$$

$$y_M = \sum_{k=1}^{m} y_k \tag{6}$$

Definition 1: Solution x strongly dominates solution y in decomposition O, denoted $x \overset{\succ}{}_{O} y$, if x is better than y in at least one objective and x is no worse than y in all other objectives.

$$x \succ y \ \ iff \ \ [\cup_{k=1}^{m} x_k < y_k] \cap [\cap_{k=1}^{m} x_k \leq y_k] \text{ is true} \tag{7}$$

$$x \underset{C}{\succ} y \ \ iff \ \ \left[\{\cup_{k=1}^{m-1} x_k < y_k\} \cup \{x_m < y_m\}\right] \cap \left[\{\cap_{k=1}^{m-1} x_k \leq y_k\} \cap \{x_m \leq y_m\}\right] \text{ is true} \tag{8}$$

$$x \underset{H}{\succ} y \ \ iff \ \ \left[\{\cup_{k=1}^{m-1} x_k < y_k\} \cup \{x_M < y_M\}\right] \cap \left[\{\cap_{k=1}^{m-1} x_k \leq y_k\} \cap \{x_M \leq y_M\}\right] \text{ is true} \tag{9}$$

Definition 2: Solution x weakly dominates solution y in decomposition O, denoted $x \underset{O}{\succcurlyeq} y$, if x is no worse than y in all objectives.

$$if \ x \succcurlyeq y \ \ then \ \ \cap_{k=1}^{m} x_k \leq y_k \text{ is true} \tag{10}$$

$$if \ x \underset{C}{\succcurlyeq} y \ then \ \ \cap_{k=1}^{m-1} x_k \leq y_k \cap x_m \leq y_m \text{ is true} \tag{11}$$

$$if \ x \underset{H}{\succcurlyeq} y \ then \ \ \cap_{k=1}^{m-1} x_k \leq y_k \cap x_M \leq y_M \text{ is true} \tag{12}$$

Definition 3: Solution x is incomparable to y in decomposition O, denoted $x \underset{O}{\succcurlyeq} y$ if x is worse than y in at least one objective and x is better than y in at least one objective.

$$x \succcurlyeq y \ \ iff \ \ \{\cup_{k=1}^{m} x_k < y_k\} \cap \{\cup_{k=1}^{m} y_k < x_k\} \text{ is true} \tag{13}$$

$$x \underset{C}{\succcurlyeq} y \ iff \ \ \left[\{\cup_{k=1}^{m-1} x_k < y_k\} \cup \{x_m < y_m\}\right] \cap \left[\{\cup_{k=1}^{m-1} y_k < x_k\} \cup \{y_m < x_m\}\right] \text{ is true} \tag{14}$$

$$x \underset{H}{\succcurlyeq} y \ iff \ \ \left[\{\cup_{k=1}^{m-1} x_k < y_k\} \cup \{x_M < y_M\}\right] \cap \left[\{\cup_{k=1}^{m-1} y_k < x_k\} \cup \{y_M < x_M\}\right] \text{ is true} \tag{15}$$

Definition 4: Solutions x and y are Pareto identical in decomposition O, denoted $x \underset{O}{\approx} y$, if each of the corresponding objective values in x and y are identical.

$$x \approx y \ \ iff \ \ \cap_{k=1}^{m} x_k = y_k \text{ is true} \tag{16}$$

$$x \underset{C}{\approx} y \ iff \ \ \cap_{k=1}^{m-1} x_k = y_k \cap x_m = y_m \text{ is true} \tag{17}$$

$$x \underset{H}{\approx} y \ iff \ \ \cap_{k=1}^{m-1} \ _k = y_k \cap x_M = y_M \text{ is true} \tag{18}$$

Property 1: For two non-identical solutions x and y, if y is not strongly dominated by x it is straightforward to show that either y weakly dominates x, or y is Pareto incomparable to x.

$$\text{If } x \nsucc y \text{ then } \{\bigcap_{k=1}^{m} x_k \geq y_k\} \cup \{\bigcup_{k=1}^{m} x_k > y_k\} \text{ is true} \tag{19}$$

$$\text{If } x \overset{\nsucc}{C} y \text{ then } \left[\{\bigcap_{k=1}^{m-1} x_k \geq y_k\} \cap \{x_m \geq y_m\}\right] \cup \left[\{\bigcup_{k=1}^{m-1} x_k > y_k\} \cup \{x_m > y_m\}\right] \text{ is true} \tag{20}$$

$$\text{If } x \overset{\nsucc}{H} y \text{ then } \left[\{\bigcap_{k=1}^{m-1} x_k \geq y_k\} \cap \{x_M \geq y_M\}\right] \cup \left[\{\bigcup_{k=1}^{m-1} x_k > y_k\} \cup \{x_M > y_M\}\right] \text{ is true} \tag{21}$$

All solutions that are not strongly dominated by any solution are defined to be *Pareto efficient*. Together these solutions make up a set of solutions called the *Pareto efficient frontier* (Kirkwood, 1996,Belton and Stewart, 2002). We define the sets C_1 and H_1 to contain the Pareto efficient solutions in a given population for complete decomposition and helper decomposition, respectively.

Definition 5:

$$y \in C_1 \; iff \; \forall x : x \overset{\nsucc}{C} y \tag{22}$$

$$y \in H_1 \; iff \; \forall x : x \overset{\nsucc}{H} y \tag{23}$$

NSGA-II also distinguishes between those solutions in the population in subsequent fronts so that the algorithm can determine which of the dominated solutions have better fitness. Subsequent fronts are defined by eliminating the solutions in the population in previous fronts from consideration and then, in the remaining solutions, finding the solutions that are not strongly dominated to define a new front. We define the sets C_i and H_i to contain all solutions on the i^{th} Pareto frontier for complete decomposition and helpers respectively, for $i \in \{1, \dots, \infty\}$, with $C_0 \equiv \emptyset$ and $H_0 \equiv \emptyset$.. Further define the sets C_i^+ and H_i^+ to contain all solutions in i^{th} Pareto frontier as well as those in the previous frontiers building up to the i^{th} frontier for the associated decomposition method.

Definition 6:

$$H_i^+ = \bigcup_{k=1}^{i} H_k \tag{24}$$

$$C_i^+ = \bigcup_{k=1}^{i} C_k \tag{25}$$

Property 2: If a solution is contained in the i^{th} frontier, then it is straightforward to show that it is not contained in any of frontiers 1 through $i-1$.

$$\forall i \in \{1, \dots, \infty\}, if\ x \in C_i\ then\ x \notin C_{i-1}^+ \tag{26}$$

$$\forall i \in \{1, \dots, \infty\}, if\ x \in H_i\ then\ x \notin H_{i-1}^+ \tag{27}$$

These Definitions and Properties lead to the following major results clarifying the relationship between dominant solutions under complete decompositions versus helper decompositions in multi-objectivization:

Postulate 1: A solution y that is strongly Pareto dominated by another solution x in complete decomposition is also strongly dominated by that solution in helpers.

$$\text{If } x \overset{\succ}{C} y \text{ then } x \overset{\succ}{H} y.$$

Proof: Assume $x \overset{\succ}{C} y$. By (8), (5), (6), and the property of addition $x_M < y_M$. In going from the complete decomposition evaluation of solutions x and y to the helper decomposition evaluation, we remove exactly one objective and replace it with the main objective, *M*. Thus the conditions in (3) will continue to hold because $x_M < y_M$.

Postulate 2: Any two solutions that are Pareto incomparable in helpers are also Pareto incomparable in complete decomposition.

$$\text{If } x \overset{\nsucc}{H} y \text{ then } x \overset{\nsucc}{C} y.$$

Proof: Assume $x \overset{\approx}{}_{H} y$. We decompose the situation into two mutually exclusive and complete cases: The solutions are either Pareto incomparable in helpers because of two or more conflicting values in the first $m-1$ objectives (Case 1) or the solutions are Pareto incomparable in helpers because of the main objective and one or more conflicting values in the other $m-1$ objectives (Case 2).

Case 1:

Intuition: The solutions are Pareto incomparable in helpers because of two or more conflicting values in the first $m-1$ objectives.

$$\left\{\bigcup_{k=1}^{m-1} y_k < x_k\right\} \cap \left\{\bigcup_{k=1}^{m-1} x_k < y_k\right\} \text{ or } \left\{\bigcup_{k=1}^{m-1} x_k < y_k\right\} \cap \left\{\bigcup_{k=1}^{m-1} y_k < x_k\right\} \text{ is true} \quad (28)$$

If in each solution x and y at least one of the $m-1$ helper objective values is superior, the solutions are Pareto incomparable in both helpers and complete decomposition since these objectives are common in both decompositions.

Case 2:

Intuition: The solutions are Pareto incomparable in helpers because of the value of the main objective for solutions x and y, and one or more conflicting values in the other $m-1$ objectives. Thus

$$\text{Either } \left\{\bigcup_{k=1}^{m-1} y_k < x_k\right\} \cap \{x_M < y_M\} \text{ is true} \quad (29)$$

or

$$\left\{\bigcup_{k=1}^{m-1} x_k < y_k\right\} \cap \{y_M < x_M\} \text{ is true} \quad (30)$$

Since the statements in (29) and (30) are symmetric (through interchange of x and y), only the analysis for (29) is shown here.

From (29) we know that $y_{k^*} < x_{k^*}$ for some $k^* \in \{1, \ldots, m-1\}$ and that $x_M < y_M$.

A. If $x_m < y_m$, together with $y_{k^*} < x_{k^*}$ for some $k^* \in \{1, \dots, m-1\}$, then the condition for $x \overset{\precapprox}{C} y$ in Definition 3 is satisfied.
B. If $x_m \geq y_m$, assume that $y_k \leq x_k$ for all $k \neq k^*$ and $k \in \{1, \dots, m-1\}$. Under this assumption $\sum_{k=1}^{m} x_k > \sum_{k=1}^{m} y_k$, indicating $x_M > y_M$, which is a contradiction. Thus $y_k > x_k$ for at least one $k \neq k^*$ and $k \in \{1, \dots, m-1\}$. Together with $y_{k^*} < x_{k^*}$ for some $k^* \in \{1, \dots, m-1\}$ the condition for $x \overset{\precapprox}{C} y$ in Definition 3 is satisfied.

Postulate 3: Solutions x and y are Pareto identical in helpers if and only if they are Pareto identical in complete decomposition.

$$x \overset{\approx}{C} y \; iff \; x \overset{\approx}{H} y$$

Proof:

Case 1:

Assume $x \overset{\approx}{C} y$, show $x \overset{\approx}{H} y$

By Definition 4 $\forall k, k \in \{1, \dots, m\} : x_k = y_k$. Using (5) and (6) $x_M = y_M$

$$\therefore x \overset{\approx}{H} y \tag{31}$$

Case 2:

Assume $x \overset{\approx}{H} y$, show $x \overset{\approx}{C} y$

By Definition 4 $\forall k, k \in \{1, \dots, m-1\} : x_k = y_k$, and $x_M = y_M$. Using (5) and (6) $x_m = y_m$

$$\therefore x \overset{\approx}{C} y \tag{32}$$

The results in Postulates 1 through 3 are important to reasoning about the *dominated* solutions under helper and complete decompositions. The following results demonstrate important properties of the relationship between the *frontiers of non-dominated solutions* in a population under helper and complete decompositions as used in NSGA-II.

Theorem A: All solutions in the Pareto efficient frontier (the first Pareto frontier) in helper objectives are also in the Pareto efficient frontier in complete decomposition for a sum-of-parts fitness function.

$$\text{If } x \overset{\succ}{}_{H} y \text{ then } x \overset{\succ}{}_{C} y$$

Proof:

Assume $x \overset{\succ}{}_{H} y$, show: $x \overset{\succ}{}_{C} y$. If $x \overset{\succ}{}_{H} y$ then one of three cases must hold:

Case 1: If $\exists k, k \in \{1, \dots, m-1\} : x_k > y_k$, then

$$\therefore \left\{\bigcup_{k=1}^{m-1} x_k > y_k\right\} \cup \{x_m > y_m\} \text{ is true} \tag{33}$$

$$\therefore \left[\left\{\bigcap_{k=1}^{m-1} x_k \geq y_k\right\} \cap \{x_m \geq y_m\}\right] \cup \left[\left\{\bigcup_{k=1}^{m-1} x_k > y_k\right\} \cup \{x_m > y_m\}\right] \text{ is true} \tag{34}$$

Case 2: If $\forall k, k \in \{1, \dots, m-1\}; x_k = y_k, \; and \; x_M > y_M$

Through use of substitution and the property of addition on (5) and (6)

$$x_m > y_m \tag{35}$$

$$\therefore \left\{\bigcup_{k=1}^{m-1} x_k > y_k\right\} \cup \{x_m > y_m\} \text{ is true} \tag{36}$$

$$\therefore \left[\left\{\bigcap_{k=1}^{m-1} x_k \geq y_k\right\} \cap \{x_m \geq y_m\}\right] \cup \left[\left\{\bigcup_{k=1}^{m-1} x_k > y_k\right\} \cup \{x_m > y_m\}\right] \text{ is true} \tag{37}$$

Case 3: If $\forall k, k \in \{1, \dots, m-1\} : x_k = y_k \;\; ; x_M = y_M$

By substitution and subtraction on (5) and (6)

$$x_m = y_m \tag{38}$$

$$\therefore \left\{\bigcap_{k=1}^{m-1} x_k \geq y_k\right\} \cap \{x_m \geq y_m\} \text{ is true} \tag{39}$$

$$\therefore \left[\left\{\bigcap_{k=1}^{m-1} x_k \geq y_k\right\} \cap \{x_m \geq y_m\}\right] \cup \left[\left\{\bigcup_{k=1}^{m-1} x_k > y_k\right\} \cup \{x_m > y_m\}\right] \text{ is true} \tag{40}$$

Theorem B: All solutions in the i^{th} or lower frontier in helpers are in the i^{th} or lower frontier in complete decomposition: $H_i^+ \subseteq C_i^+$.

Proof:

By definition Pareto fronts are mutually exclusive: $H_i \cap H_j = \emptyset : \forall i, j, i \neq j$ and $C_i \cap C_j = \emptyset : \forall i, j, i \neq j$.

For the case where $i = 1$, Theorem A shows the desired result. For the inductive step, assume

$$H_i^+ \subseteq C_i^+ \tag{41}$$

In order to show that

$$H_{i+1}^+ \subseteq C_{i+1}^+. \tag{42}$$

The proof of the inductive step will proceed by assuming that (41) does not hold and identifying a logical contradiction. To achieve this, suppose there exists a point x that is in H_{i+1} but is not in C_{i+1}^+:. We will reason about x relative to a point y that is in C_{i+1}. Suppose

$$\exists\, x, y : \; x \in H_{i+1}, x \notin C_{i+1}^+; \; y \in C_{i+1} \tag{43}$$

By these assumptions on x and y, $y \underset{C}{\succ} x$ because y is in an earlier front than x in complete decomposition ($y \in C_{i+1}, x \notin C_{i+1}^+$). Because of mutual exclusivity of the Pareto fronts

$$C_i^+ \cap C_{i+1} = \emptyset. \tag{44}$$

By (41),(43), and (44),

$$y \notin H_i^+. \tag{45}$$

By the assumption and Postulate 1, $y \underset{H}{\succ} x$.

However, this is a contradiction since $x \in H_{i+1}$ by assumption, and only solutions in H_i^+ can strongly dominate x in helpers. Thus x cannot simultaneously be in H_{i+1} and not in C_{i+1}^+. By (41) and the definition of the Pareto fronts, $y \overset{\succ}{_H} x$.

$$\nexists\, x \in H_{i+1}, x \notin C_{i+1}^+ \tag{46}$$

$$\therefore\ H_{i+1} \subseteq C_{i+1}^+ \tag{47}$$

Since $H_{i+1}^+ = H_i^+ \cup H_{i+1}$, and by (41), $H_{i+1}^+ \subseteq C_{i+1}^+$

7.3 The Job Shop Scheduling Problem (JSSP)

Please refer to Chapter 4 for a description of the JSSP.

7.4 Experiments

When the main objective is represented explicitly, such as in helpers, the best known solution during a run of the GA is never strongly dominated by other solutions – the best solution is always Pareto efficient. Based on Theorem B, the same solution is also Pareto efficient in pure decomposition. With this knowledge, we hypothesize that complete decomposition may lead to better on average performance compared to helpers for many problem instances due to a signal-to-noise argument. In helpers, when a new best fitness within a given helper is found (signal), the algorithm rewards that solution regardless of what other simultaneous negative fitness decrements (noise) have occurred in other parts of the objective values. However, if a new best fitness occurs in a solution in a non-helper component of the objective but occurs simultaneously with larger offsetting fitness decrements in one or more helpers, the solution may go unrewarded by the helper algorithm. Since complete decomposition covers all components of the objective but does not contain overlap of any objective components, it may benefit from being able to identify and reward these fitness improvements. However, in complete decomposition there is less explicit emphasis on the main objective which could hamper performance. Experiment 1 explores which

of these two forces, signal-to-noise improvements and emphasis on the main objective, are most important in the JSSP instances studied.

Making wise choices regarding multi-objectivization can be difficult as there are more than a few things to balance in the process. Experiment 2 studies the factors involved in determining good decomposition sizes. To accomplish this task, new JSSP instances are generated and tested using a variety of decomposition heuristics and decomposition balances. In both Experiments 1 and 2, there were common design decisions to provide sharper comparisons of the features in our hypotheses.

Solution representation was accomplished using a priority list of operations associated with jobs. Each job number appears in the solution string once for each operation associated with the job. The first appearance of a job number thus implies the priority for the first operation for the job. Unlike a representation that assigns unique identifiers to each operation, this representation never allows operations later in a job to have a higher priority over operations earlier in the job. Thus the technological constraints are implicitly maintained in such a representation. '0,0,1,1,1,2,1,0,2,2,0,2' is a sample string of this permutation based representation with three jobs and four machines. The first 0 in the string indicates that the highest priority operation to schedule is the first operation in job 0. The 0 in the second position in the string indicates that the second operation associated with job 0 is next most important, and so on.

To provide consistency with previous works on the JSSP using multi-objectivization, we followed experimental decisions made in Jensen (2004) and Lochtefeld and Ciarallo (2011b) when possible. In all experiments NSGA-II was used to perform the multi-objective search procedures. It is possible that other MOEAs could obtain better results than NSGA-II, but a relative comparison of performance of different MOEAs on multi-objectivization was not the focus of the experiments here. Common parameter settings and operators were set as follows. A population size of 100 was used and 100 new candidate solutions were created in each generation.

The number of decompositions per run was always set to the number of jobs, *n*. The number of fitness evaluations was set to 200*nm* with a minimum number of fitness evaluations of 20,000. Binary tournament selection was used to select parent solutions from the population. Generalized Order Crossover (GOX) was used to generate new candidate solutions. GOX takes a substring from one parent and inserts it into another parent (Bierwirth, 1995). As its name implies, the order in which operations appears is used to determine how the spliced solution is repaired to feasibility. Position Based Mutation (PBM) was applied to all new solutions after they were generated using crossover but before fitness evaluation. PBM removes a random operation from one location in the solution string and inserts the operation at another location in the solution string (Bierwirth, 1995). Both GOX and PBM generate solution strings that have a valid priority list so no additional repair operator was required. The GT schedule builder (Giffler and Thompson, 1960) was used to build active schedules. Only one schedule was built for each solution even though the GT builder can create more than one active schedule for some priority strings. The main objective value and all decomposed objective values were calculated from the solution's built schedule.

Previous objective decompositions on the JSSP were accomplished by dividing jobs into various new objectives. The total flowtime of a job was the fitness contribution of the job on its associated decomposed objectives. Job flowtimes are the sum of all delays and process times associated with each operation. As a result, we examine decompositions here based on operations rather than jobs. We define a given operation's contribution to fitness as the operation's process time plus the immediately preceding delay between the start of the operation and the end of the previous operation in the job.

7.4.1 Experiment 1 – Helpers versus Complete Decomposition

The performance of complete decomposition was compared with the performance of helpers in order to determine which method works best for the JSSP instances studied. The same set of problem instances as those studied in Lochtefeld and Ciarallo (2011b) were examined for this experiment. As much as possible, all settings were consistent with those in Lochtefeld and Ciarallo (2011b) to provide a direct comparison for the SJF methods. In the SJF cases, all operations associated with a particular job i^* are assigned to one objective. For SJF helpers, the remaining objective is assigned all fitness components (for all jobs i) of the original problem's objective. For SJF complete, the remaining objective is assigned only operations (for jobs $i \neq i^*$) that were not assigned to the first objective. As the sequencing of helpers has been shown to be important, the SJF methods assign decompositions such that the decomposition associated with the shortest job is used first, the decomposition associated with the next shortest job is the second decomposition to be used, and so forth.

For the random operation cases a random set of operations equal in number to the size of a job was assigned to one objective. The remaining operations for complete decompositions or all operations for helper decompositions were assigned to the other objective. For each run, the number of decompositions used was equal to the number of jobs, similar to the SJF cases. Also similar to the SJF cases, once a random operation was assigned to the smaller segment, it was no longer used in the smaller segment. That is, the random assignment of operations to the smaller segment was accomplished without replacement across all of the possible decompositions.

Table 20 displays the results of Experiment 1. The first column indicates the problem instance studied along with its problem size. The best known solution column indicates the best known total flowtime value for the problem instance. Previous best values were reported in Lochtefeld and Ciarallo (2011b). Values in bold in the best known column indicate a new best known value was found during the course of this experiment. Percentages formatted in bold are

the best value for all four cases – the combination of the two types and decomposition methods. Percentage values that are underlined in the SJF columns indicate the best value compared within those SJF cases. Similarly underlined values in the random operation columns indicate the best comparative performance on those two setting combinations. Values marked with a '*' indicate that the underlined result is statistically closer to the best known solution than its associated counterpart. Statistical tests were performed with the student T-test using an $\alpha = 0.05$ level of confidence.

Table 20 – Comparison of complete decomposition and helpers. The values reported for each of the decomposition methods is the percentage difference of the average best found solution with the best known solution for the instance.

Instance	Best Known Flowtime	SJF Complete	SJF Helper	RandOp Complete	RandOp Helper
la01 (10x5)	4832	**2.1411**	2.1555	3.4707	3.3745
la02 (10x5)	4459	3.0867	**3.0564**	3.4734	3.5391
la06 (15x5)	8694	4.8597	**4.5902**	6.0599*	6.5843
la07 (15x5)	8174	5.3374	**5.0951**	6.8332	6.8161
la11 (20x5)	**14629**	**5.8232***	6.2131	6.1966*	6.7863
la12 (20x5)	**12325**	**6.4003**	6.7021	6.8294*	7.6585
la16 (10x10)	7393	4.4149	**4.1825**	5.8713	5.4429*
la17 (10x10)	6555	**3.1170***	3.6355	3.7350	3.5738
la21 (15x10)	**12861**	**5.0353**	4.4117	5.9785	5.8981
la22 (15x10)	**12017**	5.3011	**5.1629**	6.2976	6.0521
la26 (20x10)	**20104**	6.2749	**5.2795**	6.5397	6.4301
la27 (20x10)	20764	5.3394	**5.0803**	6.7920	6.6868
la31 (30x10)	**38764**	**5.9649**	5.3412	5.9722*	6.4491
la32 (30x10)	42189	**4.6715**	4.9145	6.2236	6.3686
la36 (15x15)	17073	**4.6790***	5.1243	5.4398	5.1639
la37 (15x15)	17886	**4.8783**	5.1341	5.6079	5.4076
ft10 (10x10)	7501	6.5792	**6.5508**	8.5735	8.6663
ft20 (20x5)	**13977**	10.1600	**8.8453**	10.6951	10.6266
swv01 (20x10)	20688	9.9884	**9.9787**	13.1822	14.0302
swv02 (20x10)	21682	**8.5603**	8.6076	11.3743	11.7735
swv06 (20x15)	**28849**	7.3251	**7.0523**	9.8854*	10.5360
swv07 (20x15)	**27130**	**7.6628**	7.8790	10.9132	11.0785
swv11 (50x10)	108842	**8.6261**	8.9859	10.3086*	12.0560
swv12 (50x10)	109128	**8.8313**	9.4014	10.2541*	11.6673
Average	-	6.0862	**5.9742**	7.3545	7.6111

To verify correct implementation of the methods, we compared the results in Table 20 from the SJF helper cases to the corresponding data reported in Lochtefeld and Ciarallo (2011b). The

values here were very similar to those reported previously, often having variation only after the second or third significant digit. As established in a similar experiment where random sequencing of jobs in the successive decompositions during a run was compared with SJF sequencing in Lochtefeld and Ciarallo (2011b), multi-objectivization using the SJF sequenced decompositions outperforms multi-objectivization using decompositions sequenced with random operations in the helper objective. Within the SJF decompositions, SJF helpers performed slightly better than SJF complete in the overall average (5.97 to 6.09, respectively), although the SJF helper performance was never statistically better on any problem instance. In contrast to this, there were 3 problem instances out of 24 on which SJF complete was statistically better than SJF helpers.

In the random operation decompositions, multi-objectivization using complete decomposition on average outperformed multi-objectivization using helpers (7.35 to 7.61). This is further supported in these results by the fact that for multi-objectivization using random operation sequencing and complete decompositions 7 of these 24 instances had statistically lower average percentages differences from the best known solution when compared to multi-objectivization using random operations in helpers. In comparison, across the same cases, there was only one instance where multi-objectivization using random operation helpers had a statistically lower percentage from the best known solution.

There are several important observations based on these results. This experiment focused exclusively on imbalanced decompositions (decompositions that assign significantly unequal numbers of operations to the different objectives). In the SJF cases, the SJF helper approach performed slightly better than the SJF complete approach in the grand average. In the random operation case, the complete decompositions performed slightly better on average. We believe this somewhat contradictory result is due to the fact that there are conflicting forces in these comparisons. Imbalanced decompositions tend to undermine the natural dominance of complete decomposition over helpers.

Based on our analytical results from Section 7.3, we expect that complete decompositions are at least as effective as helper decompositions, all other factors being equal. Additionally, we hypothesize that in the absence of significant insight into the problem instance, signal-to-noise ratios will be best improved when working with balanced decompositions, as improvements in one segment of a decomposition would more often be found with a paired fitness decrement in the remaining part of the solution in other segments. Finally, we know that effective sequencing of objectives can improve performance when appropriate insight is available. This led to our design for Experiment 2.

7.4.2 Experiment 2 – Exploring heuristic strength and decomposition size

The signal-to-noise principle indicates that it is best to isolate fitness decrements (noise) from fitness improvements (signal). Given no knowledge of the problem structure, balanced divisions of the objective space do the best job of exploiting this principle, since it requires deep knowledge of a problem to determine which components have an intrinsic need for fitness improvements. A heuristic such as SJF, which represents a richer understanding of problem structure, may be more effective when combined with an unequal division of the objective as this unequal division places focus on the most important signal. SJF should be effective in determining which signal is more important and thus isolation of signal-to-noise can be accomplished in an imbalanced way. In this experiment we test how to effectively match the strength of a heuristic with the size of highly effective decompositions. To accomplish this task, two decompositions are tested on a range of problems with varying amounts of structure to be exploited.

Asymmetries in a problem are often the reason that heuristics are effective. For example, a greedy method of constructing a solution will typically be no better than a pure random method if all the selection choices are identical. Various levels of asymmetries in a problem are examined in this experiment in order to determine the effectiveness of balanced and imbalanced sized

decompositions as they relate to decomposition strength based on problem knowledge. We examine two decomposition types. Randomly assigning operations to segments in a decomposition exploits no knowledge of the problem structure. Alternately, segments in a decomposition built by assigning operations in the same job to a given segment uses information pertaining to the structure of the problem.

The classical ft10 and ft20 problems and other problems based on these with similar dimensions: 10x10 - la16, la17, la18, la19, and la20; 20x5 - la11, la12, la13, la14, and la15 are examined. Since 10x10 problems contain exactly 10 jobs and 10 machines, we consider these problems to be neutral in terms of the importance of machines and the importance of jobs. In the 10x10 problems, every job may be processing an operation simultaneously in a feasible schedule. The 20x5 problems contain 5 machines and as a result no more than 5 jobs can be processed simultaneously in a feasible schedule even though there are 20 jobs to process. When optimizing for flowtime in the 20x5 problems, it is very likely that finishing some jobs early in the schedule achieves the goal of minimizing flowtime. Because jobs are in high competition for the same machines, there is a flowtime advantage for having some jobs finish before others. In the 10x10 problems, there is less, but still significant, competition for machines. This equal ratio of jobs and machines make it less advantageous to have some jobs finish early in the schedule as parallel processing of jobs throughout the schedule can lead to overall low flowtimes.

We conjecture that the first level of asymmetry that a job-based decomposition can focus on effectively is the balance of number of machines and jobs. When there are more jobs than machines, placing emphasis on certain jobs likely produces improved results over random decompositions by operation. The second level of asymmetry that may be important is variation in the total of the process times of all operations associated with a given job - the total job process time. If there is large variation in the total job process time, then the order in which jobs finish is

important since it is well known that total flowtime is reduced if a long job is delayed by a short job rather than vice-versa.

7.4.2.1 Generating instances with varying asymmetries

The SJF heuristic works well on instances that have jobs with highly variable process times as the variable process times contain information that the heuristic can exploit. If all of the process times for the jobs in an instance are identical, the SJF method is very similar to a randomly sequenced set of decompositions. Most process times in test problems are generated using random draws from uniform distributions as suggested by Hall and Posner (Hall and Posner, 2001) and other previous researchers. Using uniform distributions for individual operation times results in an implicit level of asymmetry in a problem due to the random draws. In order to study how the strength of a heuristic affects performance of decompositions with various levels of balance, we examine instances with different levels of variability in job process times. Considering that the uniform distribution generates a wide range of operation processing time values with equal likelihood, we considered this the upper limit to the asymmetry of instances to study.

With the upper limit of asymmetry established by the uniform distribution, we use previously established instances to generate new and related instances with less asymmetry. Using an algorithm based on the pseudo code in Figure 49, new instances were generated from the 12 base instances la11 through la20, ft10, and ft20. The algorithm adjusts individual process times associated with operations up or down such that the average process time for all operations remains the same and the range of process times does not increase. This algorithm does not adjust the precedence constraints in the instance and as a result the generated instances will still have many of the same features as the original instance.

```
procedure ChangeProcessTimesTowardUniformJobProcessTimes( integer array of size [n*m]
processTimes, float percentToUniform)
    integer array of size [n] JobProcessTimes;
    integer jobProcessTime ← 0
    integer totalProcessTime ← 0
    for (i ← 1 to n)     // calculate the JobProcessTimes for each job
        jobProcessTime ← 0
        for (k ← 1 to m)
              jobProcessTime ← jobProcessTime + processTimes[n][m];
        end for
        JobProcessTimes[n] ← jobProcessTime
        totalProcessTime ← totalProcessTime + jobProcessTime
    end for
    integer Target ← totalProcessTime / n
    integer steps ← 0; integer maxSteps ← 0;     boolean adjustUpward ← true;
    integer Omin ← the minimum process time of all operations in the data set
    integer Omax ← the maximum process time of all operations in the data set
    for (i = 1 to n) {     // adjust each job
        steps ← 0
        adjustUpward ← true
        int maxSteps ← (Target – JobProcessTimes[i]) * percentToUniform
        if (maxSteps < 0)
            adjustUpward ← false
            maxSteps ← maxSteps * -1
        end if
        while (steps < maxSteps) //adjust until enough modifications have been made
            for (k ← 1 to m)              //adjust individual operations one by one
                  if (adjustUpward = true and P(O(i,k)) < Omax)
                      processTimes[i][k] ← processTimes[i][k] + 1
                      steps ← steps + 1
                  else if (GetProcessTimeOfOperation (i,k) > Omin)
                      processTimes[i][k] ← processTimes[i][k] - 1
                      steps ← steps + 1
                  end else if
                  if (steps >= maxSteps)
                      break out of the for loop
                  end if
            end for
        end while
    end for
end procedure
```

Figure 49 – Pseudo code for instance transformation to uniform job process times

We examined the level of variability in total job process times for the original and newly created instances. Table 21 shows the standard deviation of these total job process times. For the 0% change cases, the instances are unmodified and possess their original job process time

variability. As the change to uniform increases, observe that the standard deviation decreases a proportional amount until, in the '100' instances, all variability is driven out of the job process time. Heuristics such as SJF that process in a greedy fashion based upon differing job process times will likely be less effective as the 'to uniform' parameter is increased.

Table 21 – Standard deviation of total job process times for the original instances and their derived instances

	% change to uniform total Job process Time					
Instance	**0**	**25**	**50**	**75**	**100**	**Average**
ft10 (10x10)	88.0	66.2	44.1	22.2	0.0	44.1
la16 (10x10)	103.7	78.1	52.0	26.1	0.0	52.0
la17 (10x10)	88.1	66.6	44.2	22.4	0.0	44.3
la18 (10x10)	84.2	63.5	42.3	21.3	0.0	42.3
la19 (10x10)	54.0	41.0	27.4	13.9	0.0	27.3
la20 (10x10)	104.3	78.3	52.2	26.2	0.0	52.2
ft20 (20x5)	66.9	50.4	33.5	16.9	0.0	33.5
la11 (20x5)	69.4	52.4	34.8	17.7	0.0	34.9
la12 (20x5)	80.7	60.9	40.5	20.4	0.0	40.5
la13 (20x5)	71.1	53.6	35.7	18.1	0.0	35.7
la14 (20x5)	75.0	56.6	37.6	19.0	0.0	37.7
la15 (20x5)	64.0	48.4	32.2	16.4	0.0	32.2
Average (10x10)	87.1	65.6	43.7	22.0	0.0	43.7
Average (20x5)	71.2	53.7	35.7	18.1	0.0	35.7

Driving variability out of process times may adversely affect the difficulty of the problems. We examined the variability of the operation times in order to determine if significant convergence of operation process times was occurring in the newly generated problems. Table 22 displays the standard deviation for all operation times in the 60 instances studied. The table is structured in pairwise comparisons since these problems start with identical base operating times for the 100 operations. All standard deviation values decrease across the row. In order to keep the range of operation times the same but adjust the job process times, some variability must be driven out of the data. The 10x10 instances received fewer adjustments than the 20x5 instances as noted by the higher standard deviation values across the non-zero columns. With a range of process times from 1 to 99 in these problems and a mean process time of 50 which remained

unchanged by the transformation, standard deviation values range from 28.0 to 22.0 indicating significant variability remains in the datasets generated.

Table 22 – Standard deviation of operation process times for the original instances and their derived instances

	% change to uniform total Job process Time					
Instance	**0**	**25**	**50**	**75**	**100**	**Average**
ft10 (10x10)	27.3	26.7	26.3	25.9	25.8	26.5
ft20 (20x5)	27.3	25.9	24.8	24.1	23.8	25.2
la16 (10x10)	26.5	25.7	25.0	24.5	24.3	25.4
la11 (20x5)	26.5	24.9	23.5	22.5	22.1	23.9
la17 (10x10)	27.5	26.9	26.4	26.0	25.8	26.7
la12 (20x5)	27.5	25.4	23.8	22.6	22.0	24.3
la18 (10x10)	25.8	25.3	24.8	24.5	24.3	25.1
la13 (20x5)	25.8	24.1	22.9	22.0	21.8	23.3
la19 (10x10)	28.0	27.8	27.7	27.6	27.4	27.8
la14 (20x5)	28.0	26.3	24.9	24.1	23.7	25.4
la20 (10x10)	27.6	26.8	26.1	25.7	25.5	26.5
la15 (20x5)	27.6	26.3	25.3	24.6	24.2	25.6
Average (10x10)	27.1	26.5	26.1	25.7	25.5	26.2
Average (20x5)	27.1	25.5	24.2	23.3	22.9	24.6

7.4.2.2 Evaluating Heuristic Strength versus Decomposition Size

We suspect that balanced decompositions are best used when a strong heuristic is not present. Since a given decomposition may give poor results, we follow the principle of swapping decompositions multiple times throughout the run. Because decomposition size is known to make a difference on search quality (Lochtefeld and Ciarallo, 2011b), we assign 10%, 30% and 50% of operations in one segment-tuple and the remainder of operations to the other segment-tuple. To maintain exact coverage of the main objective, operations appear exactly once in a given decomposition.

Three types of decompositions are examined. The most basic decomposition, random operation, randomly assigns operations to segments. Each operation is given the same probability of being selected for a given segment. This decomposition does not take into account problem structure defined by how operations are assigned to jobs. The second decomposition analyzed, random job, assigns operations grouped by job to a segment. As in random operation, random job

uses an equal probability of selecting any job. The third decomposition, random job weighted by process time, assigns short jobs more frequently to the same, smaller segment. The probability of selecting a job x for the smaller segment is the inverse of the process time for job x divided by the inverse of the total process time for all jobs that have not already been selected. This assigns a higher probability for shorter jobs to appear in the smaller segment. As a result of being placed in the smaller segment, jobs should gain more emphasis on fitness improvements relative to the remaining jobs in the larger segment. The previous experiment and our analysis from Section 7.3 provide evidence that, on average, complete decomposition achieves better results than helpers for the JSSP. Therefore, we evaluate only complete decompositions in this experiment. Table 23 summarizes the factors and levels used in this experiment.

Table 23 – Factors and Levels for Experiment 2

Factor	Levels
Base Problem Instances	ft10, ft20, la11, la12, la13, la14, la15, la16, la17, la18, la19, la20
Percent of Adjustment (PA) of Base Instances to Identical Total Job Process Times.	0%, 25%, 50%, 75%, 100%
Decomposition Balance (DB)	10%, 30%, 50%
Decomposition Assignment Method (DM)	Random Operation (ByOp), Random Job (ByJob), Random Job with Weighted Probabilities (ByJobWeighted)

Each factor-level combination was run for 200 replications in order to test for statistical significance between mean performances. Several of the base problem instances, la13, 1a14, la15, la18, la19, and la20, and all of the modified problem instances did not have established best known total flowtime values. To fairly compare relative performance of the methods between instances with established and instances with yet to be established best flowtimes, we compare the average performances against the best found values in this analysis. Table 24 displays the best found values in all experimental replications run during this experiment. For the unmodified la17 problem, in the column marked 0, a new best total flowtime was found and is indicated in bold.

The best known total flowtimes for the ft10, ft20, la11, la12, and la16 unmodified instances, are reported in Table 20. At least one major trend is apparent. The best total flowtime within an instance appears to increase with an increase in the percentage to uniform process times. The trend for larger flowtimes is related to how active schedules are built. The average operation time remains relatively unchanged through the transformation to uniform job processing times. However, when variability is driven out of operation times, it becomes more difficult to 'pack' a schedule as there are fewer "long" operations and fewer "short" operations. The 20x5 problems have a larger increase in total flowtimes which corresponds to the evidence in Table 22 that indicates the transformation more strongly reduced operation time variability in the 20x5 problems.

Table 24 – Best Values Found in All Experimental Replications

	Percentage of change toward identical total job process time				
Instance	**0% (unmodified)**	**25%**	**50%**	**75%**	**100%**
ft10 (10x10)	7606	7744	7802	7806	7827
ft20 (20x5)	14533	14803	14815	15368	15440
la16 (10x10)	7393	7540	7505	7491	7500
la11 (20x5)	14746	14962	15187	15462	15567
la17 (10x10)	**6542**	6597	6607	6609	6542
la12 (20x5)	12457	12759	12966	13241	13332
la18 (10x10)	7034	7097	7097	7034	7070
la13 (20x5)	13969	14164	14593	14812	14943
la19 (10x10)	7226	7286	7226	7272	7315
la14 (20x5)	15104	15324	15516	15465	15371
la20 (10x10)	7394	7425	7407	7451	7513
la15 (20x5)	14901	15255	15488	15667	15704
Average (10x10)	7199.1	7281.5	7274.0	7277.1	7294.5
Average (20x5)	14285.0	14544.5	14760.8	15002.5	15059.5

With the best flowtime rising with increased adjustments to the job process times, one might expect that the instances are becoming more difficult as additional adjustments are made. However, an examination of the average performance of the GA for all methods, instances, and decomposition balances against the adjustment percentage reveals the opposite as indicated in Figure 50. Algorithms on the 10x10 and 20x5 instances appear to be more consistently effective

as the adjustment level increases. The downward slopes of the least-squares linear fit line indicates this trend. Furthermore, observe that the linear fit line for the 20x5 instances has a greater negative slope indicating that the adjustments made the problems in that category "easier" in comparison to the 10x10 adjustments. Since the 20x5 instances received more adjustments (because of more jobs, each with fewer operations), we conclude that the new instances became easier with increasing adjustments toward identical job process times due to the lower variability of operation times. While flowtime increases due to fewer opportunities to tightly pack the schedule, problem difficulty decreases due to fewer productive trade-offs in the sequencing of operations. Swapping two operations with very similar or identical process times will have little impact on flowtime and as a result, the problem tends to become 'easier' with increased similitude of operation process times.

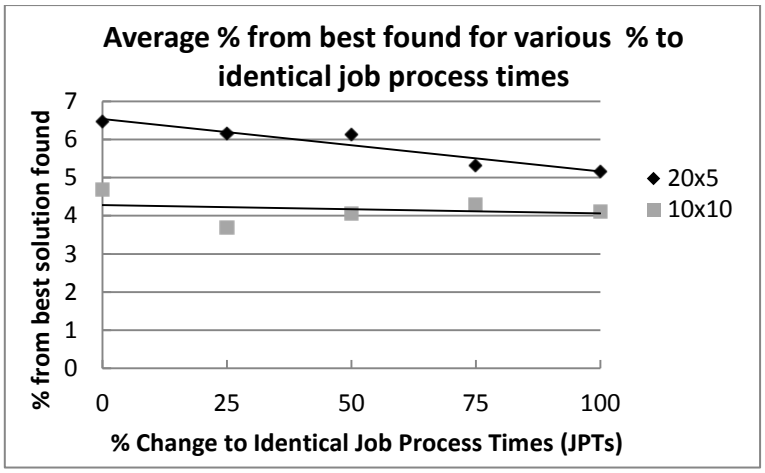


Figure 50 – Average percentage from the best found solution for all cases by the percentage change to identical job process times

We analyzed the results using ANOVA. The analysis was based on these main factors: decomposition balance, decomposition method, and adjustment to identical job process times; and a blocking factor: problem instance. Two-factor and three-factor interactions between decomposition balance, decomposition method, and adjustment to identical job process times were also included in the model. A screenshot of the ANOVA model output in the JMP software

is attached in Appendix B. The initial ANOVA model had a R^2 value of 0.413. While this value is relatively low due to highly variable performance between replications, clear patterns existed in the data and many factors were significant. The model also showed evidence of heteroscedastic variance when plotting the residual values versus predicted values. We applied the 'best' suggested Box-Cox transformation in the JMP software. This transformation improved the uniformity of variance somewhat and did not change the significance of any p-values in any of the analyses. Therefore we assumed the original model is sufficient to draw conclusions from and the results here are reported on the initial, untransformed ANOVA model which is more straightforward to interpret. Table 25 shows the effects tests for the model. Note that all of the main effects as well as 2 of the two-factor interaction effects are significant.

Table 25 – Effects tests for the ANOVA model

Source	DF	Sum of Squares	F Ratio	p-value
Base Instance	11	276,479.74	6636.2	< 0.0001*
Percent Adjustment (PA) To Identical JPTs	4	11,104.26	733.96	< 0.0001*
Decomposition Method (DM)	2	31.95	4.2177	0.0147*
Decomposition Balance (DB)	2	48.41	6.3908	0.0017*
DM * PA	8	72.65	2.3976	0.0139*
DM * DB	4	51.03	3.3687	0.0092*
DB * PA	8	36.05	1.1898	0.3005
DM * DB * PA	16	50.15	0.8276	0.6550

* - effect is significant at the $\alpha = 0.05$ level.

As Table 25 indicates, the variation across the base instances accounts for the largest portion of variation in the model, highlighting the need to include the base instance factor as a blocking factor. We examined the pairwise t-test comparisons for the significant effects to understand the nature of the instances and the performance of the methods. For all of the pairwise t-tests reported here, groups are significantly different at the $\alpha = 0.05$ level based on a $t = 1.95999$ critical value. Observe the pairwise t-tests reported in Table 26. The ft** instances appear to have more varied performance than then la** instances. This is due to the fact that the precedence constraints in the ft** instances are distributed in a way that creates higher machine density. For

instance in the ft10 instance, the first operation in each job always requires 1 of only 3 machines even though there are 10 total machines. As a result, it is more difficult to build schedules where many machines are in use simultaneously. Also, the 20x5 problem instances demonstrated worse performance compared to their 10x10 counterparts. Consider the nature of the problems: the 20x5 instances are more highly constrained as the number of jobs outnumbers the number of machines at a 4 to 1 ratio.

Table 26 – Pairwise t-tests (base instance). The ANOVA identified the overall performance of each base instance to be significantly different from each of the other base instances.

Instance	Groups	Least Sq. Mean
ft20 (20x5)	A	8.4868
ft10 (10x10)	B	7.2573
la15 (20x5)	C	6.1501
la12 (20x5)	D	5.2919
la14 (20x5)	E	5.1601
la13 (20x5)	F	5.0160
la11 (20x5)	F	4.9807
la16 (10x10)	G	4.3803
la19 (10x10)	H	3.9361
la17 (10x10)	I	3.4978
la18 (10x10)	J	3.2253
la20 (10x10)	K	2.7391

Table 27 indicates the pairwise comparison of the various levels of percent adjustment to identical job process times. In general as adjustments are made to the problem the problem appears to have a lower predicted average percentage difference from the best found solution which supports the conclusions made regarding Figure 50. As the instances are adjusted in a way that further modifies operation times to be less variable, it becomes relatively easier for the GA to find solutions that are relatively closer to the best found solution.

Table 27 – Pairwise t-tests (PA).The ANOVA identified each level of PA to be significantly different from each of the other levels, as shown by the groups identified.

Level	Groups	Least Sq. Mean
0%	A	5.5809
50%	B	5.0933
25%	C	4.9247
75%	D	4.8052
100%	E	4.6467

Examining the performance of the decomposition methods in Table 28, observe that the By Job heuristic gives the best performance, although the paired difference between job and Job Weighted was not statistically significant. We expected the heuristic By Job Weighted, inspired by SJF, to give relatively better performance than the simple by job heuristic. Overall, however, this was not the case. The By Operation decomposition heuristic gives the worst overall average performance of all three heuristics: it was identified as statistically worse than the By Job heuristic, although the difference with By Job Weighted was not statistically significant.

Table 28 – Pairwise t-tests (DM). Statistically significant differences are labeled with different group letters.

Level	Groups	Least Sq. Mean
By Operation	A	5.0305
By Job Weighted	A B	5.0115
By Job	B	4.9884

Table 29 indicates the pairwise grouping of various decomposition balances. The 10% balance is the most successful followed by the 30% balance and the 50% balance. The 10% balance has a statistically lower average percent from best found solution compared to the 50% balance.

Table 29 – Pairwise t-tests (DB). Statistically significant differences are labeled with different group letters.

Level	Groups	Least Sq. Mean
50%	A	5.0352
30%	A B	5.0118
10%	B	4.9834

Two second-order interactions were statistically significant in the ANOVA model. The interaction effect between the decomposition method and the percent adjustment to identical job process times was significant. Pairwise t-tests supporting this interaction are displayed in Table 30. As the base instances are adjusted to uniform process times, loss of variability in operation times allows the order of operations in many jobs to be switched without causing major impacts to total flowtime. We theorize this generates less between-job conflicts and, as a result, decompositions that randomly select operations in building decompositions have less internal conflict when variability is removed from the process times. Since these decompositions have less internal conflict, their internal signal (fitness improvement) is less frequently paired with simultaneous noise (fitness decrements). Thus, the By Operation decompositions become more competitive as the instances are modified toward identical job process times. Also note that the only statistical difference between the By Job decompositions and the By Job Weighted decompositions appears in the unmodified instances where the random draw differences between the methods are as different as possible. This indicates that, as we would expect, as job times become identical the decomposition methods converge to the same behavior.

The last, and probably most thought-provoking, pairwise t-tests of significant two-factor interactions in the model are indicated in

Prior to the analysis, we expected the strongest heuristic to be the random method inspired by SJF and that it would perform particularly well on imbalanced decompositions and imbalanced problems. We expected this random method to be best because it increases the probability that critical signal is visited early and often. This was not the case as the By Job Weighted decomposition with the strongest split had the third-worst performance and the heuristic worked relatively worse on the imbalanced problems. This counter-intuitive result gives rise to the question: which phenomena are controlling the By Job Weighted performance? We believe there are three major factors at work here. These factors are coverage, sequencing important

decomposed objectives early, and placing the most useful operations into decomposed objectives. We believe the interaction between these principles explains the results in our experiments.

Table 31. In the By Operation and the By Job decompositions, performance improves as split balance decreases. Further note the spread of competitiveness of the By Operation to By Job decompositions. While the two of the By Operation decompositions are the poorest heuristics, the 10% By Operation decomposition is strong (achieving the second best predicted performance). We theorize this is due to internal versus external conflict in the decompositions. In By Job decompositions, there is little conflict within a given smaller decomposition since there is little conflict within a given job – the conflict in the JSSP arises in the interaction between jobs and machines. As more random jobs or random operations are added to a segment, it becomes more difficult to find fitness improvements because of the conflict within the decomposition. Because each job has many operations, depending on the problem instance, we believe that a threshold of interactions is reached making the 30% and 50% decompositions ineffective at separating the signal in short operations from the noise of many other operations. Because the By Operation decompositions randomly select any operation across all jobs and operations may conflict between jobs, the probability of internal conflict within a given decomposition increases as more operations are added to a decomposition.

Table 30 – Pairwise t-tests for the interaction (DM*PA). Statistically significant differences are labeled with different group letters.

Level	Group	Least Sq. Mean
By Operation, 0%	A	5.6411
By Job Weighted, 0%	A	5.5919
By Job, 0%	B	5.5095
By Operation, 50%	C	5.1406
By Job, 50%	D	5.0744
By Job Weighted, 50%	D	5.0651
By Operation, 25%	E	4.9508
By Job Weighted, 25%	E	4.9167
By Job 25%	E	4.9065
By Job Weighted 75%	F	4.8137
By Job 75%	F	4.8052
By Operation, 75%	F	4.7966
By Job Weighted , 100%	G	4.6699
By Job, 100%	G	4.6466
By Operation, 100%	G	4.6235

Prior to the analysis, we expected the strongest heuristic to be the random method inspired by SJF and that it would perform particularly well on imbalanced decompositions and imbalanced problems. We expected this random method to be best because it increases the probability that critical signal is visited early and often. This was not the case as the By Job Weighted decomposition with the strongest split had the third-worst performance and the heuristic worked relatively worse on the imbalanced problems. This counter-intuitive result gives rise to the question: which phenomena are controlling the By Job Weighted performance? We believe there are three major factors at work here. These factors are coverage, sequencing important decomposed objectives early, and placing the most useful operations into decomposed objectives. We believe the interaction between these principles explains the results in our experiments.

Table 31 – Pairwise t-tests for the interaction (DM*DB).Statistically significant differences are labeled with different group letters.

Level	Group					Least Sq. Mean
By Operation, 50%	A					5.0763
By Operation, 30%	A	B				5.0459
By Job Weighted, 10%		B	C			5.0233
By Job Weighted, 50%		B	C			5.0222
By Job, 50%		B	C	D		5.0071
By Job, 30%		B	C	D	E	5.0005
By Job Weighted, 30%			C	D	E	4.9889
By Operation, 10%				D	E	4.9693
By Job, 10%					E	4.9577

The By Job Weighted heuristic should randomly place short jobs in a small segment early more often as these jobs are most often selected by the heuristic. We believed this sequencing would result in improved performance because it places more emphasis on the operations that are more likely to be important, and the increased probability of appearance implies that short jobs will often appear in the smaller segment early in the optimization. However, because the experiment was designed to make random selections with replacement when building decompositions, we failed to account for coverage in the decomposition heuristic. Previous research shows that the probability of finding a better solution increases immediately after the switching of decompositions. Further we know in principle that visiting multiple different decompositions is useful. The By Job Weighted decomposition method has degraded performance when only 10% of jobs are included in a given decomposition because as job process times are imbalanced, the method will visit certain decompositions more than others. A weighted selection process can result in three types of events: a switch of decompositions can bring in a new objective to the smaller segment that has never appeared in the smaller segment, a switch can bring in a new objective to the smaller segment that is different than the current but that has appeared in the smaller segment in an earlier decomposition, or a switch may yield no change in the objectives appearing in the segments. The latter two cases become more common as

one segment gets smaller than the other decomposition. In the 10% split for By Job Weighted decompositions, the degraded performance is due to these two cases becoming more common. We know from previous research on the JSSP that the odds of finding a new best solution are improved when a switch from one decomposition to another takes place (Lochtefeld and Ciarallo, 2011b). This assumes sufficient time is given for building blocks to develop between switches. Reducing the number of switches causes these odds to decrease. Furthermore, revisiting similar areas likely causes similar building blocks to be rebuilt. This has an opportunity cost as the algorithm could be building more effective blocks in other areas.

Our attempt to create a "fair" comparison of the SJF heuristic with the random heuristics resulted in the creation of a vastly weaker heuristic than the original SJF method due to a lack of coverage. While it is important to have an appropriate sequence as shown in previous works, it is also just as, or more, important to maintain some level of coverage so that the smaller segment visits many of the operations. While areas of a problem within one decomposition are always explicitly or implicitly covered through the definition of complete decompositions, giving focus to varied areas of the problem in unbalanced decompositions is important. This is because the difficulty of the problem means that many areas likely need to have improvements in fitness in order to achieve good on-average results.

7.5 Chapter Conclusion

The differences between complete decomposition and helper-based decompositions were investigated. Specifically, we examined how Pareto dominance relationships for solutions in both methods relate to each other. Our formal analysis indicates that a given Pareto efficient frontier in complete decomposition always contains all of the Pareto efficient solutions in a similar helper decomposition. This result is extended to draw conclusions about additional fronts beyond the initial Pareto efficient frontier. An empirical study of pure decomposition and associated helper methods shows that, on average, pure decomposition is superior for the instances studied. These

two facts, the analytical and empirical evidence here, in conjunction with previous evidence provided on the TSP that complete decomposition is superior, leads us to the conclusion that the use of complete decompositions is generally superior to the use of helpers. While helper-decompositions provide explicit emphasis on the main objective, they do not identify some possibly good fitness improvements and are resultantly less effective than complete decomposition.

The final experiment studied the JSSP using segmentation strategies based on various heuristics in order to determine if heuristic strength was a driving factor in determining good decomposition sizes. In the experiment factors affecting decomposition size were identified and an additional three principles governing the choice of decompositions and their sequences were supported. Appropriate decomposition size is driven by: an effective minimal decomposition size (Lochtefeld and Ciarallo, 2011c) heuristic strength, and conflict levels within a decomposition. Good decomposition heuristics should maintain a minimum effective size while attempting to both minimize within-decomposition conflict and exploit known problem asymmetries. The three principles governing decomposition choice are: focus through balance, focus by order of appearance, and adequate coverage. There is value in creating appropriate focus on an objective based on its value through selection of decompositions that are balanced or imbalanced. Imbalanced decompositions place emphasis on the smaller segments in a decomposition and should be used when great value can be concentrated in a small segment. There is also value in picking appropriate sequences of decompositions such that more productive focus is placed earlier in the search. Lastly there is value in adequate coverage: most or all portions of the main objective should be given focus at some point in the search process.

Chapter 8 Overall Summary and Conclusions

This work and supporting research investigated the underlying principles behind multi-objectivization in GAs. This chapter is organized in two major sections. First a summary of the work accomplished to date is provided. Then recommendations for future research are discussed.

8.1 Summary of Work

Chapter 1 provided an introduction to the work through an analogy to optimization in dog breeding and a preview of results and experiments in the remaining chapters. The scope of the research was discussed briefly, followed by deeper investigations in the remaining chapters.

Chapter 2 discusses general concepts related to Genetic Algorithms and provided a framework for classification of GAs by their diversity approaches. The diversity classification framework builds on prior work to classify GAs. Chapter 2 also outlines the differences between a pure diversity approach and multi-objectivization. The chapter provided the foundations of GAs in order to discuss the more complex multi-objective methods introduced in Chapter 3.

Chapter 3 discusses background literature related to several topic areas. Firstly, Multi-Objective Evolutionary Algorithms (MOEAs) are discussed. Pareto dominance concepts are introduced. Then, multi-objectivization research to date was reviewed. We concluded that previous research to date contains few guiding principles to assist the analyst in making design decisions regarding the actual implementation of this new optimization technique.

Chapter 4, a study on the Job Shop Scheduling Problem (JSSP), examines a multi-objectivization technique called helper-objectives. This work identifies that helper size and helper sequence are important to an optimization. Further the chapter debunks previous thoughts regarding the effect that size of the non-dominated front has on the quality of results. Lastly, this

work uncovers that helper swaps increase the frequency of the algorithm in finding new best solutions. We theorize that such swaps mitigates some negative non-linear interactions (epistasis) in the problem.

Chapter 5 introduces and studies an abstract problem that is not NP hard called the Tunable Objectives Problem (TOP). TOP is a minimization problem with a binary search landscape and an additive fitness function. TOP can be used to model a single or multiple objectives at various layers of the problem and can introduce various local optima and forms of conflict. From the study of the TOP model several principles are identified as they are applied to helper-objectives. One of these principles rests on a signal-to-noise ratio argument. Fitness improvements (signal) can be isolated from simultaneously manifested fitness decrements (noise) through the use of multi-objectivization.

Chapter 6 investigated a multi-objectivization technique called Multi-Objectivization via Segmentation (MOS) and how it functions on the Traveling Salesman Problem. Various different decompositions were developed based upon the signal-to-noise argument and empirical evidence showed their advantage over previous methods. Insight into why previous methods were effective was also provided. Effective decompositions fell into two basic categories: spatially oriented decompositions and inter-/intra-neighborhood decompositions. Based on experimental evidence and theory, a more general method was developed called Multi-Objectivization via Progressive Segmentation (MOPS). MOPS uses progressively more aggressive segmentation in order to improve search performance. MOPS was shown to have robust performance across a variety of computational budgets.

Finally, Chapter 7 examined the difference between two prevalent decomposition types: helper-objectives and complete decomposition (MOS). Through analysis and empirical study we explored the differences between the methods. The relationship between Pareto frontiers in the two decomposition types was detailed analytically. Empirical evidence shows that there is little

difference between helper-objectives and complete decompositions on imbalanced decompositions. On more balanced decompositions, complete decomposition appears to be superior. Based on this evidence we ultimately recommend pursuing additional research on complete decomposition rather than on helper-objectives. Lastly in Chapter 7 the relationship between decomposition size, heuristic strength, and decomposition balance was explored. From this investigation we conclude that strong heuristics are best paired with imbalanced decompositions and that coverage of various components of the objective is important.

In this investigation of multi-objectivization we identified that decompositions:

- should be sequenced according to their contribution to fitness (importance) (Chapter 4),
- should be selected by their likelihood to be unhampered by local optima that are present in the main objective (Chapter 5),
- should be selected based on improving SNRs that are internal to the problem (Chapter 5),
- should be selected such that concurrent helpers have complementary properties (Chapter 5),
- should be selected for their alignment with the main objective (Chapter 5), and
- should be sized according to conflict levels, strength of heuristics, and critical minimum size (Chapter 7).

8.2 Contributions

In summary the major contributions of this work were the following:

- Developed a diversity based classification scheme for GAs
- Identified numerous principles of multi-objectivization
- Revealed two major methods in which multi-objectivization gains improved results
- Extended and developed an abstract problem instance for testing of optimization techniques
- Developed a new method for generating instances of the JSSP
- Identified better decomposition techniques for use on the JSSP and TSP
- Developed a robust multi-objectivization technique

8.3 Further Research

This document is concluded with future research recommendations. While this list is not exhaustive, it is clear that additional development of multi-objectivization techniques is likely to provide additional insight.

A diversity classification scheme was proposed but many algorithms were not categorized. The categorization of a larger number of algorithms may lead to required extensions and modifications to the classification system. Such a classification of algorithms would highlight the strengths and weaknesses of such a classification approach.

The techniques of multi-objectivization are still in their infancy. As a result many possible research directions are still open for exploration. Understanding the fundamentals behind this new technique is critical to the broader body of optimization knowledge. This research has expanded that body of knowledge, but by no means do we think this is the end, but rather more of a beginning. There are numerous possible research patches in the quilt of multi-objectivization, of which we recommend several threads.

The research on multi-objectivization to date has mostly focused on additive objective functions where the parts were easily identified and decoded. Other problems such as the JSSP for makespan, the H-IFF problem, and the set covering problem likely require indirect decompositions as their objective functions are more difficult to decompose. Study of such problems may reveal broader understanding of how this optimization technique can apply to a larger range of problems.

Research on the TOP model can be used to address other multi-objective and single objective problems. Given TOPs ability to be highly customized, other researchers may find it beneficial to study and even extend the TOP model. We envision more examination at the types and characteristics of local optima that can be overcome through genetic techniques. We also see the need to study the level and type of conflict found in real problems to provide a grounding for future study using TOP. Also we recommend a study of how TOP can be applied to model problems that are not naturally binary in their representation. .

Currently, multi-objectivization has been performed by algorithms (MOEAs) designed for problems that are naturally multi-objective. It is unclear if these MOEAs are best suited for multi-

objectivization as they treat objectives in equal emphasis and may not be as elegant as a more general approach. No comparisons between various MOEAs has been accomplished to date. A general approach to determining a good algorithm could start with a comparison of MOEAs and then be extended to cover more general algorithms that are currently undiscovered.

Lastly, while multi-objectivization to date has focused on discrete decompositions that are swapped in at distinct periods of time, a more continuous approach could be evaluated. This continuous approach would slowly change the objectives used in order to provide more consistent performance for the optimization. Considering the importance of breaking epistasis, such an approach may need to be more or less aggressive in the speed and phasing of new objectives.

References

Aarts, E., Korst, J., Michiels, W., 2005. Simulated Annealing, in Search Methodologies. Springer, 187-210.

Abbass, H.A., Deb, K., 2003. Searching under Multi-evolutionary Pressures, in Evolutionary Multi-Criterion Optimization. Springer, Berlin / Heidelberg, 391-404.

Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. Mgmt. Sci., 391-401.

Aickelin, U., 2002. An Indirect Genetic Algorithm for Set Covering Problems. J. Oper. Res. Soc. 53, 1118-1126.

Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J., 2006. The Traveling Salesman Problem: A Computational Study. Princeton Univ Pr, New Jersey.

Arora, J.S., 2004. Introduction to Optimum Design. Elsevier Academic Press, San Diego, CA.

Bäck, T., 1996. Evolutionary Algorithms in Theory and Practice. Oxford University Press.

Barnett, L., 1998. Ruggedness and neutrality: The NKp family of fitness landscapes. Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life, 18–27.

Beasley, D., Martin, R.R., Bull, D.R., 1993. An overview of genetic algorithms: Part 1. Fundamentals. Univ. Comput. 15, 58-58.

Beasley, J.E., 1990. OR Library: Distributing Test Problems by Electronic Mail. J. Oper. Res. Soc. 41, 1069-1072 http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Belton, V., Stewart, T.J., 2002. Multiple Criteria Decision Analysis :An Integrated Approach. Kluwer Academic Publishers, Boston MA.

Bhattacharya, M., 2004. An informed operator approach to tackle diversity constraints in evolutionary search. Information Technology: Coding and Computing, 2004. Proceedings of ITCC 2004. 2, 326-330.

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. OR Spectrum 17, 87-92.

Blazewicz, J., Domschke, W., Pesch, E., 1996. The job shop scheduling problem: Conventional and new solution techniques. Eur. J. Oper. Res. 93, 1-33.

Bouyssou, D., 2006. Evaluation and Decision Models with Multiple Criteria: Stepping Stones for the Analyst. Springer, New York NY.

Brockhoff, D., Friedrich, T., Hebbinghaus, N., Klein, C., Neumann, F., Zitzler, E., 2007. Do additional objectives make a problem harder? Proceedings of the 9th annual conference on genetic and evolutionary computation: GECCO '07, 765-772.

Cantu-Paz, E., 1998. A survey of parallel genetic algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis 10, 141-171.

Carlson, B.P., Hougen, D.F., 2010. Phenotype feedback genetic algorithm operators for heuristic encoding of snakes within hypercubes. Proceedings of the 12th annual conference on genetic and evolutionary computation: GECCO '10, 791-798.

Cheng, R., Gen, M., Tsujimura, Y., 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Comput. Ind. Eng. 36, 343-364.

Cheng, R., Gen, M., Tsujimura, Y., 1996. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. Comput. Ind. Eng. 30, 983-997.

Clemen, R.T., Reilly, T., 2001. Making Hard Decisions with DecisionTools, 2nd rev. ed. Duxbury/Thomson Learning, Pacific Grove, CA.

Codenotti, B., Manzini, G., Margara, L., Resta, G., 1993. Global strategies for augmenting the efficiency of TSP heuristics, in Algorithms and Data Structures. Springer Berlin / Heidelberg, 253-264.

Coello Coello, C.A., Lamont, G.B., Van Veldhuisen, D.A., 2007. Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd ed. Springer, New York.

Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J., 2001. PESA-II: Region-based selection in evolutionary multiobjective optimization. , 283–290.

Corne, D.W., Knowles, J.D., Oates, M.J., 2000. The pareto envelope-based selection algorithm for multiobjective optimisation. Lecture Notes in Computer Science 1917, 839-848.

Davidor, Y., 1991. Epistasis variance: A viewpoint on GA-hardness, in Foundations of Genetic Algorithms - 1. Morgan Kaufmann, San Mateo, CA, pp. 23-35.

De Jong, K.A., 1975. Analysis of the behavior of a class of genetic adaptive systems. University of Michigan, College of Engineering Technical Reports .

De Jong, K.A., Potter, M.A., Spears, W.M., 1997. Using problem generators to explore the effects of epistasis. Proceedings of the Seventh International Conference on Genetic Algorithms, 338-345.

De Jong, K.A., Spears, W.M., 1989. Using genetic algorithms to solve NP-complete problems. Proceedings of the Third International Conference on Genetic Algorithms, 124-132.

Deb, K., Goldberg, D.E., 1989. An investigation of niche and species formation in genetic function optimization. Proceedings of the 3rd International Conference on Genetic Algorithms , 42-50.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6, 182-197.

Deb, K., Saha, A., 2010. Finding multiple solutions for multimodal optimization problems using a multi-objective evolutionary approach. Proceedings of the 12th annual conference on genetic and evolutionary computation GECCO 10, 447-454.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evol. Comput. 6, 182-197.

Della Cioppa, A., De Stefano, C., Marcelli, A., 2007. Where Are the Niches? Dynamic Fitness Sharing. IEEE Trans. on Evol. Comput. 11, 453-465.

Della Cioppa, A., De Stefano, C., Marcelli, A., 2004. On the role of population size and niche radius in fitness sharing. IEEE Trans. Evol. Comput. 8, 580-592.

Dorndorf, U., Pesch, E., 1995. Evolution based learning in a job shop scheduling environment. Comput. and Oper. Res. 22, 25-40.

Eiben, A.E., Hinterding, R., Michalewicz, Z., 1999. Parameter Control in Evolutionary Algorithms. IEEE Trans. on Evol. Comput. 3.

Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J., 2007. Parameter control in evolutionary algorithms, in Parameter Setting in Evolutionary Algorithms. Springer, Berlin / Heidelberg, 19-46.

Eiben, A.E., Schippers, C.A., 1998. On evolutionary exploration and exploitation. Fundamenta Informaticae 35, 35-50.

Eiben, A.E., Smith, J.E., 2003. Introduction to Evolutionary Computing. Springer.

Eshelman, L.J., 1991. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. in Foundations of Genetic Algorithms 1, 265–283.

Eshelman, L.J., Caruana, R.A., Schaffer, J.D., 1989. Biases in the crossover landscape. Proceedings of the third international conference on Genetic algorithms, 10-19.

Figueira, J., Greco, S., Ehrgott, M., 2005. Multiple Criteria Decision Analysis: State of the Art Surveys. Springer Verlag.

Fisher, M.L., 2004. The Lagrangian Relaxation Method for Solving Integer Programming Problems. Mgmt. Sci. 50, 1861-1871.

Fontana, W., Stadler, P.F., Bornberg-Bauer, E.G., Griesmacher, T., Hofacker, I.L., Tacker, M., Tarazona, P., Weinberger, E.D., Schuster, P., 1993. RNA folding and combinatory landscapes. Phys. Rev. E. 47, 2083-2099.

Forrest, S., Mitchell, M., 1993. Relative building-block fitness and the building-block hypothesis, in Foundations of Genetic Algorthms - 2. Morgan Kaufmann, San Francisco, pp. 109-126.

Friedrich, T., Hebbinghaus, N., Neumann, F., 2007. Rigorous analyses of simple diversity mechanisms. Proceedings of the 9th annual conference on genetic and evolutionary computation: GECCO'07 .

Garey, M.R., Johnson, D.S., Sethi, R., 1976. The Complexity of Flowshop and Jobshop Scheduling. Math. of Oper. Res. 1, 117-129.

Geard, N., Wiles, J., Hallinan, J., Tonkes, B., Skellett, B., 2002. A comparison of neutral landscapes–nk, nkp and nkq. Proceedings of the IEEE Congress on Evolutionary Computation.

Gibbs, M.S., Dandy, G.C., Maier, H.R., 2008. A genetic algorithm calibration method based on convergence due to genetic drift. Inf. Sci. .

Giffler, B., Thompson, G., 1960. Algorithms for solving production-scheduling problems. Oper. Res. 487-503.

Glover, F., Laguna, M., 1997. Tabu Search. Springer.

Goldberg, D.E., 1989a. Genetic algorithms and Walsh functions: Part I, a gentle introduction. Complex Systems 3, 129-152.

Goldberg, D.E., 1989b. Genetic algorithms and Walsh functions: Part II, deception and its analysis. Complex Systems 3, 153-171.

Goldberg, D.E., 1989c. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Crawfordsville, IN.

Goldberg, D.E., Korb, B., Deb, K., 1989. Messy genetic algorithms: Motivation, analysis, and first results. Complex systems 3, 493-530.

Grefenstette, J.J., 1993. Deception considered harmful, in Foundations of Genetic Algorithms - 2. Morgan Kaufmann, San Francisco, pp. 75-91.

Greiner, D., Emperador, J.M., Winter, G., Galván, B., 2007. Improving Computational Mechanics Optimum Design Using Helper Objectives: An Application in Frame Bar Structures. Proceedings of the Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science 4403, 575-589.

Hall, N.G., Posner, M.E., 2001. Generating Experimental Data for Computational Testing with Machine Scheduling Applications. Oper. Res. 49, 854-865.

Hamming, R.W., 1950. Error detecting and error correcting codes. Bell Sys. Tech. J. 29, 147-160.

Handl, J., Lovell, S., Knowles, J., 2008a. Multiobjectivization by Decomposition of Scalar Cost Functions. Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN-X, 31-40.

Handl, J., Lovell, S.C., Knowles, J., 2008b. Investigations into the Effect of Multiobjectivization in Protein Structure Prediction. Proceedings of the 10th International Conference on Parallel Problem Solving From Nature: PPSN-X, 702-711.

Hiremath, C., Hill, R.R. Improving genetic algorithm convergence using problem structure and domain knowledge in multidimensional knapsack problems. Int. J. of Oper. Res. 1, 145-159

Holland, J.H., 1992. Adaptation in Natural and Artificial Systems. MIT Press Cambridge, MA, USA, 1st Edition: 1975 The University of Michigan Press, Ann Arbor.

Hopp, W.J., Spearman, M.L., 2001. Factory Physics, 2nd Ed. ed. Irwin/McGraw-Hill, Boston, MA.

Jahne, M., Li, X., Branke, J., 2009. Evolutionary algorithms and multi-objectivization for the travelling salesman problem. Proceedings of the 11th annual conference on genetic and evolutionary computation: GECCO'09, 595-602.

Jelasity, M., 1998. UEGO, an abstract niching technique for global optimization. Proceedings of the 5th International Conference on Parallel Problem Solving from Nature: PPSN V, Lecture notes in computer science, 378-387.

Jensen, M.T., 2004. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. J. of Math. Mod. and Alg. 3, 323-347.

Jensen, M.T., 2003a. Guiding Single-Objective Optimization Using Multi-objective Methods, in Applications of Evolutionary Computing. Springer, 91-98.

Jensen, M.T., 2003b. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. IEEE Trans. on Evol. Comput. 7, 503-515.

Jensen, M.T., 2001. Robust and flexible scheduling with evolutionary computation. Dissertation. University of Aarhus, Denmark

Kauffman, S.A., Weinberger, E.D., 1989. The NK model of rugged fitness landscapes and its application to maturation of the immune response. J. Theor. Biol. 141, 211-245.

Kirkpatrick, S., 1984. Optimization by simulated annealing: Quantitative studies. J. of Stat. Phys. 34, 975-986.

Kirkwood, C.W., 1996. Strategic Decision Making. Wadsworth Publ. Co.

Knowles, J., Corne, D., 1999. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. Evol. Comput. 1, 98–105.

Knowles, J.D., Watson, R.A., Corne, D.W., 2001. Reducing local optima in single-objective problems by multi-objectivization. Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization: EMO'01. Lecture Notes in Computer Science, 269-283.

Knox, J., 1994. Tabu search performance on the symmetric traveling salesman problem. Comput. Oper. Res. 21, 867-876.

Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S., 1999. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. Artif. Intell. Rev. 13, 129-170.

Lawrence, S., 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement).

Lloyd, S., 1982. Least squares quantization in PCM. IEEE Trans. Inf. Theory 28, 129-137.

Lochtefeld, D.F., Ciarallo, F.W., 2011a. An Analysis of Decomposition Approaches in Multi-objectivization via Segmentation. J. of Applied Soft Comput. (under review).

Lochtefeld, D.F., Ciarallo, F.W., 2011b. Helper-Objective Optimization Strategies for the Job-Shop Scheduling Problem. J. of Applied Soft Comput. .

Lochtefeld, D.F., Ciarallo, F.W., 2011c. Multi-objectivization via Helper-objectives with the Tunable Objectives Problem (TOP). IEEE Trans. Evol. Comput. (to appear).

Lochtefeld, D.F., Ciarallo, F.W., 2010. Deterministic Helper-Objective Sequence Applied to the Job-Shop Scheduling Problem. Proceedings of the 12th annual conference on genetic and evolutionary computation: GECCO'10, 431-438.

Mahfoud, S.W., 1995. Niching methods for genetic algorithms. Dissertation. University of Illinois at Urbana. Tech report no. 95001.

Mahfoud, S.W., 1992. Crowding and preselection revisited. University of Illinois at Urbana. Tech report no. 92004.

Mattfeld, D.C., 1996. Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for Production Scheduling. Springer Verlag.

Maturana, J., Saubion, F., 2008. A compass to guide genetic algorithms. Proceedings of the 10th International Conference of Parallel Problem Solving from Nature: PPSN X, 256-265.

Mauldin, M.L., 1984. Maintaining Diversity in Genetic Search. Proceedings of the Fourth International Conference on Artificial Intelligence: AAAI-84.

Michalewicz, Z., Fogel, D.B., 2000. How to Solve it: Modern Heuristics. Springer Verlag.

Miller, B.L., Shaw, M.J., 1995. Genetic algorithms with dynamic niche sharing for multimodal function optimization. University of Illinois at Urbana, Tech. report no. 61801.

Mitchell, M., Forrest, S., Holland, J.H., 1992. The royal road for genetic algorithms: Fitness landscapes and GA performance. Proceedings of the First European Conference on Artificial Life, 245-254.

Moscato, P., Cotta, C., Mendes, A., 2004. Memetic Algorithms, in New Optimization Techniques in Engineering. Springer Verlag, 54-85.

Naudts, B., Suys, D., Verschoren, A., 2000. Generalized royal road functions and their epistasis. Comput. and Artificial Intel. 19, 317-334.

Naudts, B., Suys, D., Verschoren, A., 1997. Epistasis as a basic concept in formal landscape analysis. Proceedings of the 7th International Conference on Genetic Algorithms: ICGA'97, 65-72.

Naudts, B., Verschoren, A., 1999. Epistasis and deceptivity. Bulletin of the Belgian Math. Soc. 6, 147-154.

Neumann, F., Wegener, I., 2007. Can Single-Objective Optimization Profit from Multiobjective Optimization? in Multiobjective Problem Solving from Nature: From Concepts to Applications. Springer, 115-130.

Newman, M.E.J., Engelhardt, R., 1998. Effects of selective neutrality on the evolution of molecular species. Proceedings: Biological Sciences 265, 1333-1338.

Oppacher, F., Wineberg, M., 1999. The shifting balance genetic algorithm: Improving the GA in a dynamic environment. Proceedings of the genetic and evolutionary computation conference, 1, 504-510.

Random House: Webster's College Dictionary, 1997. Random House, New York.

Reeves, C.R., Wright, C.C., 1995. Epistasis in genetic algorithms: An experimental design perspective. Proceedings of the 6th International Conference on Genetic Algorithms, 217-224.

Reeves, C., 1999. Predictive measures for problem difficulty. Proceedings of the 1999 Congress on Evolutionary Computation: CEC 99. 1.

Saidi-Mehrabad, M., Fattahi, P., 2007. Flexible job shop scheduling with tabu search algorithms. Inter. J. of Adv. Mfg. Tech. 32, 563-570.

Schaffer, J.D., Mani, M., Eshelman, L., Mathias, K., 1999. The effect of incest prevention on genetic drift. in Foundations of Genetic Algorithms 5, Morgan Kaufmann

Scharnow, J., Tinnefeld, K., Wegener, I., 2004. The analysis of evolutionary algorithms on sorting and shortest paths problems. J. of Math. Model. and Algor. 3, 349-366.

Schraudolph, N.N., Belew, R.K., 1992. Dynamic parameter encoding for genetic algorithms. Mach. Learning 9, 9-21.

Shimodaira, H., 1997. DCGA: A diversity control oriented genetic algorithm. Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence, 367-374.

Shir, O.M., Back, T., 2006. Niche radius adaptation in the CMA-ES niching algorithm. Proceedings of the 9th International Conference of Parallel Problem Solving from Nature: PPSN IX. Lecture Notes in Computer Science, 4193, 142.

Simon, H.A., 1982. Models of Bounded Rationality. MIT Press, Cambridge, Mass.

Sipser, M., 1996. Introduction to the Theory of Computation. PWS Publishing Company, Boston, MA.

Smith, R.E., Forrest, S., Perelson, A.S., 1993. Searching for diverse, cooperative populations with genetic algorithms. Evol. Comput. 1, 127-149.

Starkweather, T., McDaniel, S., Mathias, K., Whitley, D., Whitley, C., 1991. A comparison of genetic sequencing operators. Proceedings of the Fourth International Conference on Genetic Algorithms, 69-76.

Storer, R.H., Wu, S.D., Vaccari, R., 1992. New search spaces for sequencing problems with application to job shop scheduling. Mgmt. Sci. 38, 1495-1509.

Suys, D., Verschoren, A., 1996. Extreme epistasis. Proceedings of the International Conference on Intelligent Technologies in Human-Related Sciences: ITHURS'96, 251-258.

Thierens, D., 2002. Adaptive Mutation Rate Control Schemes in Genetic Algorithms. Proceedings of the 2002 IEEE World Congress on Computational Intelligence: Congress on Evolutionary Computation, 980-985.

Tomassini, M., 2005. Spatially Structured Evolutionary Algorithms. Springer, Berlin / Heidelberg.

Ursem, R.K., 2003. Diversity-guided evolutionary algorithms. Proceedings of the 7th International Conference of Parallel Problem Solving from Nature: PPSN VII. Lecture Notes in Computer Science, 462-474.

Vaessens, R.J.M., Aarts, E.H.L., Lenstra, J.K., 1996. Job shop scheduling by local search. INFORMS Journal on Computing 8, 302-317.

Van Hove, H., Verschoren, A., 1994. On Epistasis. Comput. and Artificial Intel. 271-277.

Weinberger, E., 1990. Correlated and uncorrelated fitness landscapes and how to tell the difference. Biol. Cybern. 63, 325-336.

Weise, T., Niemczyk, S., Skubch, H., Reichle, R., Geihs, K., 2008. A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. Proceedings of the

10th annual conference on genetic and evolutionary computation: GECCO '08, 795-802.

Whitacre, J.M., Pham, T.Q., Sarker, R.A., 2006. Use of statistical outlier detection method in adaptive evolutionary algorithms. Proceedings of the 8th annual conference on genetic and evolutionary computation: GECCO '06 , 1345-1352.

Whitley, L.D., 2001. An overview of evolutionary algorithms: practical issues and common pitfalls. Inf. and Software Tech. 43, 817-831.

Whitley, L.D., 1991. Fundamental principles of deception in genetic search. in Foundations of genetic algorithms 1, 221-241.

Whitley, L.D., 1989. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. Proceedings of the third international conference on genetic algorithms , 116-121.

Wiese, K.C., Glen, E., 2003. A permutation-based genetic algorithm for the RNA folding problem: a critical look at selection strategies, crossover operators, and representation issues. BioSystems 72, 29-41.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1, 67-82.

Wolsey, L.A., 1998. Integer Programming. John Wiley & Sons, New York, NY.

Wong, Y.Y., Lee, K.H., Leung, K.S., Ho, C.W., 2003. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. Soft Computing-A Fusion of Foundations, Methodologies and Applications 7, 506-515.

Wright, M., 2001. Subcost-guided search—experiments with timetabling problems. J. Heuristics 7, 251-260.

Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. Proceedings of the 3rd international conference on Parallel Problem Solving from Nature: PPSN-III , 95–100.

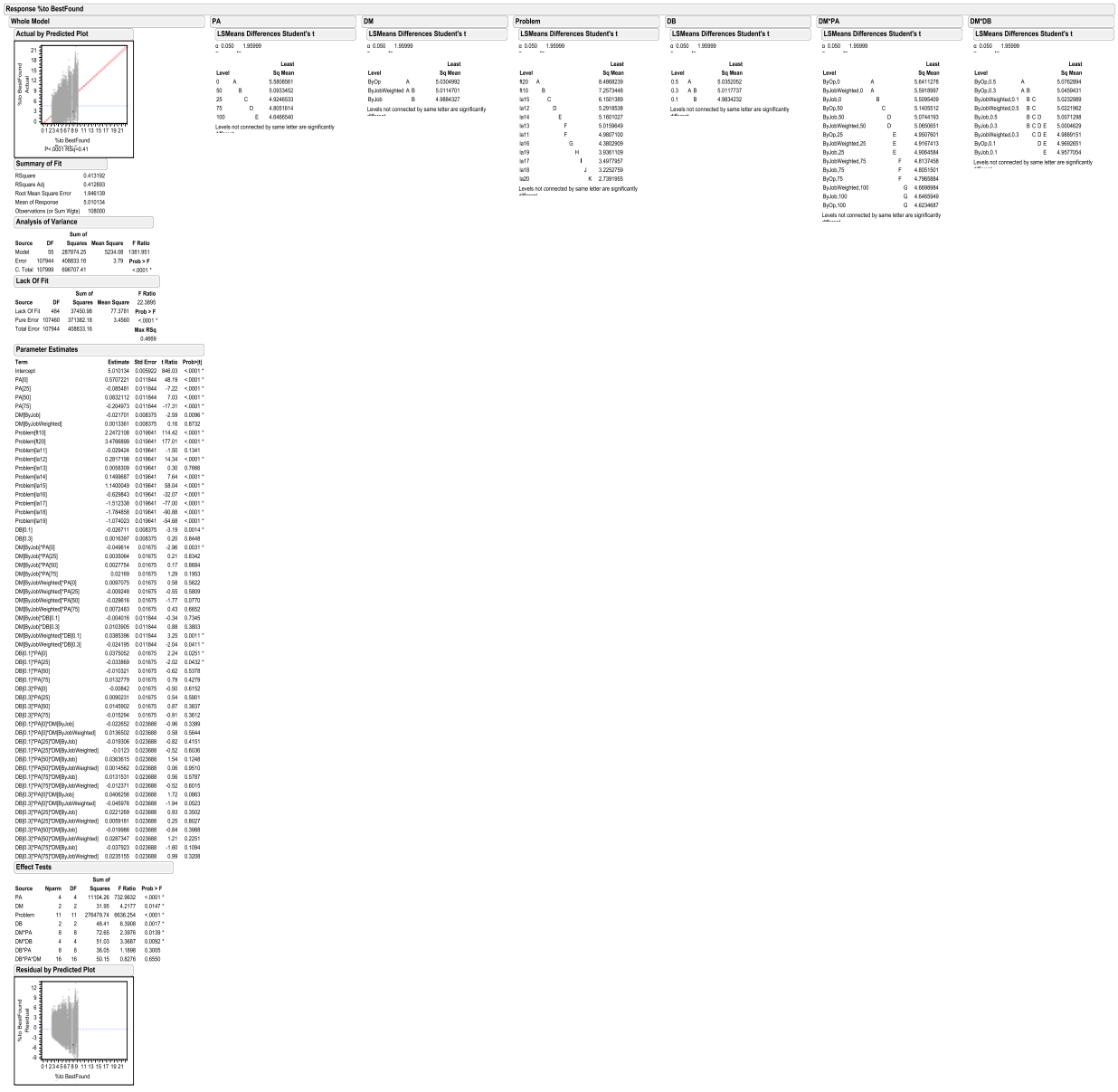
Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. on Evol. Comput. 3, 257-271.

Appendix A – Acronyms

CDSS – Cross-generational Deterministic Survival Strategy
CHC – Cross-generational elitist selection, Heterogeneous recombination and Cataclysmic mutation
CPSS – Cross-generational Probabilistic Survival Strategy
CST – Cross-Section Type
CX – Cycle Crossover
DCGA – Diversity Control oriented Genetic Algorithm
DGEA – Diversity Guided Evolutionary Algorithm
DOE – Design of Experiments
DPE – Dynamic Parameter Encoding
EA – Evolutionary Algorithm
ES – Evolutionary Strategy
EEB – Exploration / Exploitation Balance
EMO – Evolutionary Multi-objective Optimization
EVOD – Expected Value of Distances (decomposition)
FLM – Fitness Landscape Model
GA – Genetic Algorithm
GENITOR – GENetic ImplemenTOR
GOX – Generalized Order (based) Crossover
GT – Giffler-Thompson (schedule builder)
GPX – Generalized Position Crossover
H-IFF – HIerarchical-iF-and-only-iF (problem)
HUX – Half Uniform Crossover
JPH – Jobs Per Helper
JSSP – Job Shop Scheduling Problem
MCDM – Multiple Criteria Decision Making
MOEA – Multi-Objective Evolutionary Algorithm
MOGA – Multiple Objective Genetic Algorithm
MOO – Multiple Objective Optimization
MOS – Multi-Objectivization via Segmentation
MOPS – Multi-Objectivization via Progressive Segmentation
MVD – Multi-objectivization Via Decomposition
NFL – No Free Lunch
NNROD – Nearest Neighbor Ratio of Distances (decomposition)
NSGA-II – Non-dominated Sorting Genetic Algorithm version II
OBM – Order Based Mutation
OX – Order (based) Crossover
PAES – Pareto Archived Evolution Strategy
PBM – Position Based Mutation
PESA-II – Pareto Envelope based Selection Algorithm version II
PMX – Partially Mapped Crossover
PPX – Precedence Preserving Crossover
PRAM – Probabilistic Rule-driven Adaptive Model
SBM – Swap Based Mutation
SHC – Single-objective Hill Climber
SJF – Shortest Job First
SNR – Signal to Noise Ratio
SO – Subordinate Objective

SOG – Subordinate Objective Group
SOO – Single-Objective Optimization
SPEA2 – Strength Pareto Evolutionary Algorithm version 2
SSGA – Steady State Genetic Algorithm
SSSP – Single Source Shortest Path (problem)
TOP – Tunable Objectives Model
TSP – Traveling Salesman Problem
UEGO – Universal Evolutionary Global Optimizer
UX – Uniform Crossover
VRP – Vehicle Routing Problem

Appendix B – JMP ANOVA model supporting experiment 2 of Chapter 7



Response %to BestFound

Whole Model

Actual by Predicted Plot

Summary of Fit

RSquare	0.413192
RSquare Adj	0.412893
Root Mean Square Error	1.946139
Mean of Response	5.010134
Observations (or Sum Wgts)	108000

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	55	267874.25	5234.08	1381.951
Error	107944	408833.16	3.79	Prob > F
C. Total	107999	676707.41		<.0001 *

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	484	37450.98	77.3781	22.3695
Pure Error	107460	371382.18	3.4560	Prob > F
Total Error	107944	408833.16		<.0001 *
				Max RSq
				0.4659

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	5.010134	0.005922	846.03	< .0001 *
PA[0]	0.5707221	0.011844	48.19	< .0001 *
PA[25]	-0.085481	0.011844	-7.22	< .0001 *
PA[50]	0.0832112	0.011844	7.03	< .0001 *
PA[75]	-0.204073	0.011844	-17.31	< .0001 *
DM[ByJob]	-0.021701	0.008375	-2.59	0.0096 *
DM[ByJobWeighted]	0.0013361	0.008375	0.16	0.8732
Problem[lt10]	2.2472106	0.019641	114.42	< .0001 *
Problem[lt20]	3.4798856	0.019641	177.01	< .0001 *
Problem[lat11]	-0.029424	0.019641	-1.50	0.1341
Problem[lat12]	0.2817186	0.019641	14.34	< .0001 *
Problem[lat13]	0.0058309	0.019641	0.30	0.7668
Problem[lat14]	0.1499687	0.019641	7.64	< .0001 *
Problem[lat15]	1.1400049	0.019641	58.04	< .0001 *
Problem[lat16]	-0.629843	0.019641	-32.07	< .0001 *
Problem[lat17]	-1.512338	0.019641	-77.00	< .0001 *
Problem[lat18]	-1.784658	0.019641	-90.88	< .0001 *
Problem[lat19]	-1.074023	0.019641	-54.68	< .0001 *
DB[0.1]	-0.026711	0.008375	-3.19	0.0014 *
DB[0.3]	0.0016397	0.008375	0.20	0.8448
DM[ByJob]*PA[0]	-0.049614	0.01675	-2.96	0.0031 *
DM[ByJob]*PA[25]	0.0035064	0.01675	0.21	0.8342
DM[ByJob]*PA[50]	0.0027754	0.01675	0.17	0.8684
DM[ByJob]*PA[75]	0.02169	0.01675	1.29	0.1953
DM[ByJobWeighted]*PA[0]	0.0097075	0.01675	0.58	0.5622
DM[ByJobWeighted]*PA[25]	-0.009248	0.01675	-0.55	0.5809
DM[ByJobWeighted]*PA[50]	-0.029616	0.01675	-1.77	0.0770
DM[ByJobWeighted]*PA[75]	0.0072483	0.01675	0.43	0.6652
DM[ByJob]*DB[0.1]	-0.004016	0.011844	-0.34	0.7345
DM[ByJob]*DB[0.3]	0.0103605	0.011844	0.88	0.3803
DM[ByJobWeighted]*DB[0.1]	0.0385396	0.011844	3.25	0.0011 *
DM[ByJobWeighted]*DB[0.3]	-0.024195	0.011844	-2.04	0.0411 *
DB[0.1]*PA[0]	0.0375052	0.01675	2.24	0.0251 *
DB[0.1]*PA[25]	-0.033869	0.01675	-2.02	0.0432 *
DB[0.1]*PA[50]	-0.010321	0.01675	-0.62	0.5378
DB[0.1]*PA[75]	0.0132779	0.01675	0.79	0.4279
DB[0.3]*PA[0]	-0.00842	0.01675	-0.50	0.6152
DB[0.3]*PA[25]	0.0090231	0.01675	0.54	0.5901
DB[0.3]*PA[50]	0.0145952	0.01675	0.87	0.3837
DB[0.3]*PA[75]	-0.015256	0.01675	-0.91	0.3612
DB[0.1]*PA[0]*DM[ByJob]	-0.022652	0.023688	-0.96	0.3399
DB[0.1]*PA[0]*DM[ByJobWeighted]	0.0136502	0.023688	0.58	0.5644
DB[0.1]*PA[25]*DM[ByJob]	-0.019306	0.023688	-0.82	0.4151
DB[0.1]*PA[25]*DM[ByJobWeighted]	-0.0123	0.023688	-0.52	0.6036
DB[0.1]*PA[50]*DM[ByJob]	0.0363615	0.023688	1.54	0.1248
DB[0.1]*PA[50]*DM[ByJobWeighted]	0.0014562	0.023688	0.06	0.9510
DB[0.1]*PA[75]*DM[ByJob]	0.0131531	0.023688	0.56	0.5787
DB[0.1]*PA[75]*DM[ByJobWeighted]	-0.012371	0.023688	-0.52	0.6015
DB[0.3]*PA[0]*DM[ByJob]	0.0408256	0.023688	1.72	0.0863
DB[0.3]*PA[0]*DM[ByJobWeighted]	-0.045976	0.023688	-1.94	0.0523
DB[0.3]*PA[25]*DM[ByJob]	0.0221269	0.023688	0.93	0.3502
DB[0.3]*PA[25]*DM[ByJobWeighted]	0.0359181	0.023688	0.25	0.8027
DB[0.3]*PA[50]*DM[ByJob]	-0.019966	0.023688	-0.84	0.3988
DB[0.3]*PA[50]*DM[ByJobWeighted]	0.0287347	0.023688	1.21	0.2251
DB[0.3]*PA[75]*DM[ByJob]	-0.037923	0.023688	-1.60	0.1094
DB[0.3]*PA[75]*DM[ByJobWeighted]	0.0235155	0.023688	0.99	0.3208

Effect Tests

Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
PA	4	4	11104.26	732.9632	< .0001 *
DM	2	2	31.95	4.2177	0.0147 *
Problem	11	11	276479.74	6636.254	< .0001 *
DB	2	2	48.41	6.3908	0.0017 *
DM*PA	8	8	72.65	2.3976	0.0139 *
DM*DB	4	4	51.03	3.3687	0.0092 *
DB*PA	8	8	36.05	1.1898	0.3005
DB*PA*DM	16	16	50.15	0.8276	0.6550

Residual by Predicted Plot

PA

LSMeans Differences Student's t

α 0.050 1.95999

Level		Least Sq Mean
0	A	5.5808561
50	B	5.0933452
25	C	4.9246533
75	D	4.8051614
100	E	4.6466540

Levels not connected by same letter are significantly

DM

LSMeans Differences Student's t

α 0.050 1.95999

Level		Least Sq Mean
ByOp	A	5.0304692
ByJobWeighted	A B	5.0114701
ByJob	B	4.9884327

Levels not connected by same letter are significantly

Problem

LSMeans Differences Student's t

α 0.050 1.95999

Level		Least Sq Mean
lt20	A	8.4868239
lt10	B	7.2573448
lat15	C	6.1501389
lat12	D	5.2918538
lat14	E	5.1601027
lat13	F	5.0159840
lat11	F	4.9807100
lat16	G	4.3802909
lat19	H	3.9361159
lat17	I	3.4977957
lat18	J	3.2252759
la20	K	2.7391955

Levels not connected by same letter are significantly

DB

LSMeans Differences Student's t

α 0.050 1.95999

Level		Least Sq Mean
0.5	A	5.0352052
0.3	A B	5.0117737
0.1	B	4.9834232

Levels not connected by same letter are significantly

DM*PA

LSMeans Differences Student's t

α 0.050 1.95999

Level		Least Sq Mean
ByOp,0	A	5.6411278
ByJobWeighted,0	A	5.5918997
ByJob,0	B	5.5056409
ByOp,50	C	5.1405512
ByJob,50	D	5.0764193
ByJobWeighted,50	D	5.0593051
ByOp,25	E	4.9507601
ByJobWeighted,25	E	4.9167413
ByJob,25	E	4.9064884
ByJobWeighted,75	F	4.8137458
ByJob,75	F	4.8051501
ByOp,75	F	4.7965884
ByJobWeighted,100	G	4.6699884
ByJob,100	G	4.6465949
ByOp,100	G	4.6234687

Levels not connected by same letter are significantly

DM*DB

LSMeans Differences Student's t

α 0.050 1.95999

Level		Least Sq Mean
ByOp,0.5	A	5.0762894
ByOp,0.3	A B	5.0458431
ByJobWeighted,0.1	B C	5.0232989
ByJobWeighted,0.5	B C	5.0221962
ByJob,0.5	B C D	5.0071298
ByJob,0.3	B C D E	5.0004629
ByJobWeighted,0.3	C D E	4.9880151
ByOp,0.1	D E	4.9692651
ByJob,0.1	E	4.9577094

Levels not connected by same letter are significantly