Solving the Multi-objective Quadratic Assignment Problem Using a fast messy Genetic Algorithm

Richard O. Day, Mark P. Kleeman, Gary B. Lamont
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright Patterson Air Force Base, Dayton, OH 45433

Abstract- The multi-objective quadratic assignment problem is an NP-complete problem with a multitude of real-world applications. The specific application addressed in this paper is the minimization of communication flows in a heterogenous mix of unmanned aerial vehicles. Developed is a multi-objective approach to solving the general mQAP for this UAV application. The combinatoric nature of this problem calls for a stochastic search algorithm; moreover, the Multi-Objective fast messy Genetic Algorithm (MOMGA-II) [22] is used for experimentation. Results indicate that much of the Pareto optimal points are found.

1 Introduction

The multi-objective quadratic assignment problem (mQAP) is an NP-complete problem. In this paper the mQAP is mapped to a heterogenous mix of unmanned aerial vehicles (UAVs). Our model concentrates on minimizing communication flow and maximize mission success by positioning UAVs in a selected position within a strict formation. MQAP experiments are conducted using the our multi-objective evolutionary algorithm (MOEA), Multi-Objective Messy Genetic Algorithm - II (MOMGA-II). Solutions are then compared to a deterministic results (were applicable). Section II of the paper covers the problem description in great length. Section III contains a brief discussion of Evolutionary Algorithms (EAs) including the fast messy Genetic Algorithms (fmGAs). Section III also contains a discussion of MOEAs focusing in particular on the the MOMGA-II. Section IV contains design of experiments and section V presents the results and analysis.

2 General QAP Description

Currently, the Air Force uses the unmanned Aerial vehicles (UAVs) Predator and Global Hawk for reconnaissance missions over the field of battle. They have a goal right now for UAVs to perform suppression of enemy air defense (SEAD) missions by the year 2010 with the unmanned combat aerial vehicle (UCAV)[4]. In the future, they are looking to have UAVs, flying in large groups, play a bigger role in the air. One possible scenario is to have a heterogenous group of UAVs flying together to meet an objective. There could be some in the group that are doing reconnaissance and reporting the information to the UCAV, whose goal is to take out a target when it is located by one of the other UAVs. There could also be *fighter* UAVs, whose job is to defend the group of UAVs from enemy aircraft.

In a large heterogenous group, such as this one, your position with respect to the other UAVs is important. For example, it would be best to place fighter UAVs around the outside of the group in order to protect the group as a whole from enemy aircraft. It would also be advantageous to have the reconnaissance planes nearer to the ground in order to allow them to have an unobstructed field of view.

While location in the formation for their particular part of the mission is important, they also need to be in a position where they can communicate effectively with other UAVs. For example, the reconnaissance UAVs need to communicate coordinates to the UCAVs, to enable them to find their target. The fighter UAVs need to communicate with all of the other UAVs when they sense approaching enemy aircraft, so that the group can take evasive action. And UCAVs need to communicate when they have no munitions left. All of these flows of communication can also dictate where the best location in the group may be for each UAV.

This UAV communication and mission success problem is a natural extension of the mQAP. The mQAP comes from the quadratic assignment problem (QAP) and was introduced by Knowles and Corne [13]. The scalar quadratic assignment problem was introduced in 1957 by Koopmans and Beckmann. They used it to model a plant location problem[3]. It is defined where you have a fixed number of locations where each location is a fixed distance apart from one another. You also have the same number of fixed objects that need to be put into each location. Each *object* has a fixed flow to each of the other objects. The goal of the QAP is to find the best placement of the objects into the locations such that the product of the distances and flows are minimized. Mathematically, this is defined in Equation 1.

$$minC(\pi) =_{\pi \in P(n)}^{min} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi_i \pi_j} \qquad (1)$$

where n is the number of objects/locations, a_{ij} is the distance between location i and location j, b_{ij} is the flow from object i to object j, and π_i gives the location of object i in permutation $\pi \in P(n)$ where $P(n)$ is the QAP search space, which is the set of all permutations of $\{1, 2, \ldots, n\}$ [14]. This problem is not only NP-hard and NP-hard to approximate, but is almost intractable. It is generally considered to be impossible to solve optimally any QAP that has 20 instances or more within a reasonable time frame [3, 19].

The mQAP is similar to the scalar QAP, with the ex-

†The views expressed in this article are those of the authors and do not reflect the official policy of the United States Air Force, Department of Defense, or the U.S. Government. The authors also wish to acknowledge the following individuals: Jesse Zydallis for the use of his MOMGA-II code, Todd Hack and Justin Kautz for all the hard work and effort they put forth in supporting these experiments.

ception that there are multiple types of flows coming from each object. For example, the UAVs may use one communication channel for passing reconnaissance information, another channel for target information, and yet another channel for status messages. So the goal is to minimize all the communication flows between the UAVs. The mQAP is defined in mathematical terms in Equations 2 and 3

$$minimize\{C(\pi)\} = \{C^1(\pi), C^2(\pi), \ldots, C^m(\pi)\} \quad (2)$$

$$C^k(\pi) =_{\pi \in P(n)}^{min} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b^k_{\pi_i \pi_j}, k \in 1..m \quad (3)$$

and where n is the number of objects/locations, a_{ij} is the distance between location i and location j, b^k_{ij} is the kth flow from object i to object j, π_i gives the location of object i in permutation $\pi \in P(n)$, and 'minimize' means to obtain the Pareto front [14].

There has been a lot of work done with respect to the fitness landscape of QAP instances. Knowles and Corne [14] have enumerated two measurements, diameter and entropy, that they use as mQAP metrics. They use the diameter of the population as it is defined by Bachelet [1] which is shown in Equation 4:

$$dmm(P) = \frac{\sum_{\pi \in P} \sum_{\mu \in P} dist(\pi, \mu)}{|P|^2} \quad (4)$$

where $dist(\pi, \mu)$ is a distance measurement that measures the smallest number of two-swaps that need to be performed in order to transform one solution, π, into another solution, μ. The distance measure has a range of $[0, n-1]$.

The entropy measurement, which measures the dispersion of the solutions, is shown in Equation 5

$$ent(P) = \frac{-1}{nlogn} \sum_{i=1}^{n} \sum_{j=1}^{n} (\frac{n_{ij}}{|P|} log \frac{n_{ij}}{|P|}) \quad (5)$$

where n_{ij} is a measure of the number of times object i is assigned to the j location in the population.

Many approaches have been tried to solve the QAP. Researchers interested in finding the optimal solution can usually only do so for problems that are of size 20 or less. And even problem sizes of 15 are considered to be difficult [3]. But when it is feasible to find the optimal solution, branch and bound methods are typically used [7, 20, 3]. But since many real-world problems are larger than 20 instances, other methods need to be employed in order to find a good solution in a reasonable amount of time. The use of Ant Colonies has been explored and has been found to do fairly well when compared to other some of the best heuristics available for the QAP [5, 11, 16]. Evolutionary algorithms have also been applied many times [17, 15, 10, 18]. Several researchers have compared the performance of different search methods [21, 9].

The goal of this research is to see how effective a fast messy GA can be at solving the mQAP using a serial version of the MOMGA-II.

3 MOMGA-II Software Design

This section develops an understanding of the MOMGA-II and describes its growth from being a single objective EA (namely the messy GA).

3.1 Multi-Objective Evolutionary Algorithms (MOEAs)

Evolutionary algorithms (EA) include genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. EAs are a class of algorithms that use the concepts of genetics to help enable them to explore the search landscape. In an EA a collection of an individuals traits are known as chromosomes. A group of chromosomes is defined as a population. Chromosomes consist of cells in which various data types can reside; binary, integer, real-valued. These individual chromosome allele values are manipulated by EA operators. Thus, the population moves through generations by using genetic operators such as mutation and recombination. Mutation works by inserting new genetic material into the population by modifying allele values, recombination by transferring preexisting genetic material or allele values between two or more individuals of the population. There are many varieties of genetic operators, each with a different set of parameters that may be modified given a particular EA type [2]. The transition to MOEAs focuses on the phenotype domain with multiple objectives or fitness functions. The genotype domain is essentially the same with each individual chromosome evaluated for each fitness function. In order to understand the structure of the specific MOEA, the MOMGA-II, a single-objective GA referred to as the fast messy GA is discussed.

3.2 Fast Messy GA Structure (fmGA)

In fmGAs, the length is variable. This is because the fmGA allows for over-specification (and under-specification). The reason for the difference is that in a GA, the location of an allele in a chromosome is defined to mean something, but in a fmGA, the allele contains both the value and the location that the value belongs to in the chromosome. So a chromosome can have more than one value listed for the same location. This is called over-specification. The way this is usually resolved is that the first value that is listed for a location is the value used. Under-specified chromosomes don't have all of the locations covered with allele values. In that case, the values are placed onto a master template which has a value assigned for all locations. Then, the template values will replace the missing allele values.

Regarding population, a messy GA is different in that the entire set of possible solutions is represented in the initial population. Since this could be a potentially time consuming task, the fmGA was invented by Goldberg et. al., that only uses a probabilistic subset of all possible solutions in the initial population. Of course, the desire is to generate good building blocks from generation to generation per the Schema Theorem [8].

The operators most commonly used in EAs are crossover and mutation. Crossover is usually the major operator for GAs and mutation is a minor operator. Crossover is done by

picking a point or series of points on two selected chromosomes, usually called parents. Then the values are swapped between each of the parents to create two new children. For an fmGA, this operator is replaced by one called the cut-and-splice operator. It does basically the same thing, except that it picks random points for each of the parents for the cutting, versus the same random point for both chromosomes. This is needed because of the variable chromosome size.

Mutation is where a point in the chromosome is changed based on the mutation probability. This value is usually quite low. It is used to add new allele combinations to the chromosome.

The selection operator is used to determine which individuals in the population will become parents and/or which individuals will advance to the next generation. There are a number of selection operators that can be used such as rank selection, proportional selection, and tournament selection to name a few. Each of these has their merits and are best used when matched closely with other particulars of the problem.

3.3 General Multi-Objective Problem (MOP)

A multi-objective optimization problem (MOP) consists of decision variables, two or more objective functions, and constraints. These three components of an MOP are decision variables., objective function: and constraints. Standard MOP and MOEA definitions and nomenclature can be found in (Coello Coello et al., 2002). Such symbolic formulation includes feasible regions in objective space, feasible solutions, solution dominance and non-dominance, true and approximate Pareto optimal solutions P*/ Pknown and Pareto front PF*/ PFknown, fitness sharing, niche count, sharing function, mating restrictions, ranking and the required evolutionary algorithm characteristics. The goal of a Pareto-based MOEA is convergence of PFknown towards PF*. MOEAs operate on a population of candidate solutions (chromosomes) as opposed to a single solution; therefore, the strength of an MOEA is its ability to uncover multiple nondominated solutions (Pknown). The QAP is the MOP of interest in this paper.

3.4 MOMGA-II

The Multi-Objective Messy Genetic Algorithm - II (MOMGA-II) program is based on the concept of the Building Block Hypothesis (BBH). The MOMGA implements a deterministic process to generate an enumeration of all possible BBs, of a user specified size, for the initial population. This process is referred to as Partially Enumerative Initialization (PEI). Thus, the MOMGA explicitly uses these building blocks in combination to attempt to solve for the optimal solutions in multiobjective problems.

The original messy GA consists of three distinct phases: *Initialization Phase, Primordial Phase, Juxtapositional Phase*. The MOMGA uses these concepts and extends them where necessary to handle $k > 1$ objective functions. In the initialization phase, the MOMGA produces all building blocks of a user specified size.

The primordial phase performs tournament selection on the population and reduces the population size if necessary. The population size is adjusted based on the percentage of "high" fitness BBs that exist. In some cases, the "lower" fitness BBs may be removed from the population to increase this percentage.

In the juxtapositional phase, BBs are combined through the use of a cut and splice recombination operator. Cut and splice is a recombination (crossover) operator used with variable string length chromosomes. The cut and splice operator is used with tournament thresholding selection to generate the next population.

The main bottleneck in the mGA and the MOMGA is the initialization phase. This phase requires the enumeration of every possible BB, of user specified size.

Table 1: **Test Suite used - Knowles and Corne**

Test Name	Instance Category	# of locations	# of flows
KC10-2fl-1uni	Uniform	10	2
KC10-2fl-2uni	Uniform	10	2
KC10-2fl-3uni	Uniform	10	2
KC20-2fl-1uni	Uniform	20	2
KC20-2fl-2uni	Uniform	20	2
KC20-2fl-3uni	Uniform	20	2
KC30-3fl-1uni	Uniform	30	3
KC30-3fl-2uni	Uniform	30	3
KC30-3fl-3uni	Uniform	30	3
KC10-2fl-1rl	Real-like	10	2
KC10-2fl-2rl	Real-like	10	2
KC10-2fl-3rl	Real-like	10	2
KC10-2fl-4rl	Real-like	10	2
KC10-2fl-5rl	Real-like	10	2
KC20-2fl-1rl	Real-like	20	2
KC20-2fl-2rl	Real-like	20	2
KC20-2fl-3rl	Real-like	20	2
KC20-2fl-4rl	Real-like	20	2
KC20-2fl-5rl	Real-like	20	2
KC30-3fl-1rl	Real-like	30	3
KC30-3fl-2rl	Real-like	30	3
KC30-3fl-3rl	Real-like	30	3

A probabilistic approach is used in initializing the population of the fmGA. The approach is referred to as Probabilistically Complete Initialization (PCI) [6]. PCI initializes the population by creating a controlled number of BBs based on the user specified BB size and string length. The fmGA's initial population size is smaller than the mGA (and MOMGA by extension) and grows at a smaller rate as a total enumeration of all BBs of size o is not necessary. These BBs are then "filtered," through a Building Block Filtering (BBF) phase, to probabilistically ensure that all of the desired good BBs from the initial population are retained in the population. The BBF approach effectively reduces the computational bottlenecks encountered with PEI through reducing the initial population size required to ob-

Table 2: **Population sizes for N facilities and locations**

	Population by (N)			# of Generations by (N)		
Era	(10)	(20)	(30)	(10)	(20)	(30)
1	403	401	400	300	100	300
2	413	405	402	20	100	20
3	430	411	405	20	100	20
4	455	419	408	20	100	20
5	491	431	413	20	100	20
6	553	458	431	20	100	20
7	601	464	427	20	100	20
8	685	487	436	20	100	20
9	794	514	447	20	100	20
10	937	546	458	20	100	20

Table 3: **MOMGA-II settings**

Variable	Setting used
GA-type	fast messy GA
Number of eras	10
Population representation	Binary
Number of bits per facility	10 bits
Splice Probablility	1.00
Cut Probability	0.02
Allelic Mutation Probability	0.00
Genic Mutation Probability	0.00
Thresholding	No
Tiebreaking	No

tain "good" statistical results. The fmGA concludes by executing a number of juxtapositional phase generations in which the BBs are recombined to create strings of potentially better fitness.

The MOMGA-II mirrors the fmGA and consists of the following phases: *Initialization, Building Block Filtering, and Juxtapositional*. The MOMGA-II differs from the MOMGA in the Initialization and Primordial phase, which is referred to as the Building Block Filtering phase. The initialization phase of the MOMGA-II uses PCI instead of the PEI implementation used in the MOMGA and randomly creates the initial population.

The application of an MOEA to a class of MOP containing few feasible points creates difficulties that an MOEA must surpass in order to generate any feasible points throughout the search process. A random initialization of an MOEA's population may not generate any feasible points in a constrained MOP. Without any feasible solutions in the population, one must question whether or not the MOEA can even conduct a worthwhile search. In problems where the feasible region is greatly restricted, it may be impossible to create a complete initial population of feasible solutions randomly. Without feasible population members, any MOEA is destined to fail. Feasible population members contain the BBs necessary to generate good solutions. It is possible for an infeasible population member to contain a BB that is also present in a feasible solution. As it is also possible for mutation to generate a feasible population member from a infeasible population member. However,

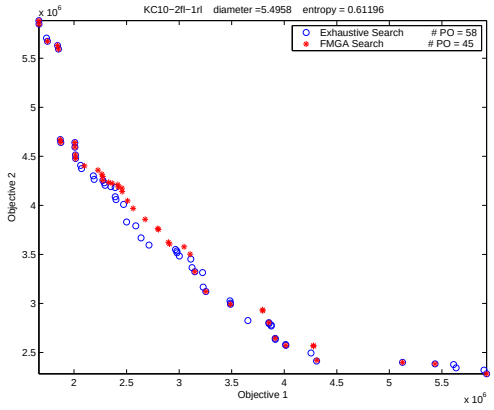

Figure 1: Pareto front found for the KC10-2fl-1rl test instance

typically feasible population members contain BBs that are not present in infeasible population members. EVOPs applied to feasible members tend to yield better results than EVOPs applied to infeasible population members. Therefore, it is critical to initialize and maintain a population of feasible individuals.

4 Design of Experiments and Testing

We wanted to compare our results with other programs that have solved the mQAP. In order to do this, we needed to use some sort of benchmark data set that we could compare. We chose to use the test suite created by Knowles [12]. See table 1 for a detailed parameter description of the test suite problems.

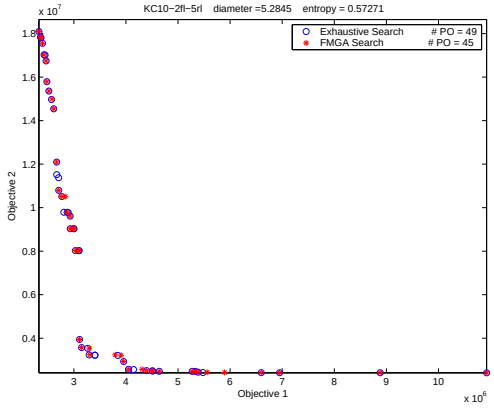

Figure 2: Pareto front found for the KC10-2fl-5rl test instance

The results of the MOMGA-II as applied to the QAP MOP using the default parameter settings for the MOMGA-II are presented in Table 3. The population size and number of generations run in each era for 10 facilities and 10 locations is shown in table 2. For 20 locations and 20 facilities are shown in table 2. And the population for the 30 facilities and 30 locations is shown in table 2. These popu-

Table 4: **Comparison of Results**

	Knowles Results			Our Results		
Test Name	# of pareto points	Diameter	Entropy	# of pareto points	Diameter	Entropy
KC10-2fl-1uni	27	7	0.71	13	5	0.66
KC10-2fl-2uni	4	6	0.39	1	0	0
KC10-2fl-3uni	135	8	0.78	118	6	0.87
KC20-2fl-1uni	80	15	0.828	24	11	0.82
KC20-2fl-2uni	19	14	0.43	538	15	1.48
KC20-2fl-3uni	178	16	0.90	51	12	0.92
KC30-3fl-1uni	705	24	0.97	126	20	0.50
KC30-3fl-2uni	168	22	0.92	58	22	0.64
KC30-3fl-3uni	1257	24	0.96	155	20	0.56
KC10-2fl-1rl	38	8	0.68	44	5	0.69
KC10-2fl-2rl	17	7	0.49	10	5	0.56
KC10-2fl-3rl	58	8	0.62	36	6	0.71
KC10-2fl-4rl	33	8	0.58	34	4	0.53
KC10-2fl-5rl	48	8	0.63	45	6	0.69
KC20-2fl-1rl	541	15	0.63	17	12	0.73
KC20-2fl-2rl	842	14	0.6	12	11	0.76
KC20-2fl-3rl	1587	15	0.66	29	12	0.91
KC20-2fl-4rl	1217	15	0.51	25	10	0.18
KC20-2fl-5rl	966	15	0.56			
KC30-3fl-1rl	1329	24	0.83	191	24	0.79
KC30-3fl-2rl	1924	24	0.86	183	24	0.77
KC30-3fl-3rl	1906	24	0.86			

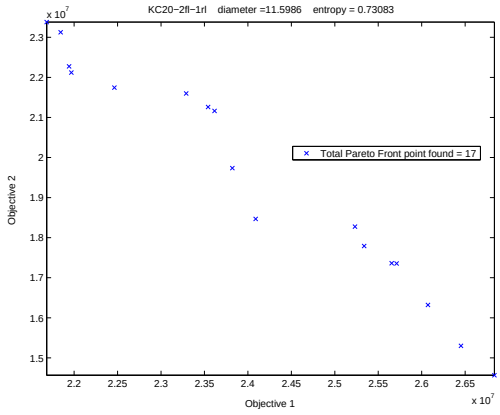

Figure 3: Pareto front found for the KC20-2fl-1rl test instance

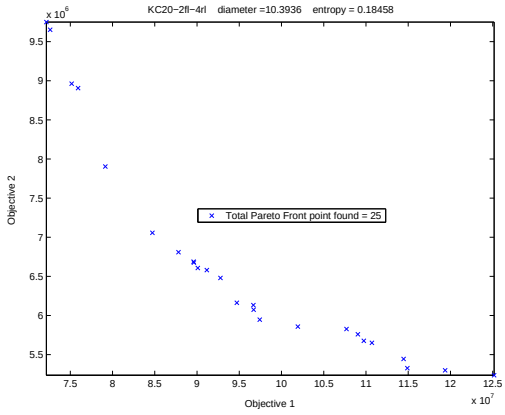

Figure 4: Pareto front found for the KC20-2fl-4rl test instance

lation values were created using the Probabilisitically Complete Initialization method referred to earlier in the paper. The MOMGA-II results are taken over 30 data runs. The MOMGA-II was run on a Beowulf PC cluster consisting of 32 dual-processor machines, each with 1-GB memory and two 1-GHz Pentium III processors (using Redhat LINUX version 7.3 and MPI version 1.2.7.1).

We first ran the MOMGA-II code in order to generate a population with good (low) fitness values for the flows. Next, we ran another program called pareto_enum that pulled out all of the unique pareto front points. Some of the data we generated was rather large, so we had to split it into smaller sets in order to run the pareto_enum program and avoid running out of memory. After we had the unique pareto points for each of our runs. We we then combined the results, one at a time, and used pareto_enum to pull out the unique pareto points for each round. We then wrote a simple Matlab program that showed how our data values improved as more runs were run.

The 30 location and 30 facility instances produced the most interesting 3-d pareto fronts, as can be seen in Figure 7. Once again we generated far fewer points than the benchmark.

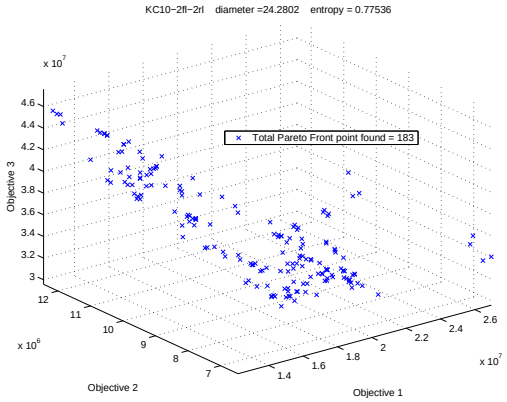

Figure 5: Pareto front found for the KC30-3fl-2rl test instance

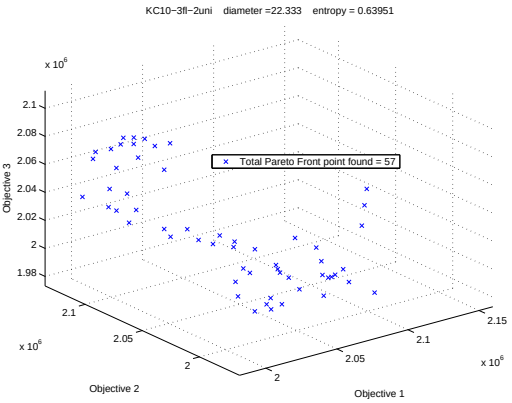

Figure 6: Pareto front found for the KC30-3fl-2uni test instance

5 Analysis

Table 4 compares our results to those found by Knowles and Corne [14]. For all of the instances with 10 locations and 10 facilities, they were able to find the pareto optimal points using an exhaustive search. For the instances with 20 locations and 20 facilities, they employed local search measures which employed 1000 local searches from each of the 100 different λ vectors. For the instances with 30 locations and 30 facilities, they employed a similar local search measure which used 1000 local searches from each of the 105 different λ vectors [13].

By comparing our results for the 10 locations with 10 facilities instances, we see that our results did not equal the results for the pareto optimal. While we don't know the actual values found by Knowles and Corne, we do know that the number of points we generated along the pareto front do not match the numbers Knowles and Corne came up with. This tells us that our present implementation is not able to discern the best points on small instances. In order to improve our results we need to look at our settings and tweak them in order to overcome possible premature convergence. We would also like to determine the exact pareto optimal points and compare them with our results in order to see

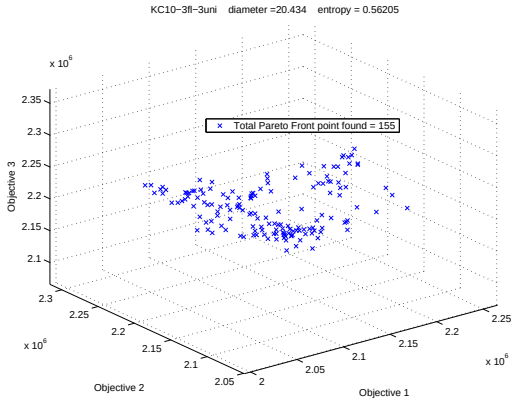

Figure 7: Pareto front found for the KC30-3fl-3uni test instance

exactly how close we are to pareto true.

We were able to see the pareto front advancing as the number of runs we ran increased. Unfortunately, we didn't generate enough members in our population in order to fully populate the front. See the various graphs from Figure 1 to Figure 2 to see the pareto front we found for the 10 location test instances.

Our 20 location and 20 facility results also show that we too few members on the pareto front compared to Knowles and Cornes test results. Once again, we feel that this is due to improper sizing of our building blocks and requires some tweaking to the MOMGA-II parameters. We can make no real conclusion about the entropy results we received, but it does appear that our diameter was almost always smaller than our benchmark. We believe that once we fix our building block size, we will also bring the diameter more in-line with the benchmark results. See Figures 3 to 4 for a visual display of the pareto front we found.

Future work includes cleaning up our code and removing some inefficiencies that we have uncovered. We also want to tweak our code and adjust our building block size in order to improve our results. We would also like to implement the code in parallel and see how much faster it will run compared the serial implementation we ran for this paper.

6 Conclusion

The mQAP is a very difficult problem to solve deterministically for problem sizes greater than 20. Even stochastic algorithms will take a while to get an answer for a large number of locations, simply because the solution space is so large, with a complexity of $O(n!)$. It's imperative to ensure that the proper building block size is used in order to populate the pareto front with enough members to get as close to pareto true as possible.

Bibliography

[1] Vincent Bachelet. *Métaheuristiques Parallèles Hybrides: Application au Problème D'affectation*

Quadratique. PhD thesis, Université des Sciences et Technologies de Lille, December 1999.

[2] Thomas A. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York - Oxford, 1996.

[3] Eranda Çela. *The Quadratic Assignment Problem - Theory and Algorithms*. Kluwer Academic Publishers, Boston, MA, 1998.

[4] DARPA. Darpa and air force select boeing to build ucav demonstrator system. 10 June 2003 `http://www.defenselink.mil/news/Mar1999/b03241999_bt123-99.html`.

[5] L. M. Gambardella, E. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problems. *Journal of the Operational Research Society*, 50:167–176, 1999.

[6] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. Morgan Kauffmann Publishers, 1993.

[7] Peter Hahn, Nat Hall, and Thomas Grant. A branch-and bound algorithm for the quadratic assignment problem based on the hungarian method. *European Journal of Operational Research*, August 1998.

[8] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: processes of inference, learning and discovery*. Computational models of cognition and perception. MIT Press, Cambridge, 1986.

[9] IEEE. *A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem*, volume 3. IEEE, 1999.

[10] IEEE. *Resolution of quadratic assignment problems using an evolutionary algorithm*, volume 2. IEEE, 2000.

[11] IEEE. *Multiple Ant-Colony Optimization for Network Routing*, volume 2241. IEEE, 2002.

[12] Joshua Knowles and David Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. Technical Report TR/IRIDIA/2002-25, IRIDIA, 2002. (Accepted for presentation/publication at the *2003 Evolutionary Multi-criterion Optimization Conference (EMO-2003)), Faro, Portugal*.

[13] Joshua Knowles and David Corne. Towards Landscape Analyses to Inform the Design of Hybrid Local Search for the Multiobjective Quadratic Assignment Problem. In A. Abraham, J. Ruiz del Solar, and M. Koppen, editors, *Soft Computing Systems: Design, Management and Applications*, pages 271–279, Amsterdam, 2002. IOS Press. ISBN 1-58603-297-6.

[14] Joshua Knowles and David Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. In Carlos Fonseca, Peter Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization, Second International Conference, EMO 2003, Faro, Portugal, April 2003, Proceedings*, number 2632 in LNCS, pages 295–310. Springer, 2003.

[15] In Lee, Riyaz Sikora, and Michael J. Shaw. A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 27:36–54, February 1997.

[16] Vittorio Maniezzo and Alberto Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 11:769–778, 1999.

[17] Peter Merz and Bernd Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4:337–352, 2000.

[18] Volker Nissen. Solving the quadratic assignment problem with clues from nature. *IEEE Transactions on Neural Networks*, 5:66–72, 1994.

[19] Panos M. Pardalos and Henry Wolkowicz. Quadratic assignment and related problems. In Panos M. Pardalos and Henry Wolkowicz, editors, *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, 1994.

[20] K. G. Ramakrishnan, M. G. C. RESENDE, and P. M. PARDALOS. A branch and bound algorithm for the quadratic assignment problem using a lower bound based on linear programming. In C. Floudas and P. M. PARDALOS, editors, *State of the Art in Global Optimization: Computational Methods and Applications*. Kluwer Academic Publishers, 1995.

[21] Eric D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location science*, 3:87–105, 1995.

[22] Jesse Zydallis. *Explicit Building-Block Multiobjective Genetic Algorithms: Theory, Analysis, and Development*. PhD thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, March 2003.