Abstract

KUMAR, SUJAY, VIJAYA. *Vitri* - A Generic Framework for Engineering Decision Support Systems on Heterogeneous Computer Networks. (Under the direction of Dr. John W. Baugh)

Vitri is an object-oriented framework implemented in Java for high-performance distributed computing. Using *Vitri*, applications can engage in cooperative problem solving by dividing their tasks among heterogeneous clusters of workstations and PCs. *Vitri*'s features include basic support for distributed computing and communication, as well as visual tools for evaluating run-time performance, and modules for heuristic optimization. It balances loads dynamically using a client-side task pool, allows the addition or removal of servers during a run, and provides fault tolerance transparently for servers and networks. Among its more powerful features are modules for heuristic optimization and decision support tools such as modeling to generate alternatives (MGA). *Vitri* also provides an asynchronous global-parallel genetic algorithm that is particularly suited for coarse-grained tasks executing on processors with large variations in processor speeds. By using dataflow techniques, in which computations are explicitly based on the availability and forwarding of data, the usual end-of-generation synchronization points are removed from the algorithm. The tools in *Vitri* are applied to a number of different applications from the civil engineering domain. The results indicate the adaptability of *Vitri* to various problems and its utility as a tool for managing engineering decision support systems.

Vitri - A Generic Framework for Engineering Decision Support Systems on Heterogeneous Computer Networks

by

Sujay V Kumar

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Civil Engineering

Raleigh, NC

2002

APPROVED BY:

Dr. John Baugh Jr., Chair	Dr. Downey Brill
Dr. Ranji Ranjithan	Dr. Purushothaman Iyer

This thesis is dedicated to my parents Mr. Vijayakumar and Mrs. Komalavally for their love and support throughout my education.

Biography

Sujay V. Kumar was born on January 25, 1975 in Rayonpuram, Kerala, India. He was brought up in a small, picturesque village called Valayanchirangara. He attended the Valayanchirangara High School and Union Christian College, Alwaye for his secondary and higher secondary education. He then joined the Indian Institute of Technology, Bombay in 1992 to pursue his undergraduate study, and graduated as the best student in his class in Civil Engineering in 1996.

He pursued graduate study at the North Carolina State University, Raleigh, where he obtained Master of Science in Civil Engineering in 1998. In August 1998, he started working towards his Ph.D. in the computer aided engineering program in Civil Engineering. His research interests include high performance computing, decision support systems, mathematical modeling, and optimization.

Acknowledgments

I would like to thank Dr. John Baugh, my advisor and committee chairman, for his support, encouragement, and invaluable guidance throughout the course of this work. He was one of the first people to formally introduce me to the wonderful world of computing and has since motivated me to pursue a research career in this field. I am also grateful to Dr. Downey Brill and Dr. Ranji Ranjithan for introducing me to various research interests. As my advisors, they have always helped me stay focused with their insight and constant encouragement. I would like to thank Dr. Purushothaman Iyer, for serving on my committee, and the administrative staff of Civil Engineering for their help and support. I am also deeply indebted to Dr. Dan Loughlin for his support and help at various stages in my research.

Graduate Study at NCSU has been a wonderful experience, during which I made a number of dear friends. I would like thank Vinay Karle and Ken Harrison especially for helping me with my difficult situations, and for being true friends. Thanks to Christos Anastasiou for always showing me the lighter side of life, Shoma Chakravarty for all the wonderful dinners. I also would like to thank my former office mates, Raghu Kurlagunda, Kishan Chetan, Naveen Akunuri, Amey Parandekar for their help, ideas, and constructive criticism. Special thanks to "Dr. Atul Gupta" for all the wonderful tennis and golf sessions during the last five years.

I am also fortunate to have the support of a wonderful family which has been a ceaseless source of support and encouragement. I would like to thank Venuammavan, Sridevi ammayi, Matti, Unni, Paru, and Rajammavan for all the love and affection that you have bestowed on me. My deep gratitude to my father Vijayakumar, and my mother

Komalavally, for instilling in me the high value of education and all the hardships they went through which made this endeavor possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Scientists, engineers, and decision makers are typically faced with problems that have significant social and economic impacts. The process of designing and selecting management strategies for such problems is a complex process. Traditional engineering design focuses on finding strategies that meet the regulatory standards for the problem at hand. The acceptance of a strategy in practice, however, typically depends on a number of other factors such as cost, equity, reliability, political, social, and legal constraints. Incorporating these intangibles often makes the design process challenging. The purpose of this thesis is to develop a computing framework to assist the engineering design and analysis process. The framework is called *Vitri* (a Sanskrit word meaning "distributed"). This chapter describes some of the issues associated with the engineering design process and addresses the need for incorporating new generation of tools, technologies and concepts that motivated the design of *Vitri*.

There are numerous analysis and computer packages that assist a typical engineering design process. A structural analysis program, for instance, can calculate parameters such as deflection, stresses, etc., associated with different types of structures. A structural engineer involved in a design typically follows an iterative procedure adjusting the design parameters until a feasible solution is found. The procedure is often repeated to incorporate other criteria of interest in a real-life situation. Such a brute-force or trial-and-error approach can be ineffective and expensive when the problems increase in complexity and size. Further, the routine invocation of a variety of tools with iterative simulation runs on

a single workstation can be computationally prohibitive.

The idea of using high performance computing resources to handle computationally intensive problems has been reported in the area of problem solving environments (PSEs) [5]. A PSE is a system that provides a complete, usable and integrated set of high level facilities for solving problems in a specified domain. Gallopoulos et al. [5] define a PSE to be a computer system that provides all the computational facilities needed to solve a target class of problems. PSEs allow users to define and modify problems, choose solution strategies, and manage the required computational resources. PSEs may provide expert tools such as algorithms, software tools, hardware resources, etc., for complex problem solving domains. The use of PSEs allows rapid prototyping of applications without the specialized knowledge of underlying computer hardware or software. The idea of developing PSEs have been reported as early as the 1960s [4], with the early efforts mostly focusing on the development of scientific software libraries to facilitate the reuse of high quality software. These early PSEs had severe limitations in their ability to handle large scale problems mainly due to the lack of adequate computing resources. The recent advances in computing technologies have resulted in the ability to use powerful computing resources for solving a wide range of problems. As a result, modern PSEs aim to make use of high performance computing resources, coupled with advances in hardware and software tools [11] such as high-speed workstations, parallel architectures and software, windowing environments, high-level languages, etc.

The concept of a high performance computing environment acting as a backbone for computational support has strongly influenced the design of *Vitri*. One of the powerful components of *Vitri* is a high performance computing environment for distributed computing that harnesses the resources of a network of computers to provide adequate computing power to deal with various problems. This environment with convenient access to heterogeneous distributed computing resources facilitates problem solving without the overheads associated with low-level parallel/distributed application development.

The limitations associated with the traditional design process have also led to the use of a number of new algorithmic tools. Formal tools and concepts such as optimization-based techniques help in searching for strategies that optimize certain desired goals while

meeting user imposed constraints. Such tools help in eliminating ineffective enumeration of potential solutions and finding good solutions that are difficult to identify using a trial-and-error approach. *Vitri* provides a number of formal computational approaches such as heuristic optimization techniques that incorporate domain knowledge to improve efficiency over random search. These techniques provide generic capabilities that are suitable for dealing with difficult, ill-behaved problems.

Although similar, the activities of problem solving and decision making are slightly different. Problem solving typically involves defining goals and suitable courses of action to achieve them, whereas decision making involves choosing appropriate strategies from the available alternatives. Decision support systems (DSSs) are formal approaches for computer assisted decision making. Adelman [1] defines a DSS as an "interactive computer system which utilizes analytical methods, such as decision analysis, optimization algorithms, program scheduling routines, etc., for developing models to help decision makers formulate alternatives, analyze their impacts, and interpret and select appropriate options for implementation." The environmental decision support system (EDSS) [6] developed by the MCNC Environmental Programs Group in cooperation with US EPA and NC State University is such a system that helps scientists and environmental planners model and evaluate environmental quality issues and make decisions at different levels of granularity. A number of applications of DSSs in the engineering domain has also been reported. A spatial DSS for vehicle routing was developed by Keenan [7] that provides techniques to assist in the routing and scheduling of vehicles. Loucks and Costa [8] presented a summary of interactive computer technologies and DSSs for studying water resources problems. DSSs for applications to solid waste [9], structural design [2], transportation [3], and automotive safety [10] applications have been developed by researchers at NC State University.

Researchers have also identified the need for integrated modeling and decision making approaches to gain better understanding of real-life systems [12]. Collaborative approaches that integrate the knowledge of interdisciplinary approaches and the use of knowledge and tools of multiple disciplines have helped bridge the gap between the decision making and problem solving processes. Though a powerful modeling or problem solving environment would benefit a modeler, it may not explicitly satisfy the needs of a decision

maker. A decision maker, for instance, might be interested in details such as the tradeoffs between cost and certain regulatory standards, the effect of uncertainties on solutions, efficient management practices in a given scenario, etc. Formal tools and concepts can be used in an iterative decision making process to assist with such *what-if* analyses. Tools based on optimization can not only be used to find optimal solutions, but also to find good alternatives. For instance, *Vitri* provides a genetic algorithm-based tool, modeling to generate alternatives (MGA), that generates a small set of slightly sub-optimal solutions that are different in decision space. The MGA tool helps in generating alternatives that can be valuable in a decision making scenario. *Vitri* also provides multiobjective optimization algorithms that help in exploring potential tradeoff relationships among different objectives.

In the early stages of dealing with a problem, designers often brainstorm the ideas and come up with simple prototypes to gain a better understanding. It can be observed that both PSEs and DSSs emphasize the importance of a framework with capabilities for rapid prototyping so that the designer/decision maker can learn about the problem incrementally and quickly. A prototyping framework that helps in applying different tools and techniques rapidly to a number of design scenarios becomes essential in the modern design and analysis process. Technologies such as component-based modeling and object-oriented tools help in developing modular systems that are flexible and extensible. The use of such prototyping systems helps designers in learning about the relationships between the parameters involved and in enhancing the overall effectiveness of the design process. The design of *Vitri* is based on object-oriented principles, allowing it to be a flexible, extensible system. As a result, the tools in *Vitri* can be reused for testing different prototypes and applications.

The combination of a high performance computing component to overcome computational limitations and the optimization-based tools in *Vitri* helps in bringing DSSs closer to realizing the decision making power of a true joint-cognitive system. *Vitri* not only facilitates an iterative decision making process, but also helps the decision maker incrementally learn about the problem at hand. The improved knowledge allows the refinement of existing designs and models and the flexible use of DSS capabilities facilitates the development of future prototypes. By providing tools and resources to assist problem solving, and insulating the user from the complexities of underlying hardware and software, *Vitri* facilitates

rapid prototyping of various applications.

1.1 Organization of Thesis

The thesis is organized into three main parts. The first part describes the design of the basic features of *Vitri* such as the distributed computing environment and the distributed genetic algorithms (GA). The use of *Vitri* on substantial applications is described next, followed by the description of its advanced features, MGA and multiobjective optimization tools.

Chapter 2 describes various components of *Vitri* and their design using object-oriented principles. The chapter describes the issues involved in the design and the rationale behind using a specific language, tools, and protocols. The design of the distributed computing system and the connectivity of various algorithms are described with the help of UML diagrams.

The implementation of the asynchronous distributed GA (ADGA) is described in Chapter 3. This chapter also presents a comparative study of the performance of ADGA with a synchronous distributed GA (SDGA). The performance study involves comparison using both fine grained and coarse grained problems.

The application of distributed GA tools in the reliable design of water distribution systems is presented in Chapter 4. This problem is also used to compare the performance of ADGA and SDGA. Chapter 4 also presents a new approach for incorporating a formal notion of redundancy in water distribution system design.

Chapter 5 and 6 compare the performance of the multiobjective optimization tools in *Vitri* when applied to a number of challenging problems. Various performance measures are used to compare the performance of the algorithms with other well established algorithms. The incorporation of these algorithms in *Vitri* is discussed in Chapter 2.

Chapter 7 describes the implementation of the MGA tool. The applicability of the technique is illustrated using sample problems, including an application in seismic performance evaluation.

The appendix A describes a sample execution of *Vitri*. The command line options

for both client and server executions are explained, along with the methods to select simulation variables for the distributed GAs. Appendix B describes the graphical tools in *Vitri* that can be used to monitor the performance of the distributed environment. A summary of the application of *Vitri* to various problems is presented in Appendix C. It lists a table of all the applications, including problems that were used as experiments and prototypes.

References

[1] L. Adelman. *Evaluating Decision Support and Expert Systems*. John Wiley and Sons, New York, 1993.

[2] J. W. Baugh, S. C. Caldwell, and E. D. Brill. A mathematical programming approach for generating alternatives in discrete structural optimization. *Engineering Optimization*, 28:1–31, 1997.

[3] J. W. Baugh, G. R. Kakivaya, and J. R. Stone. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91–123, 1998.

[4] G. J. Culler and B. D. Fried. An on-line computing center for scientific problems. In *Proceedings of the IEEE Pacific Computer Conference*, page 221, 1963.

[5] E. Gallopoulos, E. N. Houstis, and J. R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1(2):11–23, 1994.

[6] H. A. Karimi, S. S. Fine, C. J. Coats, A. F. Hanna, and K. J. Gallupi. An environmental decision support system using high performance computing and communications technologies. In *Proceedings, SCS Simulation Mutliconference*, pages 40–45, Phoenix, AZ, April 1995.

[7] P. B. Keenan. Spatial decision support systems for vehicle routing. *Decision Support Systems*, 22:65–71, 1998.

[8] D. P. Loucks and J. R. Costa. *Decision Support Systems: Water Resources Planning*, volume 26 of *Ecological Sciences*. Springer Verlag, 1990.

[9] S. R. Ranjithan, M. A. Barlaz, E. D. Brill, S. Y. Fu, A. Kaneko, S. R. Nishtala, and H. R. Piechottka. Integrated solid waste management: 2. decision support system. In *Proceedings of the 2nd Congress: Computing in Civil Engineering*, pages 1158–1165. ASCE, 1995.

[10] P. J. Schubert and D. H. Loughlin. Efficient optimization of large k real-time control algorithm. In *Proceedings of Uncertainity '99: The Seventh International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL., 1999.

[11] M. S. Shields, O. F. Rana, D. W. Walker, and D. Golby. A Java/CORBA-based visual program composition environment for PSEs. *Concurrency: Practice and Experience*, 12:687–704, 2000.

[12] H. A. Simon, G. B. Dantzig, R. Hogarth, C. R. Piott, H. Raiffa, T. C. Schelling, K. A. Schepsle, R. Thaier, T. Tversky, and S. Winter, editors. *Report of the Research Briefing Panel on Decision Making and Problem Solving*. National Academy of Sciences, National Academy Press, Washington D.C., 1986.

Chapter 2

Design of *Vitri*

Vitri is an object-oriented framework implemented in Java for high-performance distributed computing. Using *Vitri*, applications can engage in cooperative problem solving by dividing their tasks among heterogeneous clusters of workstations and PCs. *Vitri* provides a number of tools which can be broadly classified as:

- **High Performance Computing and Communications (HPCC) Module:**

  Vitri can utilize a heterogeneous distributed network of computers, providing computational resources to solve complex problems. The low-level implementation details associated with communication between processors are encapsulated through well defined interfaces.

- **Optimization Modules:**

  Optimization is the use of various mathematical algorithms to minimize or maximize certain objectives subject to a set of constraints. Heuristic optimization techniques such as genetic algorithms (GA) and simulated annealing (SA) are increasingly used to solve large combinatorial optimization problems that are encountered in engineering design and analysis. *Vitri* incorporates distributed versions of GAs, for both single and multiple objective problems. *Vitri* also includes an asynchronous global parallel GA that is particularly suited for coarse grained tasks executing on processors with large variations in processor speeds. This algorithm uses dataflow techniques to remove the usual end-of-generation

latencies associated with synchronous global parallel GAs.

- **Modeling to Generate Alternatives (MGA) Tool:**
  The solution found by the optimization algorithm may not be adaptable in a real situation because many objectives and constraints may not be specified in the mathematical formulation. The idea behind MGA [2] is to use optimization techniques to generate a set of solutions similar in the modeled objective space but very different in decision space. *Vitri* includes MGA modules based on GA.
- **Visual Tools:**
  Vitri also includes a number of graphical interfaces to evaluate and monitor the runtime performance of the distributed system.

The design of the various components of *Vitri* is influenced by a number of technologies. The following sections describe those components and the tools and principles that facilitate their implementation.

2.1 Object Oriented Frameworks

Components in *Vitri* are implemented using object oriented design principles. *Vitri* integrates these components into a framework that facilitates their effective use in solving various problems. The use of object-oriented principles help *Vitri*'s flexibility and reusability, thereby enabling rapid prototyping of new applications.

An object oriented approach to software development is based on modeling objects from the real world. It is a way of thinking abstractly about a problem using real world concepts. The structure of an object oriented program mirrors the structure of the problem domain. This approach speeds the development of new programs, and if properly used, improves maintenance, reusability, and modifiability of software.

An object oriented framework represents a software system in a certain domain and provides reusable design for applications within that domain [9]. The major advantage of frameworks is their ability to reuse a proven software architecture. The reusable, "semi-complete" nature of the object oriented frameworks makes it easier to build correct, portable, efficient, and inexpensive applications. Application developers in the past have

invested a lot of time and effort in developing complex applications from scratch since there were no "off-the-shelf" frameworks available. In addition to conserving time and money, the use of a framework simplifies the creation of domain-specific, intelligent tools. The primary features of object oriented application frameworks are modularity, reusability, extensibility and inversion of control [5].

- **Modularity**

  Frameworks enhance modularity by encapsulating volatile implementation details. This prevents a program from becoming so interdependent that a small change will effect propagating changes. Encapsulation helps in changing the implementation of an object without affecting the applications that use it. Though encapsulation is not unique to object oriented languages, the ability to combine data structure and behavior in a single entity makes the approach more powerful than would be with imperative programming languages.

- **Reusability**

  By developing generic components, the existing framework can be used to develop entirely new applications. The reusability of components helps developers combine features of the existing system instead of developing applications from scratch.

- **Extensibility**

  Frameworks define stable interfaces that can be extended by new applications. Extensibility helps in timely customization of new application services and features.

- **Inversion of Control**

  The "inversion of control" property defines the difference between a framework and a traditional software library. In a framework, the framework classes call application classes, in contrast to the usual class libraries that are called from the application. Therefore, the framework can often be seen as a skeleton application, implementing the most important design decisions at an abstract level.

An object-oriented framework normally provides a number of points of flexibility in the design, called "hot spots" [10]. Hot spots are abstract classes or methods that must be

implemented in order to use the framework for a specific application. The parts of a framework that cannot be altered are called the kernel or frozen spots. The kernels are constant parts in each instance of the framework. Thus, the use of hot spots provides implicit reuse of high quality and proven software.

Since the object-oriented design is considered as the modeling of the referent system, a graphical tool for specifying, visualizing, and constructing software systems can be used to convey the structure of the program effectively. The Unified Modeling Language (UML) [7] provides a formal graphical notation for modeling the artifacts of a software system. It unifies many object oriented design and analysis methods such as those developed by Booch, Rumbaugh (OMT), and Jacobson. UML is accepted in industry as a standard tool for the modeling and design of software systems. The UML notation is used in the following sections to describe the object oriented design of *Vitri*.

2.2 Distributed Computing Systems

A distributed system is an infrastructure that allows the use of a collection of autonomous computers connected by an interconnection network. The computers do not usually share memory, but coordinate and share resources by communicating over the network. With improvements in microprocessor technology and high-speed networks, distributed computing is fast emerging as cost-effective alternative to tightly-coupled multiprocessors and parallel computers. Some of the properties and issues associated with a distributed system can be summarized as:

- **Resource Sharing**

  The set of computers connected by a network shares physical and computational resources. The resources could be hardware or software. For instance, given a network of workstations, workstation A may want to use the idle computing cycles of workstations B and C to enhance the speed of a particular computation. Distributed databases are examples of sharing of software resources, where a large database may be stored on several host machines, and consistently updated or queried by a number of agent processes.

- **Communication**

  Distributed systems typically use two main programming paradigms for communication: message passing and shared memory. Message passing is a technique in which processors communicate with each other through messages. These messages are normally commands, event notifications, data, etc. In a distributed shared memory system, all the computers in the network share a common address space. Instead of sending data over the network, the computers share information by writing to the common memory space.

- **Heterogeneity**

  The individual processors in a distributed system may consist of diverse computing hardware, operating systems, or software. Some of the heterogeneity issues are handled by the use of standard message passing standards and low-level protocols that can be readily implemented across different platforms.

- **Concurrency**

  Each processor in a distributed system acts independently of all other processors and operates concurrently with them. At times it is necessary to coordinate the activities of processors, especially in situations such as concurrent updates, to preserve the integrity of the system.

- **Fault Tolerance**

  There are many possibilities of failure in a distributed system compared to a single computer. It is essential that distributed systems incorporate measures to enhance their fault tolerance to prevent the breakdown of the system in case of failures such as a failed node.

- **Load Balancing**

  To maximize throughput in a heterogeneous network, load balancing is required to keep the servers busy by efficiently distributing the workload. There are many methods to achieve load balancing. The simplest method is static load balancing, where a problem is decomposed and subtasks are assigned to processors only once. This partition normally occurs in the early stages of the application. This scheme can be effective in a dedicated, homogeneous system. A dynamic method

of load balancing is normally used when the computational loads vary and when a heterogeneous system is in place.

- **Scalability and Extensibility**

  Compared to tightly-coupled multiprocessors, distributed systems can be expanded by adding more processors if and when they are available. It is important that the distributed system incorporates the additional resources efficiently.

2.2.1 Types of Parallelism

There are typically three types of parallelism associated with parallel/distributed applications. Applications that can be subdivided into sets of tasks that require little or no communication are called perfectly parallel applications, whereas data parallel applications have the same operations performed on different data elements in parallel. The third type of parallelism is called control parallelism, where different operations are performed simultaneously on different processors.

2.2.2 HPCC Module

The HPCC module in *Vitri* is designed to exploit the concurrency in mainly perfectly parallel and data parallel programs. The purpose of this module is to provide a framework that can be easily adapted to handle concurrent applications. The framework is designed to shield the user from the intricacies of concurrency and message passing in a distributed system. This section discusses how the issues associated with the design of a distributed system are addressed in *Vitri*.

Load Balancing in *Vitri*

As mentioned earlier, the performance of a distributed system can be enhanced by load balancing that ensures that each processor is doing its fair share of work. Since a dynamic load balancing method is more effective for a heterogeneous network of workstations, *Vitri* employs a popular dynamic scheme called the "Pool of Tasks" paradigm [8]. The essential idea behind this paradigm is to divide the overall computation into a collection of tasks

which are then scheduled dynamically among a number of processors. This is typically implemented as a client server program where the client manages a set of tasks as shown in Figure 2.1. The servers request tasks from the client as they become idle. Once the servers perform the required computations on the task, the results are sent back to the client.

The pool of tasks paradigm provides automatic dynamic load balancing. This paradigm is especially effective in situations where the servers have very different computational capabilities, because the least loaded or more powerful servers do more work than the slower ones and all the servers stay busy until the end of the problem. Since the faster processors end up completing more tasks than the slower ones in a given amount of time, the idle times of the servers are minimized, improving the throughput of the distributed system

Fault Tolerance in *Vitri*

In contrast to the traditional approach in which a client initiates communication with servers, the roles of client and servers are reversed. In Vitri, the client waits for the servers to connect to it and the servers can connect or disconnect at any time, simplifying fault tolerance. Whenever a server fails, the task performed by that server is simply returned to the pool and subsequently requested by another server. In the implementation, this feature is handled by catching the appropriate exception thrown by the program when a socket connection fails.

The reversed client-server architecture is shown in Figure 2.2. The client opens a Java ServerSocket and waits for incoming connections from the servers. When a connection from a server is accepted, the server gets a port bound to it which can be used for communication with the client. The client also gets a new port bound to the accepted connection, while the original ServerSocket keeps listening for new connections. Information is passed between the client and servers with non-blocking *writes* and blocking *reads*.

Concurrency Issues in *Vitri*

The concurrent updates done by the processors in *Vitri* occur in the pool of tasks implementation. The process of removing a task from the pool is referred to as a "critical section,"

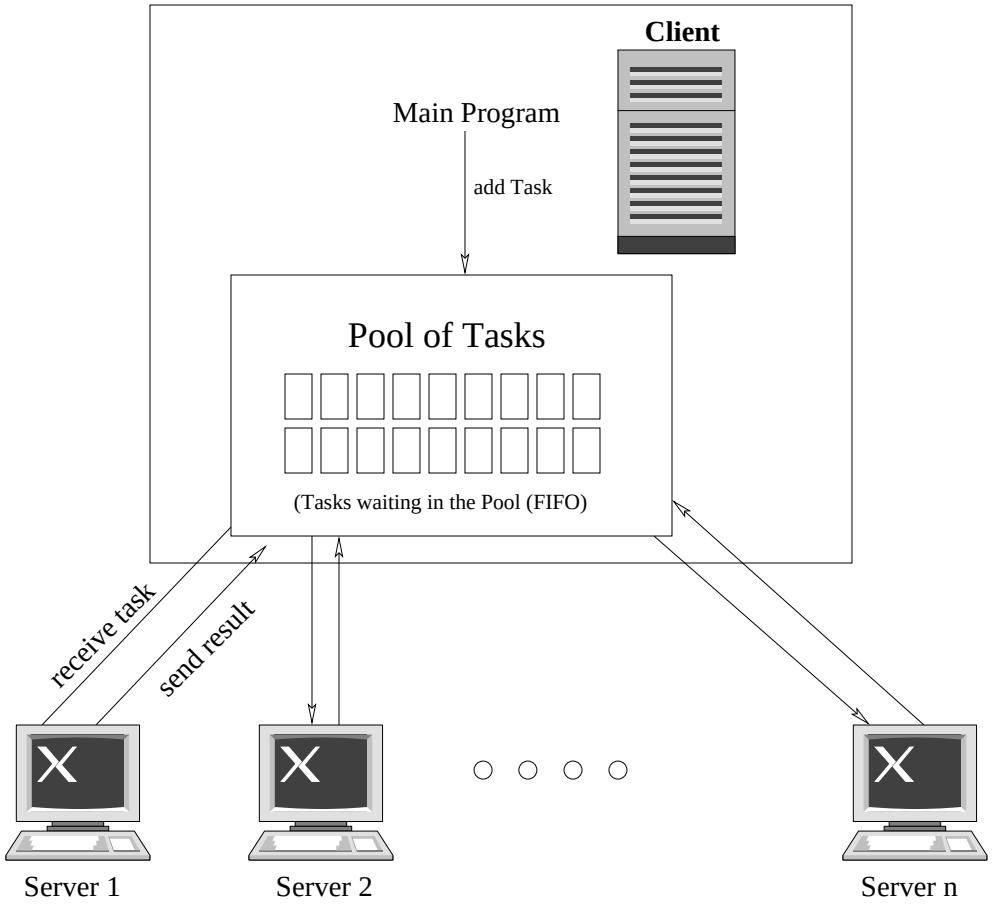


Figure 2.1: The Pool of Tasks Paradigm

which mitigates race conditions in processes or threads that execute concurrently and share memory. This is required to avoid the duplicate evaluation of tasks due to violation of mutual exclusion in a critical section problem.

A simple signal and continue (or wait and notify) monitor is used in *Vitri* to handle the critical section. In this type of monitor, a thread that owns the monitor can suspend itself inside the monitor by executing a wait command and enters a wait set. The thread will stay suspended until some time after another thread executes a notify command inside the monitor. When the notifying thread releases the monitor, the waiting thread will be resurrected and reacquire the monitor.

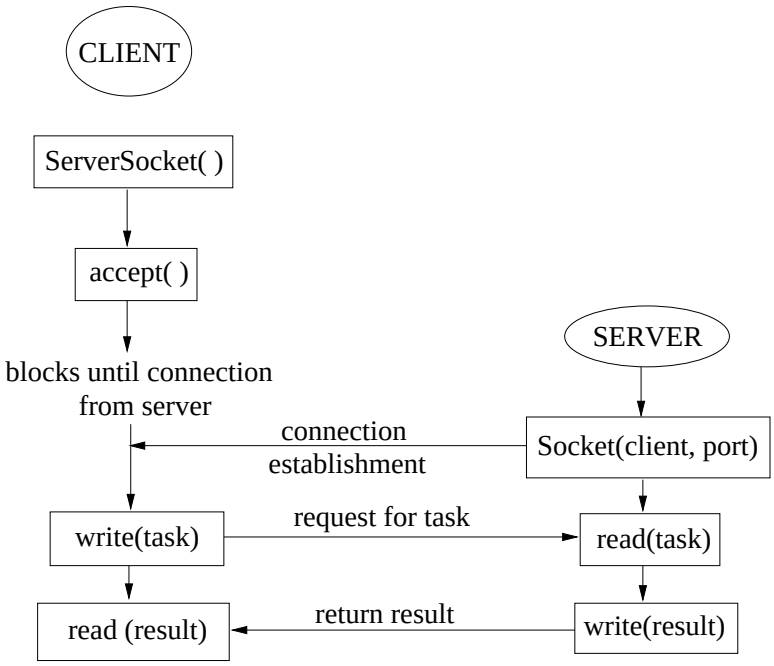


Figure 2.2: Reversal of Client and Server Roles

Heterogeneity, Scalability, Extensibility of *Vitri*

The reversal of the roles of client and servers allows for a fluctuating pool of workstations. As a result, the servers can be used for computations depending on their availability. When a server is available, it can be added to the pool and disconnected when it needs to be used for some other purpose. Thus, *Vitri* provides a scalable, extensible framework that can incorporate a heterogeneous set of workstations.

2.3 Programming Environment

This section describes the programming environment employed in the design and application of *Vitri*. There are many programming languages that facilitate the development of object-oriented frameworks. Further, the development of a distributed computing framework requires the use of a network environment, appropriate protocols for communication, etc. The motivation behind using a specific language and protocols in *Vitri* are explained

in the following sections.

2.3.1 Programming Language - Java

In recent years, the Java programming language and its virtual machine environment have attracted the attention of the scientific community. Java is a modern object-oriented language and includes a number useful features, such as concurrency constructs, portable machine independent interpretation, etc. Java provides a robust, secure language because of the strict type checking at both compile and run times. Java also provides a number of technologies such as object serialization, RMI, etc., for networking applications. The object serialization technology can be used to incorporate checkpointing facilities in the program. Java has been used in a number of HPCC applications [3, 13]. Since Java uses an interpreting system, the programs suffer in performance compared to those written in languages such as C and Fortran. However, modern techniques such as just-in-time compilation can eliminate most of the interpretation overheads. Further, since the tools in *Vitri* mostly deal with coarse grained problems, the interpretation overhead for simply coordinating the evaluation these tasks, which are typically executions of commercial software packages, can be considered to be minimal. In Java, the garbage collection takes place primarily when the program is idle, such as waiting for user input. As a result, Java's performance during execution is not affected compared to other garbage-collected languages. Java also simplifies the use of multiple operating systems within a single execution due to its portability. For instance, Windows, Unix and Linux workstations can co-exist and participate in a single run. The use of Java provides easy application development due to the ever growing wealth of class libraries it provides for newer application domains. These advantages of Java outweigh the disadvantages for the type of applications intended to be used by *Vitri*. As a result, Java is chosen as the programming language for the implementation of *Vitri*.

2.3.2 Communication Mechanism - Sockets

There are a number of "off-the-shelf" distributed computing systems and programming libraries that can be used for customized applications. Technologies such as MPI [6] and PVM [1] are widely used in the scientific community. These paradigms have various features

that make them attractive for certain types of applications and platforms. However, they also have a number of limitations. For instance, PVM provides few tools for security and load balancing, and provides no support for self describing objects, whereas MPI works only on homogeneous clusters and does not include features such as fault tolerance.

In *Vitri*, Java sockets are chosen to implement interprocess communication features. A socket [12] is a unique interface used for transmission of information in a network. Sockets are an accepted industry standard, and are implemented on a variety of platforms. They are also inexpensive for the application in terms of memory and performance. Java provides a class, Socket, that implements one side of a two-way connection between two Java programs. Additionally, Java also includes a ServerSocket class that implements a socket that servers can use to listen for and accept connections to clients. Most of the existing distributed computing environments use sockets as their underlying communication mechanism.

In a traditional client-server system, a client tries to rendezvous with the server by making a connection request on a specific port that is bound to a socket on the server. If the request is accepted, the server gets a new socket bound to a different port. On the client side, if the connection is accepted, a socket successfully gets created and the client can use the socket to communicate with the server. The client and server communicate with each other by reading from and writing to their sockets.

2.3.3 Communication Protocol - TCP

Java sockets using the transmission control protocol (TCP) over the internet protocol (IP) are used for interprocess communication in *Vitri*. TCP is a connection-oriented protocol as opposed to the user datagram protocol (UDP), which is connectionless. TCP provides reliable communication and guarantees that the data are not corrupted during transmission. Although UDP has much less startup latency compared to TCP, it does not guarantee reliable communication.

2.3.4 Communication Medium - Ethernet

Ethernet is a local area network (LAN) technology that is used to transmit information between computers on a network. The Ethernet system consists of three basic elements: (1) the physical medium used to carry Ethernet signals between computers, (2) a set of protocols that allows multiple computers to arbitrate access fairly to a shared Ethernet channel, (3) an Ethernet frame that consists of a standardized set of bits used to carry data over the system. Ethernet uses the CSMA/CD (carrier sense multiple access with collision detection) protocol to determine which host gets access to the network. In CSMA, each host must wait until there is no signal on the channel, then it can begin transmitting. If some other interface is transmitting, there will be a signal on the channel, which is called carrier. All other interfaces must wait until the carrier ceases before trying to transmit. It is possible for two hosts to sense that the network is idle and to start transmitting their frames simultaneously. When this happens, the Ethernet has a way of detecting the "collision" of signals, at which point the transmission is stopped and the frames are resent after a random delay. The CSMA/CD protocol provides fair access to the shared channel so that all stations get a chance to use the network.

2.4 Software Architecture

The following sections describe the software architecture of the major components of *Vitri*. The design is conveyed using UML diagrams and skeletal code wherever necessary.

The various modules in *Vitri* are organized as different packages as shown in Figure 2.3. The vitri.ga package refers to the GA related code repository. vitri.net contains the pool of tasks implementation, vitri.df contains the generic dataflow representations, and vitri.dist contains abstractions such as a representation of a distributed program. The dotted lines represent the dependencies between different packages in which an arc from A to B indicates that package A uses package B. The vitri.ga package uses the dataflow implementation for the asynchronous GA, and uses the vitri.net and vitri.dist packages for the distributed implementation of GAs.

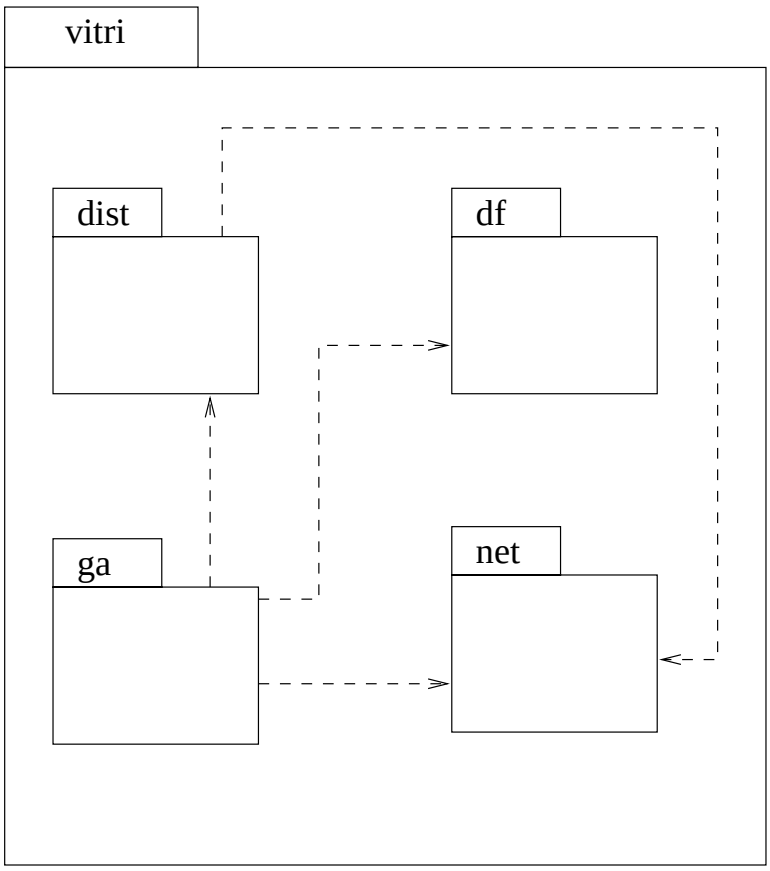


Figure 2.3: Package structure of the code repository

2.4.1 HPCC module

This section begins with the vitri.net package, which implements the most basic aspects of the system. This package contains the lowest level of abstractions and does not depend on any other module for execution.

Figure 2.4 illustrates the design of the pool of tasks in *Vitri*. The pool of tasks structure is implemented by the Pool class, which contains the queues in which tasks are stored while they are awaiting evaluation and while they are getting evaluated. The Pool upon construction, initiates a PoolThread class, which manages the connections to different servers. This is achieved by opening a ServerSocket and waiting for servers to connect to it. Once a server is connected, a ConnectThread is spawned that starts communicating with

the server through a TCP/IP socket connection.

The Job class implements objects that can be placed in the pool of tasks and it contains the object to be evaluated and the result of the evaluation. Objects to be evaluated must implement the Task interface and the objects that are returned as the result of a server side evaluation must implement the Result interface. The CustomSerializable interface provides methods to implement customized marshalling and unmarshalling of objects written to the socket's I/O streams by means of writeSerializable and readSerializable methods.

Originally, the object serialization mechanism in Java was used in *Vitri* instead of customized marshalling and unmarshalling of objects. Object serialization supports the encoding of objects and the objects reachable from them into a stream of bytes and it supports the complementary reconstruction of the object graph from the stream. The use of object serialization, however, is not efficient when a large number of objects are constantly written in and out of the socket streams. The implementation of Java's ObjectOutputStream maintains the mapping of objects written to it, that might otherwise be unreachable by an application, can result in a situation of running out of memory. A customized marshalling and unmarshalling in which the required data is converted to a stream of bytes and the corresponding reconstruction, instead of the entire object, can be used to eliminate this problem.

Figure 2.5 shows how the pool of tasks implementation is integrated with the client server model. Client and Server classes represent the implementation of client and server, respectively. The Program class is an abstract representation of a distributed program that provides methods to add a job to the pool of tasks, and to block for the result of an evaluation. The Client and Server classes are frozen spots that use instances of a custom Program, Task, and Result classes to manage a distributed environment.

2.4.2 Hot Spots provided by the HPCC module

A distributed version of a perfectly parallel program can be implemented by using the hot spots provided by the HPCC module. Classes representing the task that need to be computed and the result of the computation, need to be implemented (say CustomTask and CustomResult, respectively). These two classes implement the interfaces Task and Result,

respectively. Skeletal code for these classes are shown below. Customized marshalling and unmarshalling can be implemented in these classes.

```
public class CustomTask implements vitri.net.Task {
    //define constructor, variables
    public CustomTask( .. ){
       ...
    }
    // implement the actual computation performed on the task
    public Result runTask(){
       ...
    }
    // customized marshalling and unmarshalling if needed
    public void writeSerializable( ObjectOutputStream out ){
       ...
    }
    public void readSerializable (ObjectInputStream in ){
       ...
    }
}
public class CustomResult implements vitri.net.Result{
    //define constructor, variables
    public CustomResult( .. ){
       ...
    }
    // customized marshalling and unmarshalling if needed
    public void writeSerializable( ObjectOutputStream out ){
       ...
    }
    public void readSerializable (ObjectInputStream in ){
       ...
    }
}
```

These classes are used in a main program (say CustomProgram), that implements the run method of the Program class. The skeletal code for the CustomProgram is shown below.

```
public class CustomProgram extends vitri.dist.Program{
    //default constructor
    public CustomProgram( .. ){
       ...
    }
```

```
    public Object run(boolean distributed){
    //create tasks that need to be placed in the pool
       ...
    new CustomTask();
       ...
    //add them to the pool as jobs
    addJob( ... );
    //block for the results if needed
    block( ... );
    //process results if required
    }
}
```

2.4.3 Optimization Modules

The optimization modules in *Vitri* are based on GAs. *Vitri* provides two distributed versions of GAs: a synchronous distributed GA (SDGA) and an asynchronous distributed GA (ADGA). Both GAs are based on client server models so that they can be integrated with the HPCC module.

Figure 2.6 shows the overall structure of the distributed GA implementations. The GeneticAlgorithm class is an abstract representation of a simple GA. The SyncGeneticAlgorithm and AsyncGeneticAlgorithm classes represent the SDGA and ADGA, respectively and they both extend the GeneticAlgorithm class. The population classes SyncPopulation and AsyncPopulation, which are subclasses of Population, correspond to the SDGA and ADGA, respectively. The Population class contains an aggregation of a number of Organisms. The Fitness class represents the implementation of a fitness of an organism. The Fitness class allows the user to define the variables to be returned after fitness evaluation, as typically information other than the fitness values are required to assess the quality of solutions generated by a GA.

More details of the ADGA are shown in Figure 2.7. The Node and the Arc classes correspond to operators and data dependencies in a dataflow program, respectively. The genetic operations in the ADGA are represented as nodes in a program by extending the Node class (Compete, Compare, Mate and Evaluate). In Figure 2.7, these four classes are represented by GAOperation. Evolve is a special node that executes the genetic operations

and updates the subsequent population. The Arc class is used as a wrapper for the Organism class in an ADGA. The data dependencies in the dataflow graph are satisfied when the nodes in the graphs are linked together by the appropriate arcs (or the correct organism).

The overall connectivity of the GAs with the distributed architecture is shown in Figure 2.8. As explained above, to implement a distributed version of an algorithm, a main class that extends Program and customized versions of Task and Result classes need to be implemented. In the case of a distributed GA, the main class is a program (SyncGeneticAlgorithm for SDGA, AsyncGeneticAlgorithm for ADGA) that initializes the GA populations and computes the genetic operations. In a GA, the inherent parallelism comes from the fact that the fitness evaluations of organisms are independent of each other. The Organism class implements the Task interface since it is the object that needs to be evaluated, and the Fitness class implements the Result class since it is the result of the evaluation.

2.4.4 Hot Spots for the Optimization Module

Vitri provides a number of hot spots for the distributed GAs so that they can be adapted easily for a specific problem. Customization is achieved by implementing a class that provides the genetic representation, recombination, and mutation operators along with the fitness evaluation method for the specific Organism. Further, a customized class that extends the Fitness must be implemented. This class includes the variables that are returned after the fitness evaluation in the server. Other classes such as SyncGeneticAlgorithm, AsyncGeneticAlgorithm, Population, etc., are defined as frozen spots.

Sample code for a specific GA organism implementation is shown below.

```
public class CustomOrganism extends vitri.ga.Organism{
  //define a representation, variables
     ...
  //define constructors
     ...
  //randomly create a new organism
  public void randomize(){
     ...
  }
  //create the recombination operation
```

```
  public void crossover(){
     ...
  }
  //create the mutation operation
  public void mutate(){
     ...
  }
  //define how fitness evaluation is carried out
  //and return the appropriate Fitness class
  public Fitness evaluate(){
     ...
  }
  //define a clone method
  public Object clone(){
     ...
 }
}
```

The custom implementation of a Fitness class can be defined as:

```
public class CustomFitness extends vitri.ga.Fitness{
   //define attributes that need to be returned after
   //a fitness evaluation
     ...
   //define constructors
     ...
   //define clone method
     ...
}
```

2.4.5 Multiobjective Optimization Tools

Vitri provides a number of multiobjective optimization tools based on the methods presented by Ranjithan et al. [11] and Chetan [4]. *Vitri* provides implementations of two multiobjective optimization algorithms: (1) The aggregate weighting method-based approach called the noninferior surface tracing evolutionary algorithm (NSTEA) and (2) a constraint-based approach called constraint method based evolutionary algorithm (CMEA). The structure of these tools in *Vitri* are shown in Figure 2.9.

The NSTEAGeneticAlgorithm and CMEAGeneticAlgorithm represent the main programs corresponding to NSTEA and CMEA, respectively. The multiobjective GAs use

a custom implementation of the MultiObjectiveOrganism class. Both NSTEA and CMEA involve repeated executions of GAs to determine the set of noninferior solutions. Each of these executions can be done in a distributed fashion, using either the SDGA or ADGA. As a result, the multiobjective GAs can also be executed in a distributed fashion. Since the MultiObjectiveOrganism class extends the Organism class, the custom organisms are forced to implement the Task, similar to the case with ADGA and SDGA.

The hot spots provided in the multiobjective GA algorithms are similar to that of the single objective GAs. In order to use the multiobjective GA algorithms, a custom organism class that extends MultiObjectiveOrganism need to be implemented. A custom organism class for multiobjective GAs must implement the getZ method (in addition to other methods of Organism class described earlier) that returns the values of different objectives considered.

2.4.6 MGA Tool

The MGA techniques are embedded in the SDGA implementation by providing a number of methods in various GA classes. The software architecture of the MGA implementation is shown in Figure 2.10.

To obtain MGA solutions for a certain problem, an organism class that extends the MGAOrganism class must be implemented. The MGAOrganism class requires the implementation of a method that calculates the difference criteria, which are used to determine the relative distance between solutions in the decision space. Further, if a problem-specific criterion is used to choose MGA solutions, it can be implemented in the getMGACriteria method.

The MGA solutions can be generated by running the SDGA either sequentially or by using a distributed system. Similar to the case with SDGA and ADGA, the custom organism and fitness classes must implement the hot spots to use the distributed environment.

2.4.7 Summary

The software architecture of *Vitri* reveals many properties and features of object oriented frameworks. The components of *Vitri* are modeled as a set of classes with capabilities for customizable extensions. The reusable features in *Vitri* enable it to share components for different applications. From the UML diagrams presented in earlier sections, it can be seen that *Vitri* creates a number of skeleton designs using inheritance. Further, polymorphism allows users to redefine methods and classes. *Vitri* offers the prospect of reusing designs and code for future applications. The modularity of the designs allows *Vitri* to hide the idiosyncrasies of working with different architectures and heterogeneous machines and to extend the framework for different applications. *Vitri* employs the "inversion of control" property by acting as a skeleton application, allowing different applications to execute using the features it provides.

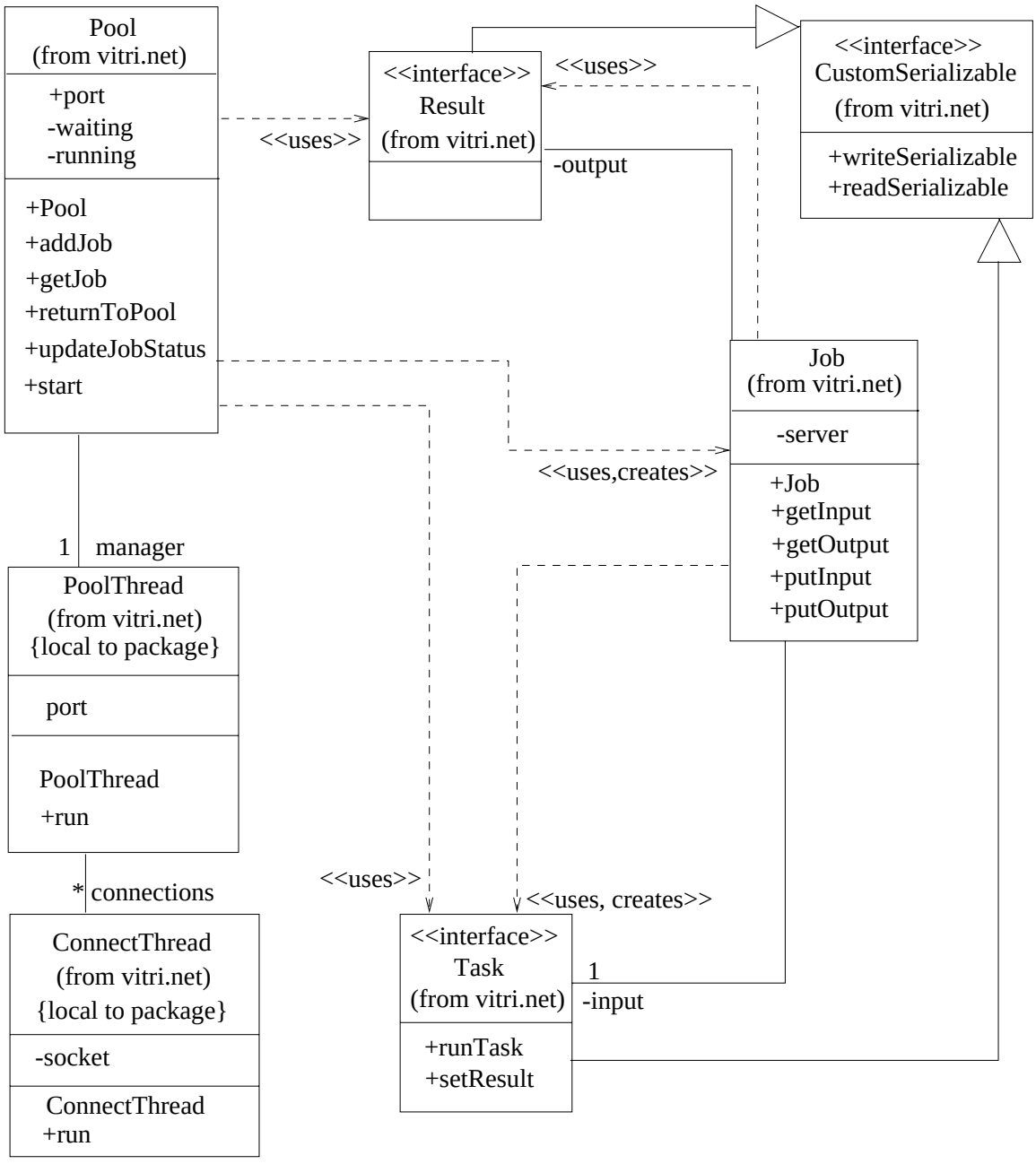


Figure 2.4: Class Diagrams for the Pool of Tasks Implementation

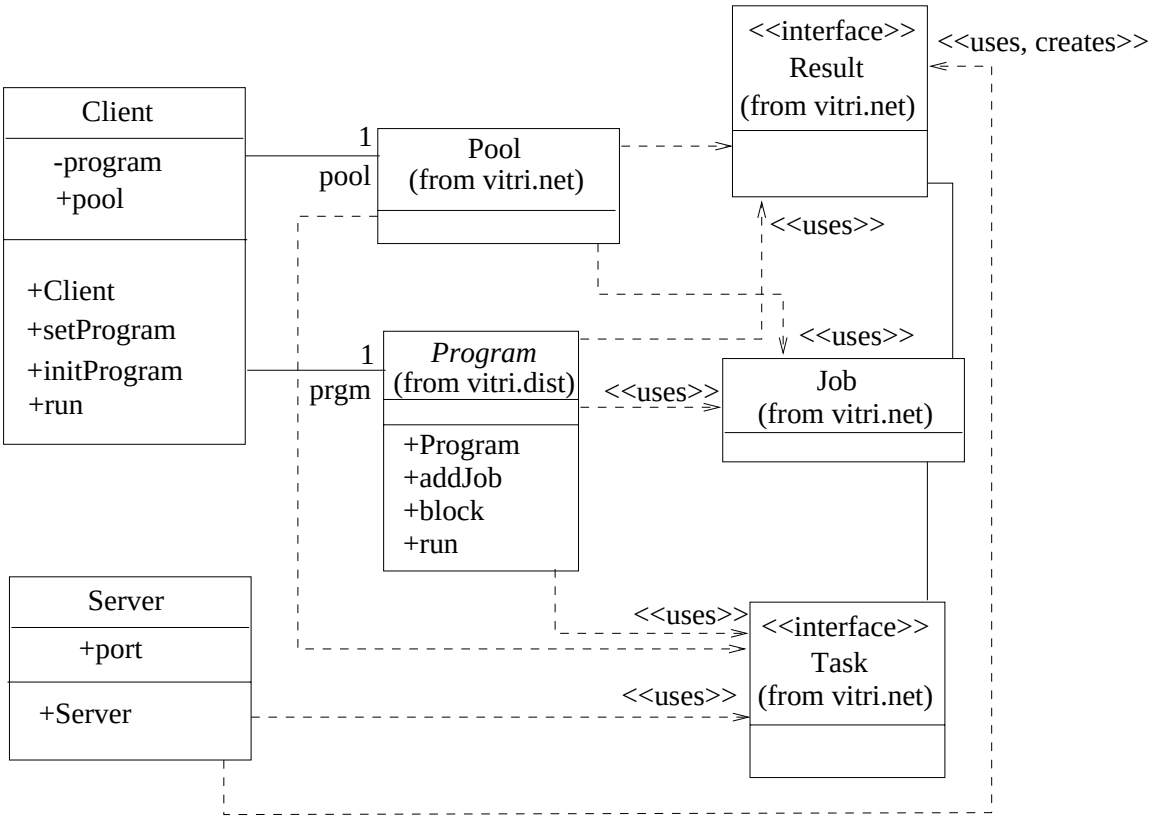


Figure 2.5: The Pool of Tasks Implementation combined with a Client Server Model

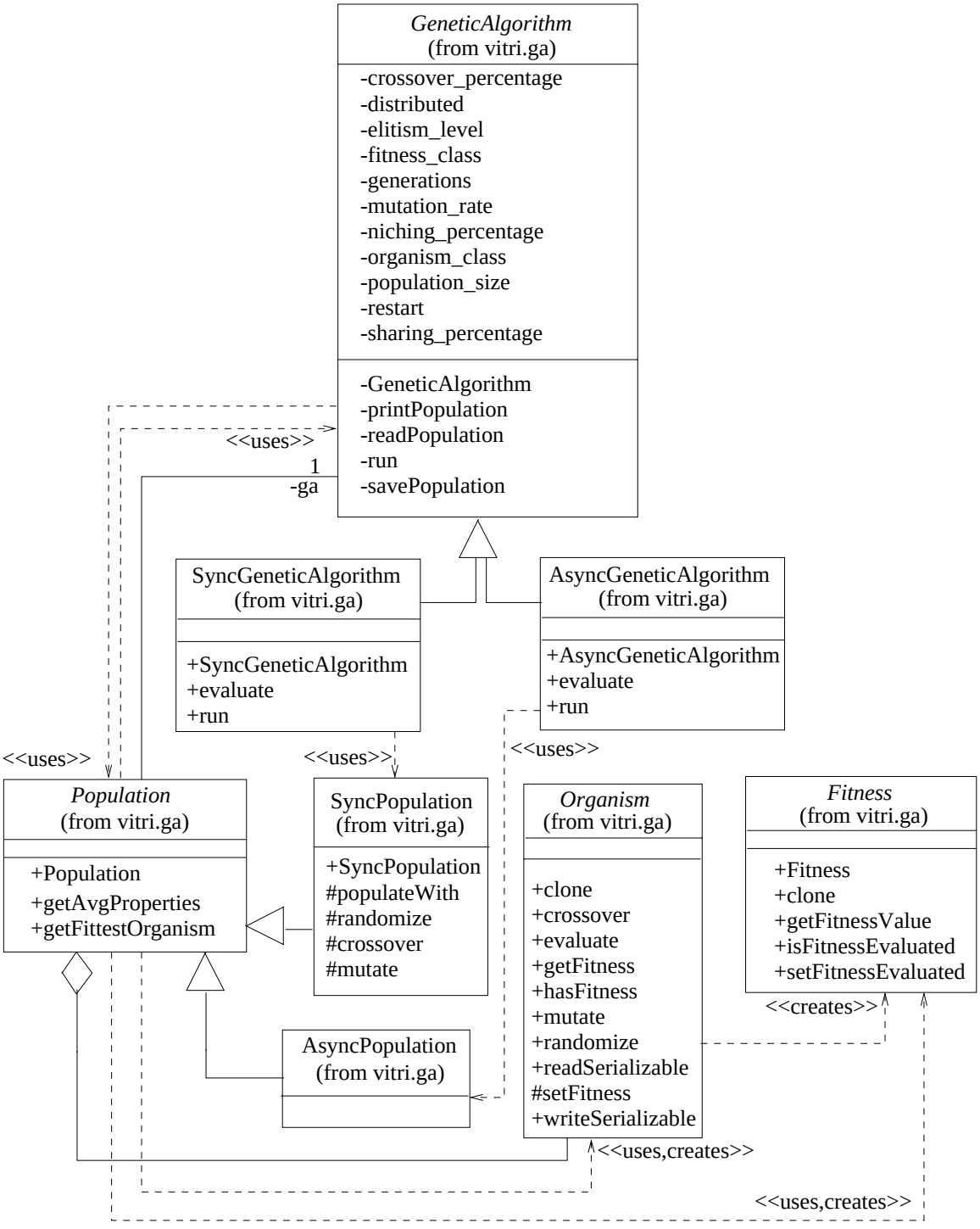


Figure 2.6: Class Diagrams for the Distributed GA Implementations

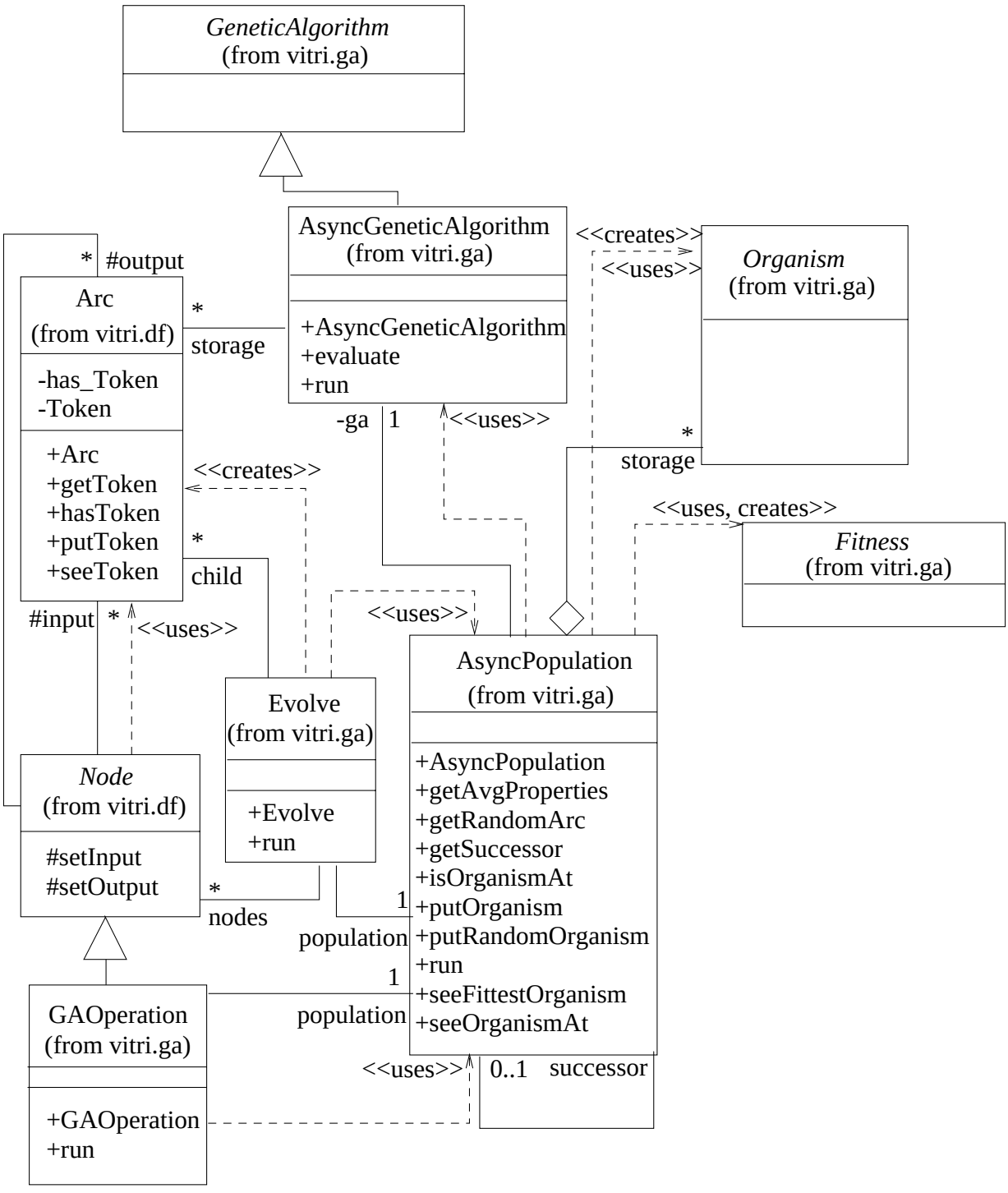


Figure 2.7: Class Diagrams for the ADGA

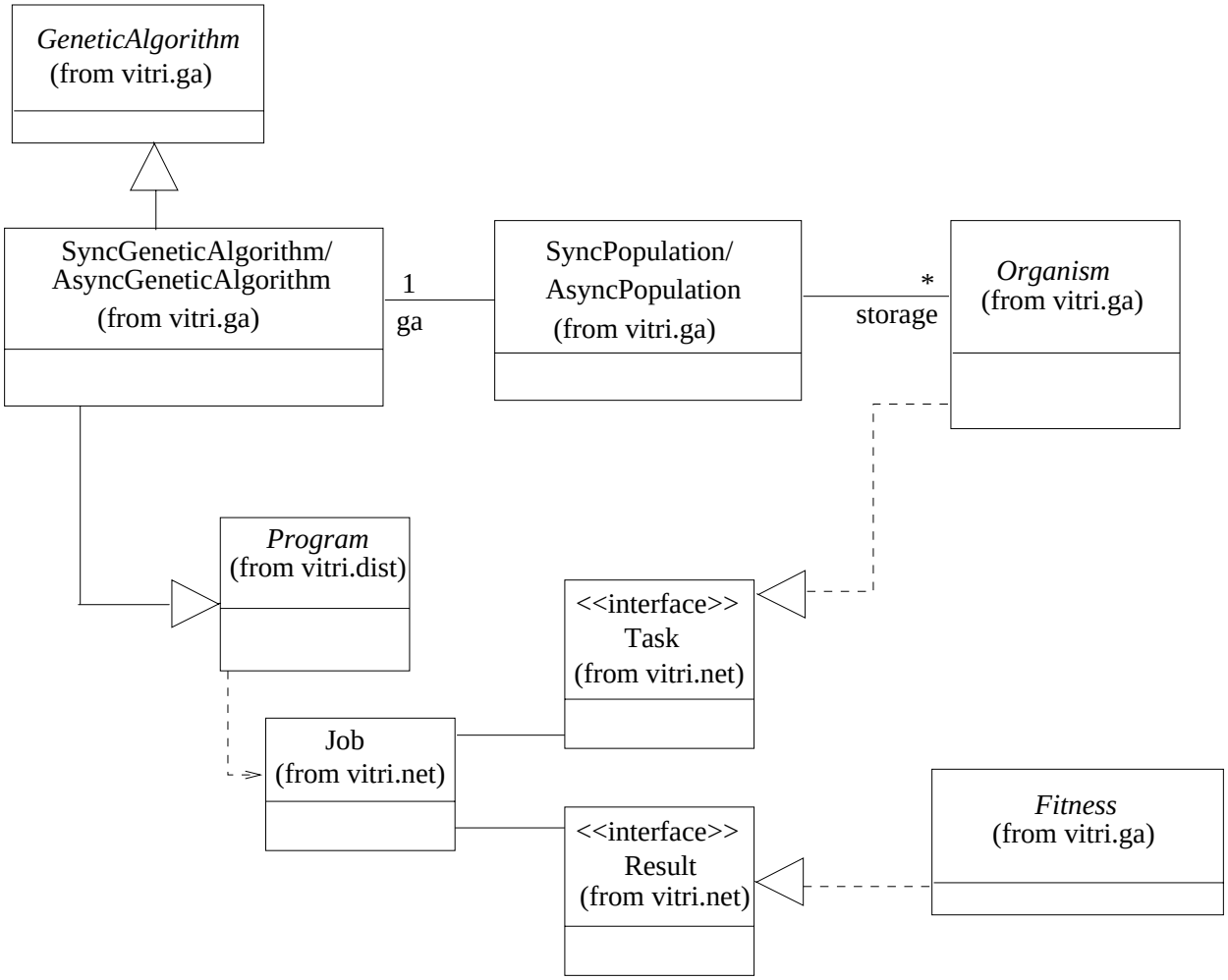


Figure 2.8: Connectivity of the GA implementations with the HPCC module

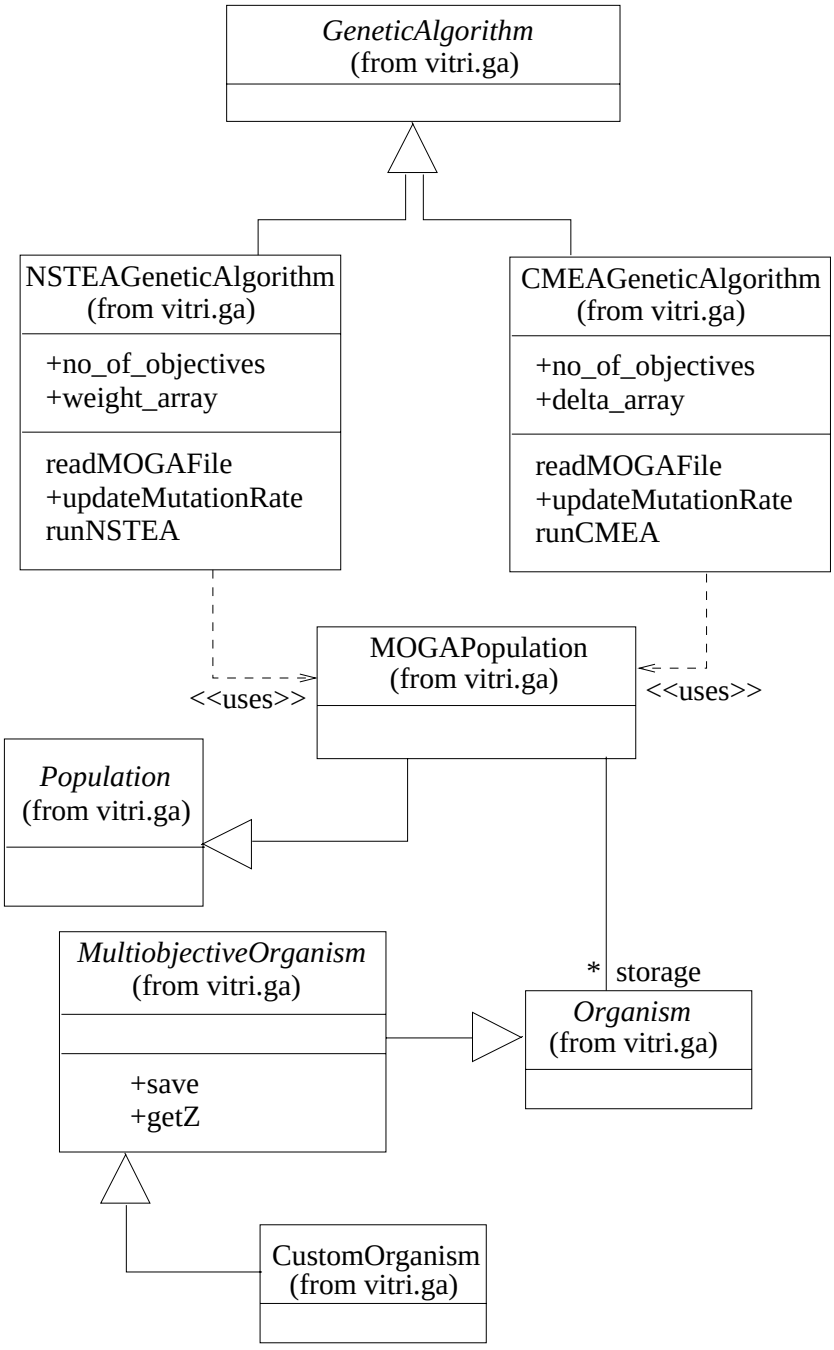


Figure 2.9: Class Diagram for the Multiobjective GA modules

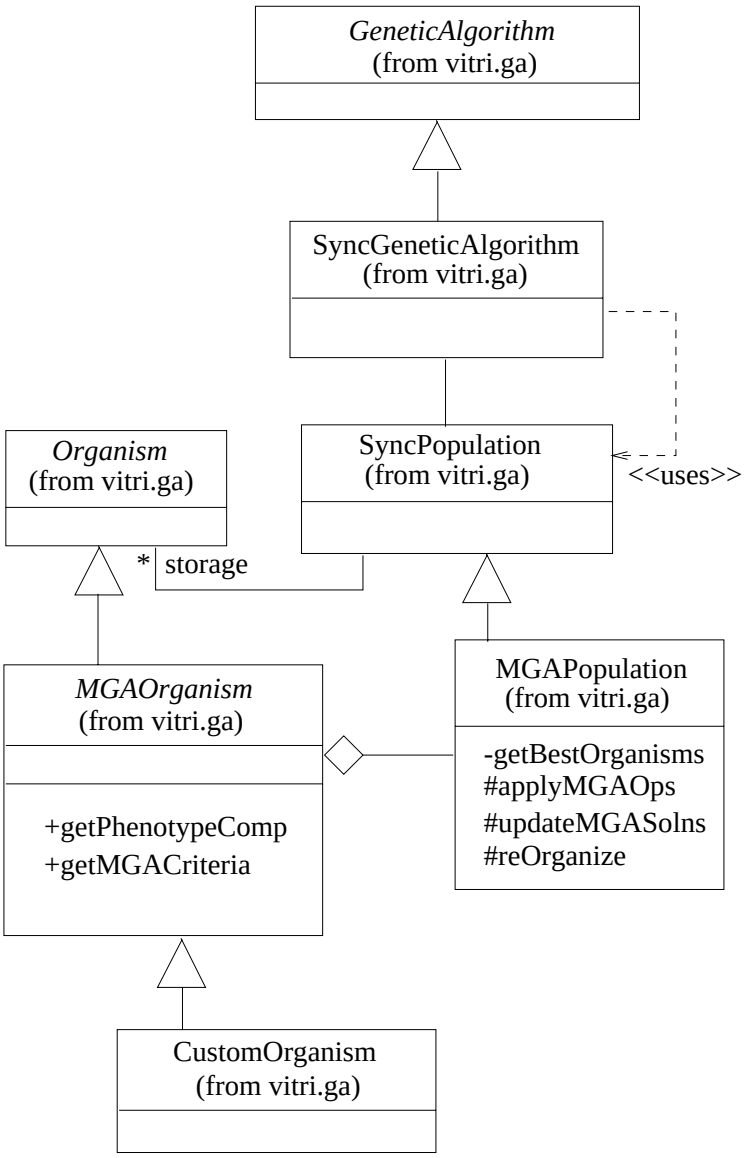


Figure 2.10: Class Diagram for the MGA module

References

[1] A. Beguelin, J. Dongarra, A. Geist, R. Mancheck, and V. Sunderam. A user's guide to PVM: Parallel virtual machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, 1992.

[2] E. D. Brill, J. M Flach, L. D. Hopkins, and S. R. Ranjithan. MGA: A decision support system for complex, incompletely defined problems. *Transactions on Systems, Man and Cybernetics*, 20(4):745–757, 1990.

[3] P. Capello, B. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-based parallel computing using java. Technical report, University of California at Santa Barbara, 1997.

[4] S. K. Chetan. Noninferior surface tracing evolutionary algorithm (NSTEA) for multi-objective optimization. Master's thesis, North Carolina State University, Raleigh, NC, 2000.

[5] M. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Communications of the ACM, Special Issue on Object-Oriented Application Frameworks*, 40(10), October 1997.

[6] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, MA, 2 edition, 1999.

[7] Object Management Group. UML 1.1 documentation. http://www.rational.com, 1997.

[8] H. P. Hofstee, J. J. Likkien, and J. L. A. Van De Snepscheut. A distributed implementation of a task pool. *Research Directions in High-Level Parallel Programming Languages*, pages 338–348, 1991.

[9] P. Nowack. Architectural abstractions for frameworks. In Jan Bosch and Stuart Mitchell (Eds.), editors, *Lecture Notes in Computer Science, Object-Oriented Technology, ECOOP'97 Workshops*, volume 1357, pages 116–118, Jyvaskyla, Finland, June 1997. Springer Verlag.

[10] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, Reading, MA, 1995.

[11] S. R. Ranjithan, S. K. Chetan, and H. K. Dakshina. Constraint method-based evolutionary algorithm (CMEA) for multiobjective optimization. *Lecture Notes in Computer Science*, LNCS 1993:299–313, 1993.

[12] S. Sechrest. An introductory 4.3BSD interprocess communication tutorial. Technical report, Computer Science Research Group, Department of Electrical Engineering, University of California, Berkeley.

[13] D. W. Walker and O. F. Rana. The use of Java in high performance computing: A data mining example. In P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, editors, *Proceedings of the Seventh International Conference on High Performance Computing and Networks*, volume LNCS 1593, pages 863–872. Springer Verlag, 1999.

Chapter 3

Asynchronous Genetic Algorithms for Heterogeneous Networks using Coarse-Grained Dataflow

(Chapter 3 is a reprint of the manuscript submitted to IEEE Transactions on Evolutionary Computation)

by John Baugh and Sujay Kumar

Abstract

Genetic algorithms (GAs) are an attractive class of techniques for solving a variety of complex search and optimization problems. Their implementation on a distributed platform can provide the necessary computing power to address large-scale problems of practical importance. On heterogeneous networks, however, the performance of a global parallel GA can be considerably limited by synchronization points during the computation, particularly those between generations. We present a new approach for implementing asynchronous GAs based on the dataflow model of computation — an approach that retains the convergence features of global parallel GAs. Our implementations are developed on a custom fault-tolerant distributed framework that also provides automatic load balancing. A simple analytical model in terms of parameters such as communication and computation cost is developed for predicting performance in other computer and networking contexts. Experiments conducted with an air quality optimization problem and others show that the performance of

GAs can be improved considerably through dataflow-based asynchrony.

3.1 Introduction

Recent advances in desktop computers and networking technology have lead to an enormous increase in affordable computing power. This revolution in computing technologies has provided the means to solve otherwise computationally intractable problems. Beowulf-class distributed computing provides a viable alternative to supercomputers by integrating the use of hardware systems, software tools and generic programming approaches. Distributed systems consist of loosely coupled autonomous processors that exchange information over a network. Distributed systems are useful since they are flexible in terms of expansion and scalability of resources and they improve performance through parallelism. Problems that require more resources than a single workstation can often be solved using the combined computational power of a distributed system.

Optimization problems typically form an important class of the engineering design and application domain. Large combinatorial optimization problems (NP-complete) are difficult to solve by analytic or simple enumeration methods. Moreover, traditional heuristics that are used to find sub-optimal solutions are not always satisfactory since they can converge to local optima. Genetic algorithms (GAs) offer a unique heuristic approach based on the principles of natural evolution and genetics and are well suited for difficult combinatorial optimization problems. Unfortunately GAs are computationally intensive and parallelizing them help to reduce the total execution time.

The notion behind parallel programs is to divide the tasks in hand into a number of subtasks and solve them simultaneously using multiprocessors. GAs can be parallelized in many different ways depending on how the subtasks are defined and assigned to the processors. One of the simple implementations of parallel GAs is a global master-slave GA. This class of GAs typically employs a single population without locality considerations. In the master-slave type of parallel GAs, a master processor stores the population and executes the genetic operators, with the fitness evaluations being carried out by the slave processors.

The degree to which GAs can exploit the resources of a networked environment de-

pends on a number of design decisions that affect the parallelism available. In a synchronous distributed GA (SDGA) implementation, for instance, the computation waits until all the fitness evaluations in a particular generation are complete before proceeding to the next generation. In a heterogeneous network, slow processors can impede the progress of the program at the synchronization points by leaving faster processors idle while the slower ones are finishing their computations. The speedup gained by parallel processing can be significantly constrained by these synchronizations between generations. An asynchronous approach that removes these bottlenecks has been developed by applying the dataflow model of computation to GAs. The resulting asynchronous distributed GA (ADGA) improves the performance of the GA by eliminating the end-of-generation synchronization points.

In this paper, implementation of the ADGA is presented. The ADGA and SDGA are both applied to the same two problems, for comparison with respect to parameters such as execution time, speedup, and efficiency. A theoretical model that predicts the performance characteristics of the GA implementations is also presented. In addition to the performance parameters mentioned above, scalability is also considered. Scalability identifies the suitability of an algorithm for solving laborious problems and gives a measure of the cost of running the algorithm for the expected level of success.

3.2 Related Work

GAs were developed by John Holland [20] and have been widely studied, experimented with and applied in many engineering and scientific problems. GAs are modeled loosely on the principles of evolution and genetics, employing a population of individuals or candidate solutions. The individuals undergo selection in the presence of variation inducing operators such as recombination and mutation. The individuals are evaluated according to some predefined quality criterion called fitness.

There have been several studies on parallel implementations of GAs [27]. Many of these studies tried to exploit the inherent parallelism in GAs to achieve better performance. One of the ways to parallelize GAs is called global parallelization, where the evaluations of individuals are done in parallel [9]. This method can achieve significant speedups if the

communication costs are small compared to the computation costs. Kwok and Ishfaq [23] describe a parallelized version of a GA, in which the population is divided into a number of evolving subpopulations. Using an Intel Paragon processor, they report near linear speedups in solving the problem of scheduling allocation and sequencing of tasks on the processors. Kim and Zeigler [21] present an architecture with hierarchically arranged clusters each solving different degrees of abstracted problems on a Motorola MPC-2000 workstation. Fogarty and Huang [14] describe a distributed-memory architecture to evolve a set of rules for a pole balancing application. Hauser and Manner [18] describe a global parallel GA on three different parallel computers, but report good speedups only on a NERV multiprocessor, which has very low communication overhead. All the schemes mentioned above are based on the use of high performance, tightly coupled parallel computers.

Several GA implementations using a network of workstations (NOWs) have been reported [26, 33]. Kumar et al. [22] present a distributed GA using PVM (Parallel Virtual Machine) to solve various network scheduling problems on a LAN of HP-UX workstations. Easton and Mansour [13] describe a distributed GA on a network of message passing workstations to solve labor scheduling problems. Calegari et al. [8] present an island-based distributed GA with a network consisting of 80 Sparc workstations and report near linear speedups. A parallel GA on a distributed network of workstations for analyzing biological sequences is described by Anbarasu et al. [3]. The approach was implemented on the PARAM 1000, a parallel computer with a cluster of workstations and they obtained significant improvements over a sequential GA. Bevilacqua et al. [7] apply a distributed GA, on a heterogeneous network of workstations to a problem of parameter optimization in medical image analysis.

Numerous researchers have studied the performance characteristics of distributed GAs. The performance of a distributed GA that incorporates parallel cooperative-competitive genetic operators was studied by Aguirre et al. [2]. They performed simulations with an island-GA and reported significant speedups and convergence. Sekanina and Dvorak [29] performed simulations to assess performance characteristics such as efficiency and speedup for a ringed GA, a type of island-based GA with a particular topology.

Though there are many techniques to parallelize GAs, it is important that paral-

lelization not affect the quality of solutions. Techniques other than global parallelization introduce fundamental changes in the structure of the GA [9]. For example, multiple population GAs involve the use of a number of subpopulations, which necessarily interact with each other. In this class of GAs, the individual subpopulations may converge to an inferior solution if the subpopulations are not interacting well enough (i.e., if the interaction parameters are not tuned along with other GA parameters). The level of interaction is usually required to be above a certain threshold, which is mostly problem specific. Gordon and Whitley [16] compared a number of different parallel GAs for a wide range of optimization functions. The performance of the simple global parallel GA was comparable to that of island-based approaches.

The limitations of global parallel GAs due to synchronization points have been studied by a number of researchers. Schleuter [28] proposed an asynchronous implementation to eliminate the synchronizations and exploit more parallelism. This approach was based on a spatial population with communication happening between any two nodes in the network. Mayurama et al. [25] developed a fine grained ADGA and assessed its performance on a tightly coupled multiprocessor as well as on workstations connected by a local area network. Experiments on function optimization and graph partitioning produced linear speedups on both types of platforms. The parallelism of GAs was also examined by Gordon et al. [17] using implicitly parallel programming languages such as Sisal. The steady-state GA has also been proposed as a type of asynchronous GA [32]. In this paradigm, only a single population of individuals is maintained at any given time. The newly generated individuals are returned to the single population by a replacement operator, which selects the individuals to be removed. This approach also introduces fundamental changes in the GA and has been known to suffer from problems such as premature convergence.

The execution time of a parallel GA can be influenced by two factors: the time taken for the execution of tasks and the time taken for communication. For coarse grained problems, the execution times are greater than the communication times and for fine grained problems, the communication times are significant. Cantu-Paz [9] showed that the performance of a fine grained GA is considerably affected by the communication time and that coarse grained GAs show a great improvement in performance with increasing paralleliza-

tion. Many optimization problems in scientific and engineering applications are computationally intensive and coarse grained in nature. Keeping these factors into account, the distributed GAs are implemented as global parallel GAs.

3.3 Distributed GA Framework

The distributed GA implementations described in this paper are implemented on a generic, custom fault-tolerant, distributed computing framework called *Vitri*. *Vitri* provides a distributed environment to solve problems by dividing the tasks among heterogeneous clusters of workstations and PCs. By defining the subproblems for a specific application, *Vitri* can be used for various approaches such as Monte Carlo simulation and simulated annealing, as well as the GA implementations described in this paper. This section provides a brief description of *Vitri*'s features.

As a result of the recent improvements in microprocessor and networking technologies, computer networks have emerged as viable alternative to supercomputers for many applications. Anderson et al. [4] note that the case for workstation clusters is stronger than ever, given (1) the growing availability of switched networks that scale well with the number of servers, (2) the extraordinary performance of modern workstations, and (3) the I/O bottleneck that makes "memory over the network" less costly than disk I/O. The suitability of distributed computing on NOWs has been demonstrated in previous studies. Skordos [31] presented an approach to simulate subsonic fluid dynamics on a cluster of non-dedicated HP Apollo workstations. Abramson et al. [1] developed a tool called Nimrod for performing parameterized simulations over a network of loosely coupled workstations. Studies such as those by Sharma and Baugh [30] and Chadha and Baugh [11] have shown that distributed computing on non-dedicated heterogeneous hardware is a practical approach in diverse application areas, including finite element analysis and vehicle routing and scheduling.

In a distributed system, parallelism is achieved by breaking up tasks into smaller tasks, assigning them to different processors, and coordinating the operations of the processors. The motivation behind developing *Vitri*, a generic distributed computing framework, is to provide an environment for solving independent subproblems in a distributed manner.

Vitri uses the *Pool of Tasks* paradigm [19] in which servers request tasks from the client's task pool, perform the necessary computations, and send the results back to the client, as shown in Figure 3.1. To maximize throughput in a heterogeneous network, load balancing is required to keep servers busy by efficiently distributing the workload. The use of a task pool is effective in achieving automatic load balancing by minimizing the idle times of servers, since faster servers request more tasks than slower ones.

In contrast to the traditional client-server approach, in which a client initiates connections with its servers, the roles of client and servers are reversed. In *Vitri*, a client waits for servers to connect, and servers may connect or disconnect at any time. This feature helps in using idle workstations as they become available by initiating server processes on them. The reversal of client and server roles also enhances fault tolerance: Whenever a server fails, the task being performed by that server is returned to the pool and subsequently requested by another server.

Vitri is an object-oriented framework that implements concurrency and distributed patterns for a set of problems. An object-oriented framework is a reusable design of a software system described by a set of generic classes and the way they collaborate. A framework provides reusability by providing hooks for the features that are problem specific. Hence, frameworks developed for applications within a specific problem domain lead to savings in cost and time invested in developing future applications.

Vitri is implemented in the Java programming language. Java sockets using the Transmission Control Protocol (TCP) over Internet Protocol (IP) are used for passing messages between clients and servers since it provides reliable connection-oriented communication. Information is passed between the client and the servers with non-blocking *writes* and blocking *reads*.

3.4 Dataflow Principles

As mentioned earlier, the ADGA implementation presented in this paper is based on the dataflow principles. Dataflow [12] is a term that refers to algorithms or machines whose order of execution is based on the availability and forwarding of data.

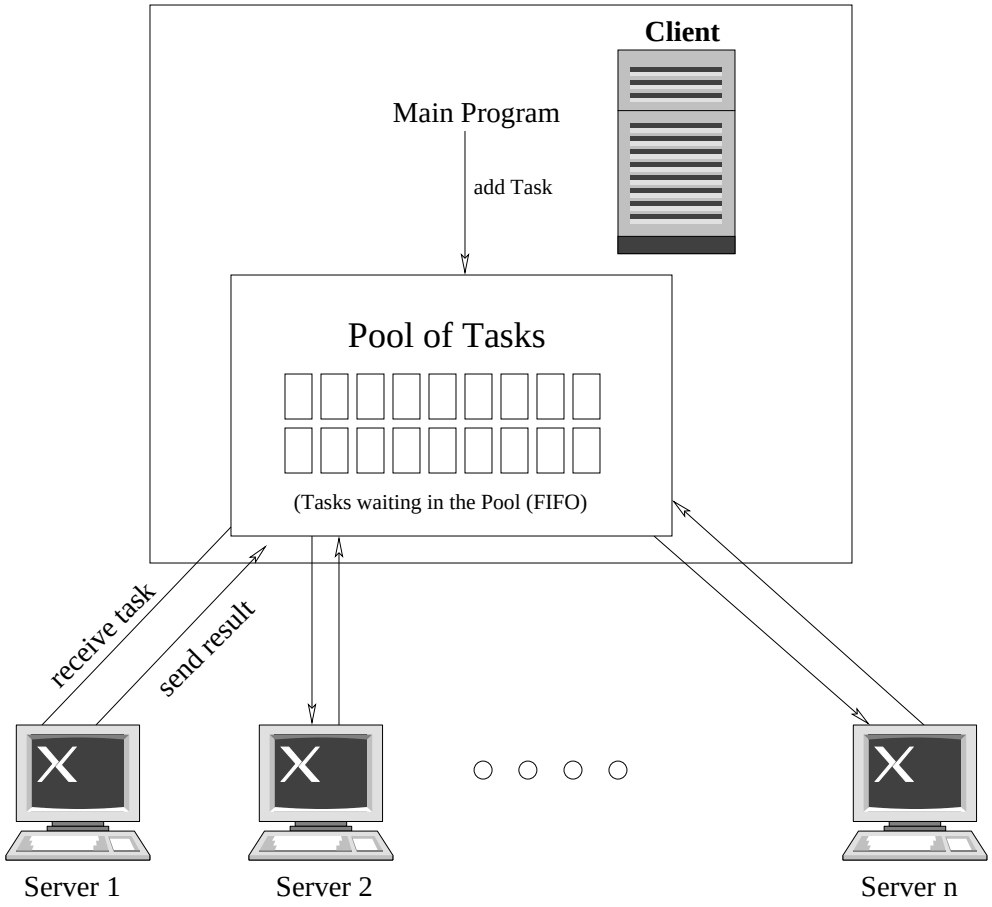


Figure 3.1: Pool of Tasks Paradigm

A dataflow program is a directed graph with nodes that represent operators and directed arcs that represent data dependencies. Nodes are computational tasks, and may be primitive machine-level instructions or arbitrarily complex functions. As a result, the dataflow model is applicable to fine- or coarse-grained parallelism. In addition to supporting varying levels of parallelism, the dataflow model also supports various types of parallelism. For instance, *vectorizing* and *pipelining* are simply special cases of standard flow graphs.

In the dataflow model, data values are carried on tokens, which travel along arcs. These arcs are first-in, first-out (FIFO) queues of unbounded capacity. The status of nodes can be determined by a simple firing rule: A node is said to be *firable* when the data it

needs are available. When a node is fired, its input tokens are absorbed. The computation is performed and the result is sent to its output arcs for other nodes to use. There is no communication between tasks—each task simply receives and outputs data.

The dataflow model has the following properties [5]:

- *parallelism*—nodes may execute in parallel unless there is an explicit data dependence between them;
- *determinacy*—results do not depend on the relative ordering in which nodes execute.

The natural parallelism in dataflow occurs because it does not force over-specification of an algorithm. The firing rule only says when a node *can* fire. It does not require that it be executed at any particular time. Controlling node execution is generally based on one of two schemes: *data-driven* or *demand-driven*. In the data-driven approach, the results of a computation at a node are immediately made available to its output arcs (and hence other nodes). Each node is fired as its input becomes available until all possible firings have occurred. In contrast, a demand-driven scheduler waits until a request is made for data. If the data are available, they are simply returned; otherwise the request propagates backward through the graph, causing the execution of nodes that will fulfill the initial request of data. Thus, data-driven execution is like demand-driven execution where all data have been requested.

3.4.1 Control Flow in a Global Parallel GA

Typically the steps involved in a GA are as follows:

- Start with a randomly generated population, select individuals in the next generation by using some selection algorithm.
- Perform genetic operators such as crossover and mutation to simulate genetic variation in the population.
- These steps are continued until a specified termination condition is met.

The selection scheme used in the SDGA is a competition based process referred to as "tournament selection." In this scheme, two randomly selected individuals are chosen and fitter of the two is chosen as the winner. To generate a new population from an old one, the

selection is done $P/2$ times, where the population size is P. The crossover and mutation operators are then applied to each of the P individuals and the whole process is continued.

The control flow in the SDGA is that of an imperative program, where data dependencies are implicit. In an SDGA, the program proceeds to the next generation only after all evaluations in the current generation are finished. These synchronizations at the end of each generation cause the speedup gained by parallel processing to be limited by the slowest server. To remove this data dependency, we present a new asynchronous distributed GA (ADGA) that combines loop unrolling with the asynchronous instruction scheduling implemented as a dataflow graph. The asynchronous approach performs precisely the same computations as the synchronous one except that the data dependencies between generations are eliminated.

3.4.2 Using Dataflow for Asynchronous Genetic Algorithms

The ADGA is implemented by unrolling the loops in a typical GA and representing the synchronous instruction scheduling as a dataflow graph. The numerical computations in a GA remain the same in the asynchronous implementation. The individuals in a population are generated as a result of applying the genetic operators to the members of the previous population. As a result, creation of a specific individual is dependent only on the individuals from which the selection is made. In the ADGA implementation, the arcs are implemented as "in-place buffers". Further, the ADGA uses a data driven implementation.

This precedence relationship is represented as a dataflow graph in Figure 3.2. This figure shows the overall GA process with four generations. For simplicity, a population size of 10 is used in the Figure. In the Figure 3.2, $G1$, $G2$, etc., represent different generations. The boxes $D11$, $D12$, etc., represent the dataflow graphs. In a particular generation, there are $P/2$ such graphs, where the population size is P. The solid boxes (such as $D13$) represent graphs that are yet to produce output individuals. The inputs to each dataflow graph are the individuals that will be used in the genetic operations. For instance, the input to the dataflow graph $D11$ are individuals 2, 5, 8, and 1 from generation 1 ($G1$). The shaded ovals denote the availability of individuals in the corresponding generation. For instance, the positions 5 and 6 in generation 2 are yet to be filled by the computations of graph $D13$

and hence appear vacant.

Each of the dataflow graphs consists of four *Copy* nodes, four *Evaluate* nodes, four *Compare* nodes, two *Compete* nodes and one *Mate* node. For example, Figure 3.3 shows one of the dataflow graphs produced. Individuals in a population are randomly selected and are inputs to the *Copy* nodes. Once the individuals become available for selection, the *Copy* nodes get fired. The *Evaluate* nodes evaluate the fitness of the organisms. The *Compare* nodes compare each individual to the best solution. The selection process actually takes place in the *Compete* nodes. The selected individuals are inputs to the *Mate* node, where the genetic operations are carried out. The outputs of *Mate* nodes are the new individuals in the next generation. Each of the $P/2$ dataflow graphs in a generation and the nodes within each of them are executed in parallel. In a SDGA, a slow server might impede the progress of the GA by making faster processors wait until it finishes its fitness evaluations. Since the *Copy* nodes in each generation get fired as soon as their inputs become available, the processes in the new generation need not wait until the old generation is complete. This approach to introducing asynchrony helps to exploit maximum parallelism by eliminating synchronizations between generations. Since the basic structure of the GA remain the same, the convergence properties of the GA remains the same as that of a synchronous approach.

It can be seen that the calculations in subsequent generations proceed in ADGA as soon as the data depedencies are satisfied. Figure 3.4 displays the number of "active" generations when a given generation is still evaluating organisms. The individuals in a certain generation are created by the dataflow graphs in the previous generation. When all the individuals in a certain generation are created, the number of active subsequent generations is determined. This value is considered to be the number of active generations. Figure 3.4 shows that when all the individuals in generation 1 are created, there are 3 active subsequent generations. A sample distribution of the organisms at a given instant is shown in Figure 3.5. It shows that when all the individuals in generation 1 are created, approximately half of the individuals in generation 2 have been created, along with some individuals in generation 3 and 4. These plots give a measure of the amount of unrolling occuring in the ADGA.

3.5 Theoretical Analysis of Distributed GAs

There are many different ways to characterize the performance of a parallel algorithm [15]. The total execution time is a typical measure used to evaluate the performance of an algorithm. However, as the execution times vary with problem sizes, execution times must be normalized when comparing the performance of an algorithm at different problem sizes. Speedup and efficiency of an algorithm are such performance metrics. In this paper, a theoretical model to predict parameters such as execution time, speedup and efficiency of SDGA and ADGA is presented.

During the execution of the GA, each server spends some time in communicating with the client, some time in executing a task and some time it remains idle. The computing time normally depends on factors such as problem size, characteristics of the server, etc. The communication time normally depends on the physical bandwidth of the communication medium linking the client and the server and on the amount of data being transmitted. The idle times are often caused by the lack of available tasks for the servers. The availability of tasks is dependent on the relative ordering in which servers perform the tasks. In *Vitri*, the individuals of a particular population are placed in the "pool of tasks" to be evaluated and the servers pick up tasks from the pool. The idle times are relatively small when tasks are instantly available to the servers for evaluation. As the end of a generation approaches in a SDGA, the number of tasks in the pool decreases. If a slow server happens to be evaluating the last task in a generation, all the servers will wait for the slow server to complete its evaluation before proceeding to the next generation. Thus, slow servers can act as bottlenecks and considerably slow down the execution.

The performance model considered here predicts the performance metrics for the distributed GA implementations as a function of the number of generations (G), population size (P), total number of processors used (N), the average execution time (t_{comp}) for a single fitness evaluation and the average time for communication (t_{comm}) with the client for the execution of a single task. The theoretical model also predicts the performance metrics when a homogeneous as well as heterogeneous set of computers are used.

The performance metrics – execution time, speedup, and efficiency – are defined

as follows: The total execution time of a distributed GA is the total time taken for a GA execution. The total execution time (T_1) for a sequential GA can be estimated as:

$$T_1 = P\,G\,t_{comp} \tag{3.1}$$

T_N is used to denote the total execution time for a distributed GA with N servers. Efficiency can be considered as the fraction of time that processors spend doing useful work. Efficiency provides a measure of the effectiveness with which algorithms use the computational resources. Efficiency is defined as:

$$e = \frac{T_1}{N\,T_N} \tag{3.2}$$

The speedup, which is the factor by which execution time is reduced by using N processors, can be defined as :

$$s = Ne \tag{3.3}$$

3.5.1 Homogeneous System of Servers

In this case, a network of computers consisting of N identical processors is considered. In a GA, for a single generation to complete, P organisms are evaluated. It is assumed that all of the N processors start simultaneously, and that each of them takes t_{comp} to execute a fitness evaluation and t_{comm} for communication with the client. The GA execution can be assumed to be taking place as a number of blocks, with each block representing the set of tasks performed by the N servers as shown in Figure 3.6. The pattern of blocks repeats itself each generation. At the end of a generation, assume that there are only n tasks left to be evaluated and as a result, $N - n$ processors will be idle at the end of each generation. Since in each block N tasks get evaluated, the total number of blocks in a generation is equal to $\lceil \frac{P}{N} \rceil$. From the figure, the time taken for a single generation (T_g) and the total time taken for the SDGA (T_{sync}) can be estimated as:

$$T_g = \lceil \frac{P}{N} \rceil (t_{comp} + t_{comm}) \tag{3.4}$$

$$\begin{aligned} T_{sync} &= T_g G \\ &= \lceil \frac{P}{N} \rceil (t_{comp} + t_{comm}) G \end{aligned} \tag{3.5}$$

The predicted efficiency and speedup can be written as:

$$e_{sync}^{hom} = \frac{Pt_{comp}}{\lceil \frac{P}{N} \rceil N(t_{comp} + t_{comm})} \tag{3.6}$$

$$s_{sync}^{hom} = \frac{Pt_{comp}}{\lceil \frac{P}{N} \rceil (t_{comp} + t_{comm})} \tag{3.7}$$

In the case of an ADGA, the servers are not constrained by the lack of available tasks at the end of a generation, assuming that tasks in subsequent generations can be evaluated. The total number of tasks in an ADGA evaluation is PG. Since there are N servers the total time taken by the ADGA (T_{async}) can be written as

$$T_{async} = \frac{PG(t_{comp} + t_{comm})}{N} \tag{3.8}$$

Neglecting the time lost in communication, the efficiency and speedup can be estimated as:

$$e_{async}^{hom} = \frac{t_{comp}}{(t_{comp} + t_{comm})} \tag{3.9}$$

$$s_{async}^{hom} = N\frac{t_{comp}}{(t_{comp} + t_{comm})} \tag{3.10}$$

3.5.2 Heterogeneous System of Servers

To simulate a heterogeneous system, n_s identical slow servers are introduced in a system of N servers. For simplicity, it is assumed that the $(N - n_s)$ fast servers are also identical. Further, each of the n_s processors is assumed to be slower than the each of the $(N - n_s)$ fast ones by a factor f. Similar to the homogeneous case, the SDGA execution takes place as a pattern of recurring blocks. The cumulative value of t_{comp} and t_{comm} is considered in this model to simplify the task-ordering model. A "block" in this case is considered to be one which corresponds to the execution of a slow server (Figure 3.7). t and t^{slow} represents the sum of t_{comp} and t_{comm} for a fast and slow server, respectively. Figure 3.7 shows a sample execution pattern with f being 4. It can be observed that the total time taken to complete a generation is clearly influenced by the slow server. In the time a slow server does one evaluation, a fast server does 4 evaluations. The number of blocks in a generation can be estimated as:

$$nb = \lceil \frac{P}{f(N - n_s) + n_s} \rceil \tag{3.11}$$

Depending on the ordering of tasks, the number of tasks that remains at the end of generation becomes important. The number of tasks present in the final block in a generation (δ_1) can be estimated as:

$$\delta_1 = (P - (nb - 1)(f(N - n) + n)) \tag{3.12}$$

If there are more tasks in the last block of a generation than the number of fast servers, the slow servers will receive tasks to evaluate. Taking these factors into account, the execution times can be estimated as:

$$T_{sync} = \begin{cases} (nb-1)f\,t\,G + t\,G & \text{if } \delta_1 \leq (N - n_s) \\ nb\,t\,f\,G & \text{otherwise} \end{cases} \tag{3.13}$$

Since the end-of-generation synchronizations are eliminated in an ADGA, the whole GA execution can be considered to be the ordering of PG tasks among the processors. The number of blocks is estimated as:

$$nb = \lceil \frac{PG}{f(N - n_s) + n_s} \rceil \tag{3.14}$$

At the end of the GA execution, if the last block in the GA execution contains more tasks than the number of fast processors, the slow processors will be involved in the end-of-GA computations. The number of tasks present in the final block of GA execution δ_2 can be estimated as:

$$\delta_2 = PG - (nb - 1)(f(N - n_s) + n_s) \tag{3.15}$$

The estimated time of ADGA can be estimated as:

$$T_{async} = \begin{cases} (nb-1)ft + t & \text{if } \delta_2 \leq (N - n_s) \\ nb\,t\,f & \text{otherwise} \end{cases} \tag{3.16}$$

The speedup and efficiencies can be estimated in the same way as explained in the foregoing section.

3.6 Results and Analysis

The distributed GA implementations are applied to two different problems to analyze the performance of the algorithms, namely the 0/1 knapsack problem from the OR litera-

ture [24] and an air quality management problem. The knapsack problem and the air quality optimization problem are used as examples of fine grained and coarse grained problems, respectively. The results are presented comparing the run time and efficiency of each distributed implementation, varying the number of processors. All the studies have been conducted with varying number of workstations consisting mainly of Sun Sparc5 and ULTRA10 processors running Solaris 2.6. The results are compared with a theoretical model presented above.

3.6.1 0/1 Knapsack Problem

The 0/1 knapsack problem is representative of the large class of problems known as combinatorial optimization problems. In the knapsack problem, the objective is to maximize the obtained profit without exceeding the capacity of the knapsack. The problem can be described mathematically as follows:

$$\text{Maximize} \quad P(\mathbf{x}) = \sum_{j=1}^{n} x_j p_j \tag{3.17}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} w_j x_j \leq C \tag{3.18}$$

where P is the total profit associated with the knapsack, p_j is the profit of placing item j in the knapsack, w_j is the weight of item j, and C is the capacity of the knapsack, and $\mathbf{x} = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ such that $x_j = 1$ if selected and $= 0$ otherwise, and n is the number of available items. The knapsack problem is a fine grained problem since the fitness evaluation is comparatively a less costly operation. The communication times in a distributed GA execution for the knapsack problem would be significant compared to the computation times.

3.6.2 Air Quality Optimization

The second problem considered in this paper is used to analyze the distributed GA implementations is an air quality management problem. Tropospheric ozone formed from the emissions of vehicles and industrial sources is considered a major pollutant. Ozone is a major component of photochemical smog and is responsible for many respiratory irritations

and damage to vegetation. GAs can be used to identify cost effective strategies for meeting target ozone levels. When exploring control strategies for a region with hundreds of sources, each in turn having thousands of processes, it is impractical to search the entire feasible region enumerating each strategy. GA-based optimization provides a means for efficiently searching the decision space for potential control strategies. An ambient least cost (ALC) model [6] is an optimization formulation that incorporates source marginal control costs and emission dispersion characteristics to compute the source emissions at the least cost. The available models to predict the complex transport and chemistry range in sophistication. The Empirical Kinetic Modeling Approach (EKMA), a Lagrangian box model, can be used to obtain approximate ozone concentrations. The ALC optimization can be performed using EKMA as the air quality model. This problem is coarse grained since each evaluation of fitness requires the execution of EKMA and for this problem, the computational cost is significantly greater than the communication costs. The case studies involve air quality optimization with EKMA for an urban region around Charlotte, North Carolina.

3.6.3 Scalability Issues with distributed GAs

Scalability analysis of a distributed algorithm requires two specifications: characteristics of an algorithm and characteristics of the distributed environment. Cantu-Paz and Goldberg [10] describe the scalability issues associated with parallel GAs. In this discussion, the factors limiting the performance of a global parallel GA is explored. The scalability issues are addressed by two measures:

- *Size* scalability defines the scalability of the system with an expectation of linearly increased performance with incremental expansion of the number of processors.
- *Problem* scalability defines a measure of the speedup with increase in problem size.

The scalability issues associated with synchronization strategies in a global parallel GA are addressed in this paper. Analyzing the scalability of an algorithm means predicting its potential elapsed times for varying inputs. The scalability analysis also help in capturing the critical parameters affecting the performance and their impact on the overall elapsed

time. In the following sections, the performance of the algorithm will be examined for the two types of scalability discussed above.

3.6.4 Empirical Results

Knapsack Problem

The SDGA and ADGA implementations are used to solve the 0/1 knapsack problem with the number of processors varying from 3 to 30. To study the effect of problem scalability, the fitness evaluation times are artificially varied to achieve four different levels of granularity. The ratio of t_{comp} to t_{comm} is considered as a measure of problem granularity. t_{comm} is measured to be approximately 250 milliseconds and so the t_{comp} times are artificially set to be 250, 500, 750 and 1000 milliseconds to create four levels of granularity. The GA runs are conducted for a population size of 100 for 200 generations. To simulate a heterogeneous system, a slow server is introduced with f being 5.

Figure 3.8 shows a typical plot of the execution times of the GAs with increasing number of processors for a fixed problem size, for both homogeneous and heterogeneous systems. The typical speedup and efficiency curves are shown in Figures 3.9 and 3.10, respectively. It can be observed from the graphs that the measured values are close to the values predicted by the theoretical model. The execution times of SDGA follow a step function pattern implying that in between each step, there is no reduction in the execution time for using a higher number of processors, also implying a decrease in efficiency. The same behavior is exhibited by the speedup curves for SDGA.

The use of a higher number of processors lead to increasing gains in execution times for both GAs. It can also be observed that the SDGA performance is considerably affected by the heterogeneity in the system, whereas the ADGA is not severely affected. The speedups and efficiencies of both GAs decrease in going from a homogeneous to a heterogeneous system. The decrease is marginal in case of the ADGA, whereas the SDGA parameters are reduced significantly by the heterogeneity.

The effect of increasing the problem granularity is examined by analyzing the performance parameters with increase in problem size. As expected, the execution times

increase significantly with increase in problem granularity as shown by the Figure 3.11. The trends in speedup and efficiencies with increasing granularity are shown in Figures 3.12 and 3.13. The efficiency and speedups of both GAs increase with increase in problem size. These figures illustrate the increased impact of heterogeneity on SDGA performance compared to that of ADGA. Further, the difference in performance of ADGA and SDGA increases with increasing problem granularity.

Air Quality Optimization

In this section, the GA implementations are applied to the air quality management problem discussed earlier. The study included with a network of workstations (1-19). In each problem, the GA was run for 50 generations using a population size of 50. This problem can be considered to be fairly coarse grained since the t_{comm} is small compared to t_{comp}. Hence, t_{comm} is ignored in the analysis presented below. To simulate a heterogeneous system, a slow processor with f factor 5 is used.

The performance parameters, the execution time, speedup, and efficiency, measured for both homogeneous and heterogeneous systems are found to be in close agreement with the values predicted by the theoretical model. Figures 3.14 to 3.16 compares the effect of introducing a slow server on the performance of SDGA and ADGA. Similar to the trends observed with the knapsack problem, the SDGA is considerably slowed by the introduction of a slow server, whereas the performance of the ADGA is only marginally affected by the heterogeneity (Figure 3.14). With an increase in the number of processors, the ADGA in a heterogeneous system even outperforms the corresponding SDGA in a homogeneous system. This fact is also evident from the Figures 3.15 and 3.16, where the ADGA in a heterogeneous system shows better speedup and efficiencies than the corresponding ADGA in a homogeneous system, as the number of processors increases.

The EKMA model used in the above analysis is a simple photochemical model to predict the ozone formation. A Eulerian numerical model called Urban Airshed Model (UAM) incorporates more complex chemical reactions and dispersion equations than EKMA and is computationally more intensive. A single simulation of UAM requires approximately 20 minutes on a Sun ULTRA10 workstation. The ALC optimization with a regulatory-

scale model such as UAM is an extremely coarse grained problem. The distributed GAs are applied to the above case study for the Charlotte region. The GA runs were made with a population size of 100, up to 90 generations, using only 3 Sun ULTRA10 machines. A comparison of the computational times are presented in Table 3.1. These runs were conducted with only relatively minor down times of the servers. It can be observed that the ADGA provides an improvement of approximately 10% with only three processors. With the use of a higher number of processors, the substantial improvements in performance can be achieved as evident from the study using EKMA.

Table 3.1: Computational times for the UAM runs

	Time taken by Sequential GA (estimated)	Time taken by SDGA (measured)	Time taken by ADGA (measured)
time (months)	4.06	1.735	1.551

3.6.5 Scalability Analysis

Scalability with Fixed Problem Size

One approach to quantifying scalability is to determine how the execution time and efficiency vary with an increase in the number of processors. This analysis helps in determining parameters such as the optimum number of processors required to solve a specific problem within a certain time, efficiency of using a certain set of computers, etc. In some cases, there are no improvements in efficiency or the total execution time for an increase in the number of processors. As a result of the scalability analysis, the number of processors for a specific platform which provides the best efficiency can be determined. In contrast to the SDGA, the asynchronous speedup curves display more linearity, implying that an increase in number of processors contributes directly to improvements in performance. From the graphs representing the efficiencies, it can be seen that using a larger number of processors with SDGA may not necessarily provide a higher efficiency and can actually lead to lower

efficiencies.

The following observations about the performance of the distributed GAs can be made:

- The total execution times decrease with N and it tends to get bounded when the number of servers increases to a point when availability of tasks gets limited. The SDGA is also constrained heavily by the way tasks are scheduled. For a SDGA in a heterogeneous system, the scheduling of tasks is dependent on how the slow servers finish their evaluations. As a result, the slow servers limit the speedup that can be gained by using a larger number of fast processors.
- Speedup increases with N. However, the speedup curves for SDGA exhibit a step function behavior similar to that of the execution times as explained above.
- The efficiencies are mainly dependent on N and P. They do not display a monotonically increasing or decreasing trend. However, the ADGA exhibits higher efficiencies compared to the SDGA. This type of behavior underlines the importance of scalability analysis in choosing the parameters for optimum performance.
- An algorithm is considered highly scalable if the amount of computation scales with the number of processors to keep the efficiency constant. From the efficiency curves, it can be observed that for the ADGA with a homogeneous set of processors, the efficiencies remain constant, implying that the algorithm is highly scalable. Also, in the heterogeneous cases, efficiencies tend to become constant when higher number of processors are employed. In case of the SDGA, it can be seen that efficiencies tend to decrease with the number of processors.

Scalability with Scaled Problem Size

Another factor in considering the performance of an algorithm is the problem size under consideration. Parameters such as P, G, t_{comp}, t_{comm} can be considered as parameters characterizing problem size. Similar to the analysis with fixed problem size, the analysis with a scaled problem size helps in analyzing the tradeoffs in performance for scaled problem sizes.

The following observations can also be made for problem scalability of the distributed GAs.

- The total execution times increase with P, G, t_{comp} and t_{comm}. This is equivalent to the fact that with increase in problem size, the total execution time also increases.
- The speedup and efficiencies increase with problem size for both SDGA and ADGA, and ADGA scales better.

These observations provide insights into the behavior of both types of GAs and help the user in the choice of the algorithm parameters for a particular problem. From the analysis, the performance characteristics such as the execution times, speedups and efficiency of a particular implementation on a specific platform can be predicted. The tradeoffs in performance with changes in parameters such as the number of processors, problem size etc., can also be determined from the scalability analysis.

3.7 Conclusions

In all our results, number of generations has been used as the termination condition for GAs. In both SDGA and ADGA, the computations performed are very similar although their timings are offset. The solution quality from both types of GAs are quite close to each other. The results presented in this paper demonstrate that the distributed GA implementations can be used to gain considerable improvements in performance over a sequential GA. The ADGA provides additional gains in performance with the elimination of end-of-generation latencies associated with the SDGA. The results show that ADGA is highly suitable for problems that are highly coarse grained such as the air quality optimization problem using UAM, and when a large number of servers are available. Further, performance of the SDGA suffers in a heterogeneous network and the ADGA performs much better than the SDGA.

Using the theoretical model presented in this paper, it can be seen that an appropriate choice of parameters for the best performance of SDGA can be determined. For example, for a given number of processors, the optimum population size that provides the best computational performance for SDGA can be calculated. However, in the case of

ADGA, all the available servers can be utilized without having to compromise the performance. In a non-dedicated network, where the availability of servers throughout the execution is not guaranteed, the SDGA performance could suffer if some of the servers are removed in the middle of an execution. The ADGA is especially suitable in such a situation because of its dynamic ordering of tasks.

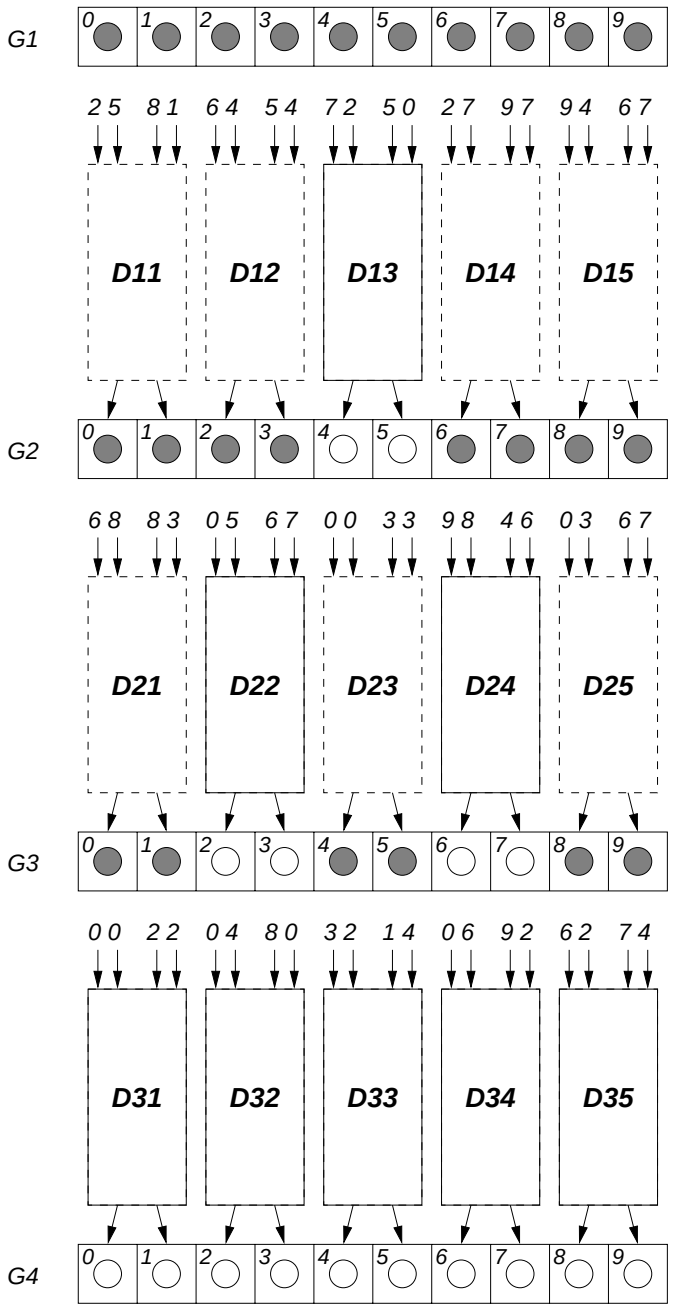


Figure 3.2: Dataflow Graphs Dynamically Unfolding

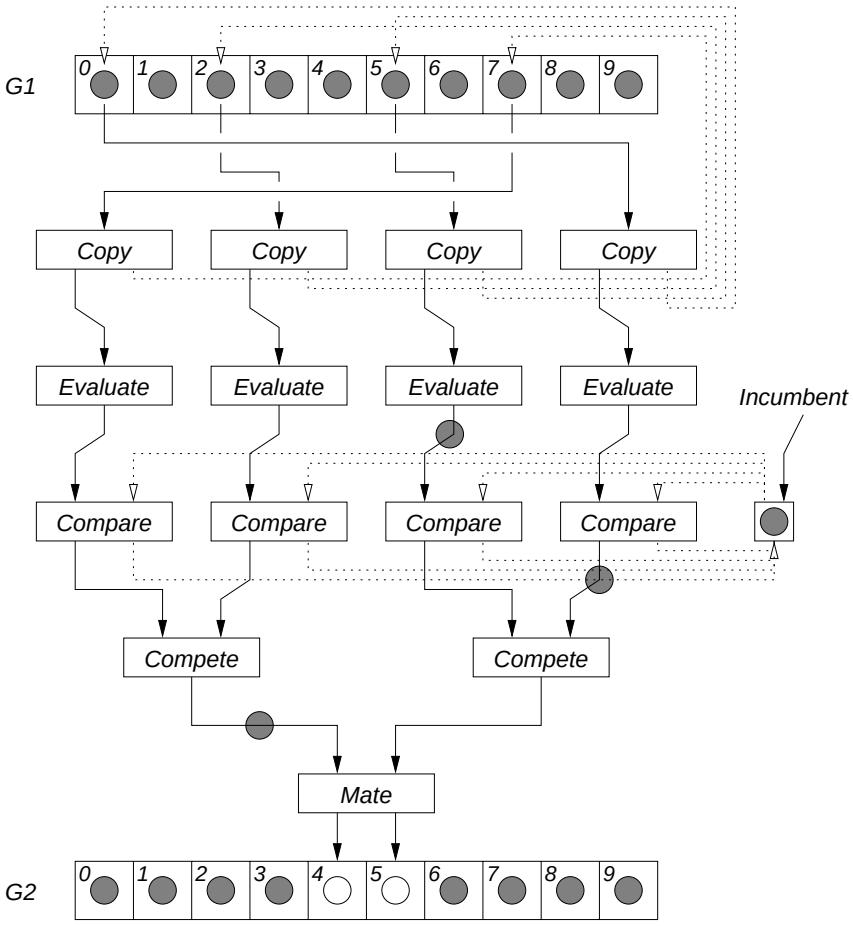


Figure 3.3: Details of Dataflow Graph *D13*

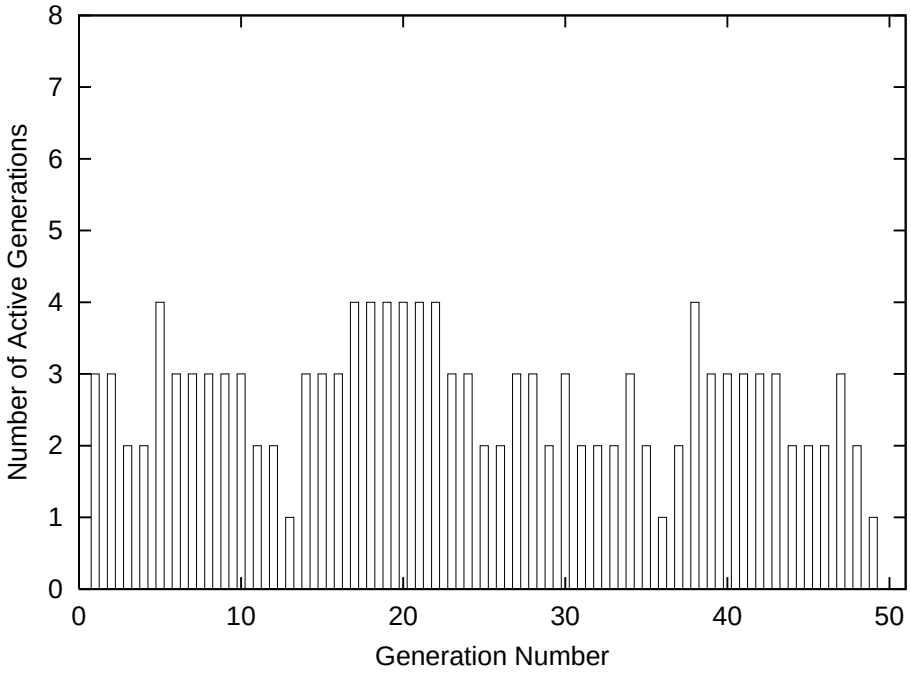


Figure 3.4: Unrolling of Subsequent Generations in ADGA

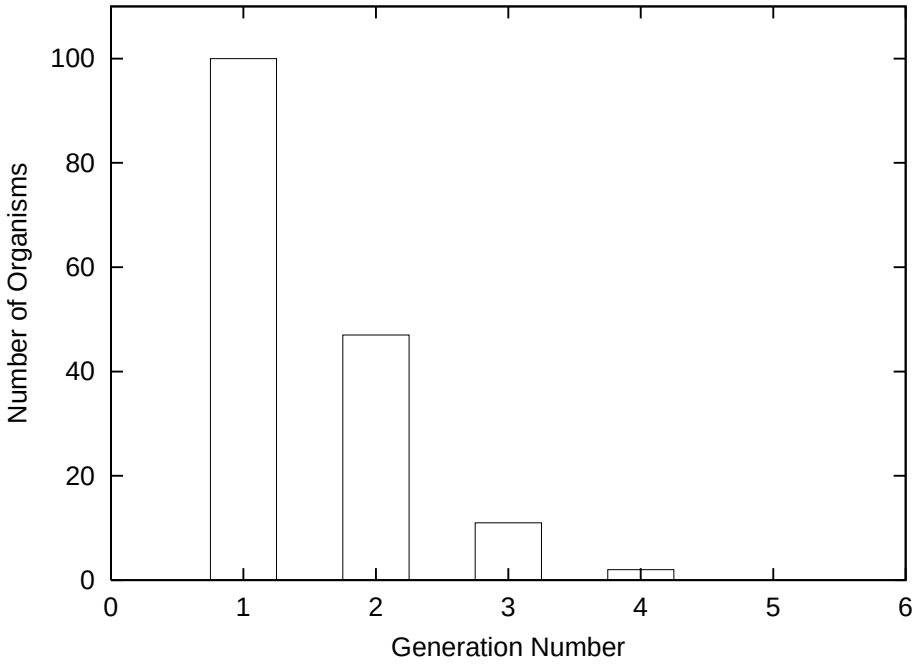


Figure 3.5: A Sample Distribution of Organisms at a Given Instant in ADGA

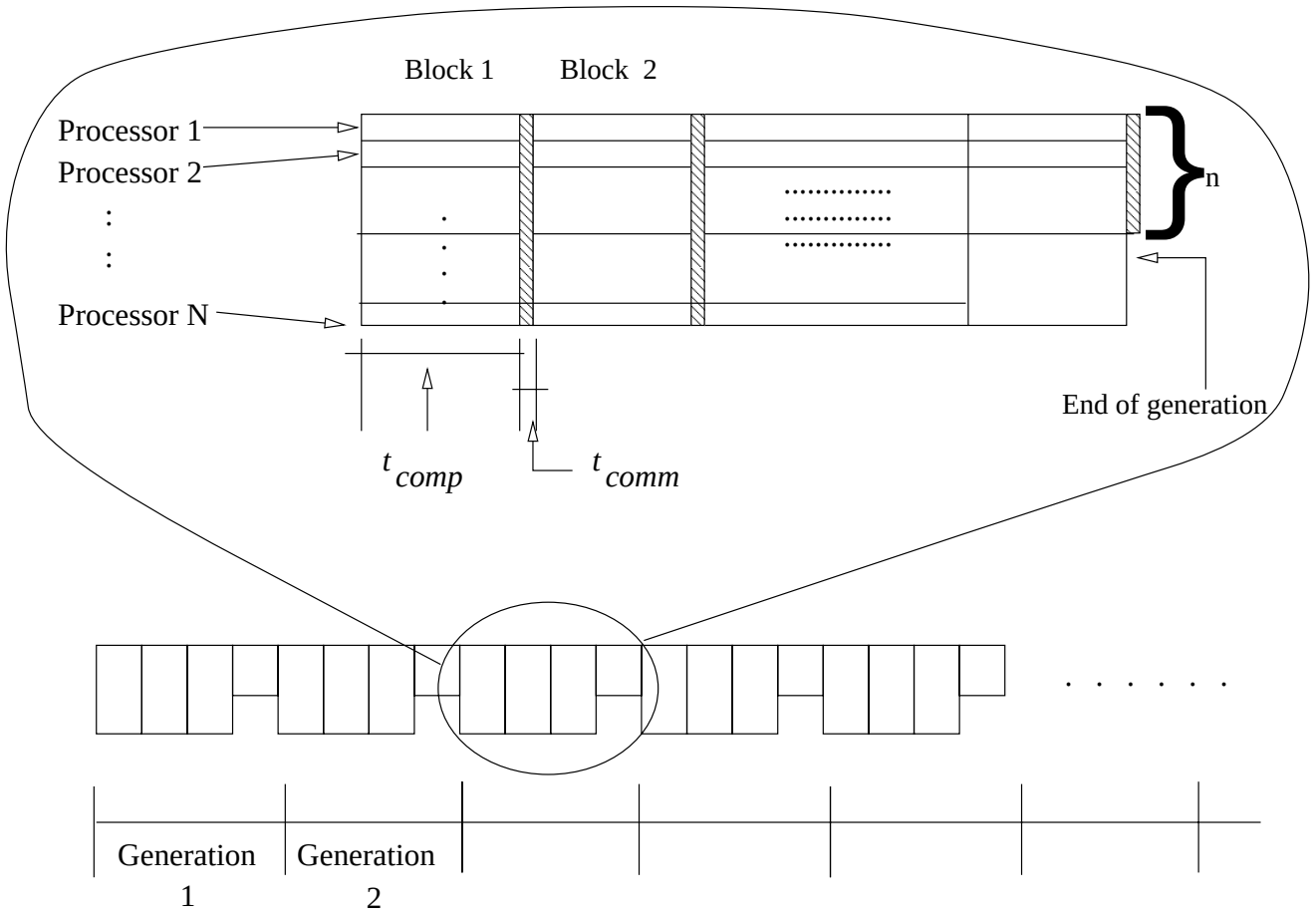


Figure 3.6: Distribution of Tasks in a Generation for the SDGA

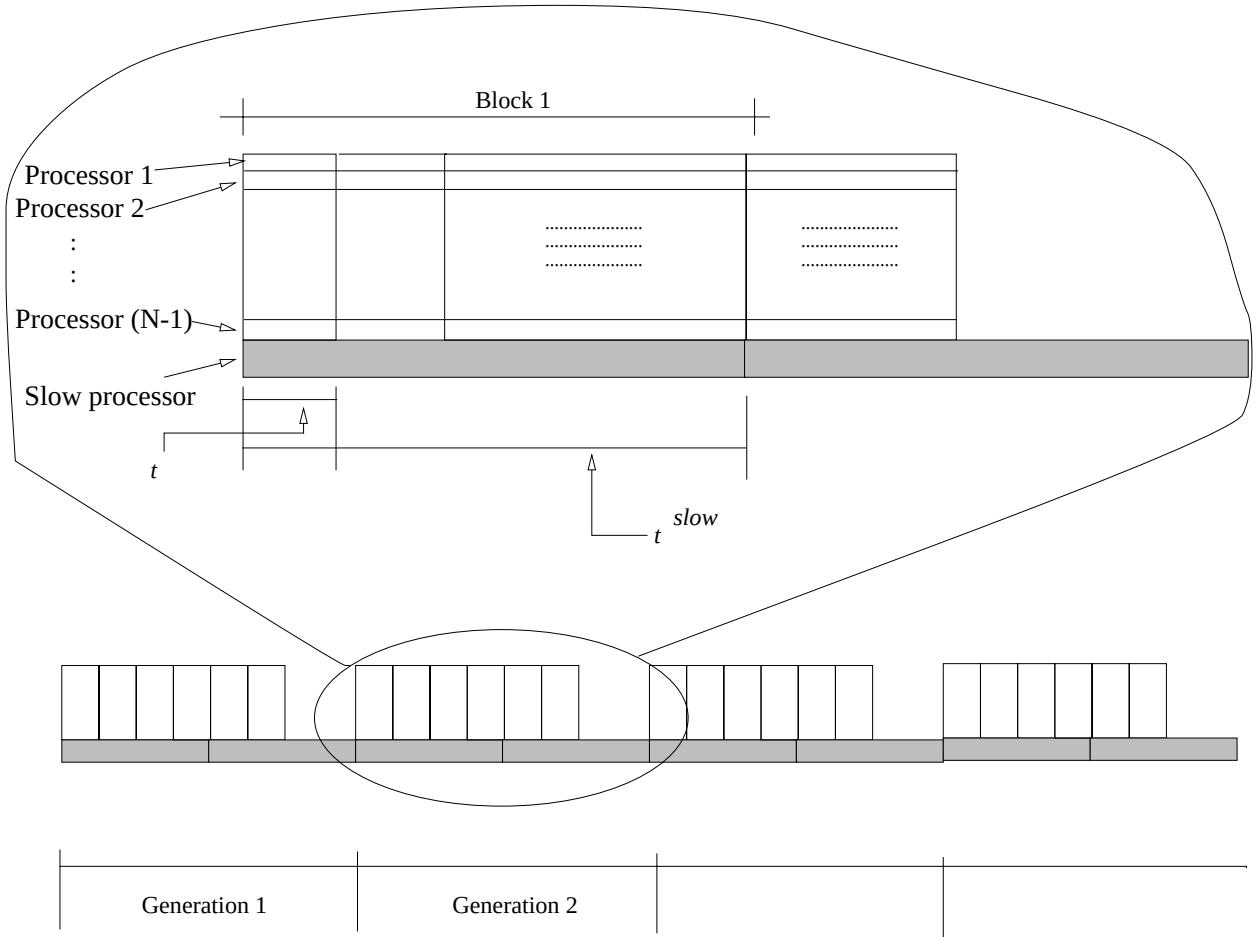


Figure 3.7: Distribution of tasks in a generation for SDGA for a heterogeneous system

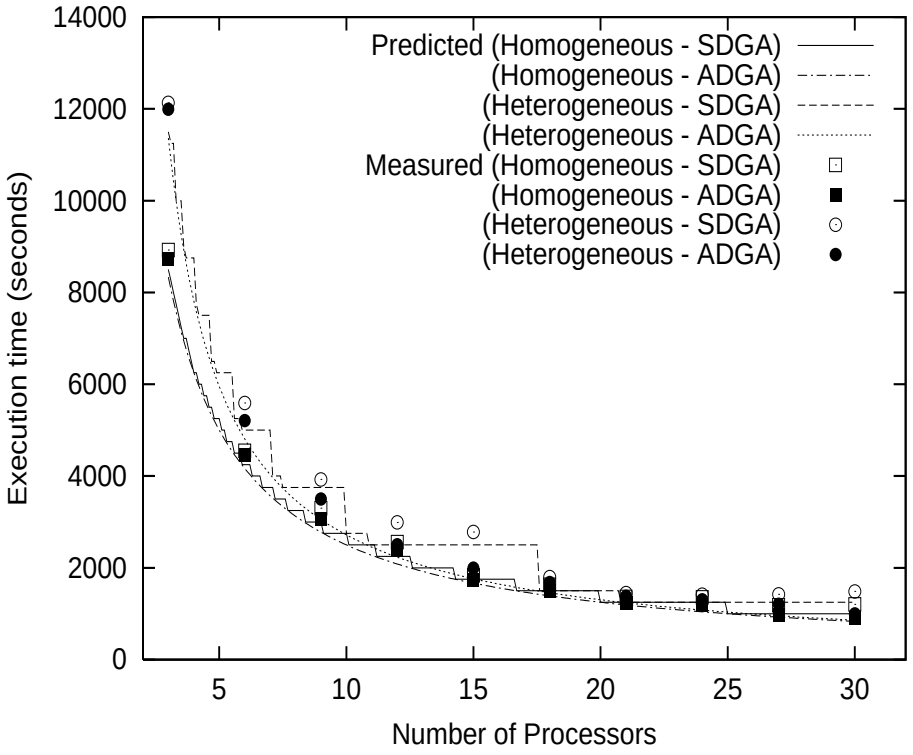


Figure 3.8: Comparison of measured execution time with the predicted values for SDGA and ADGA for the 0/1 knapsack problem

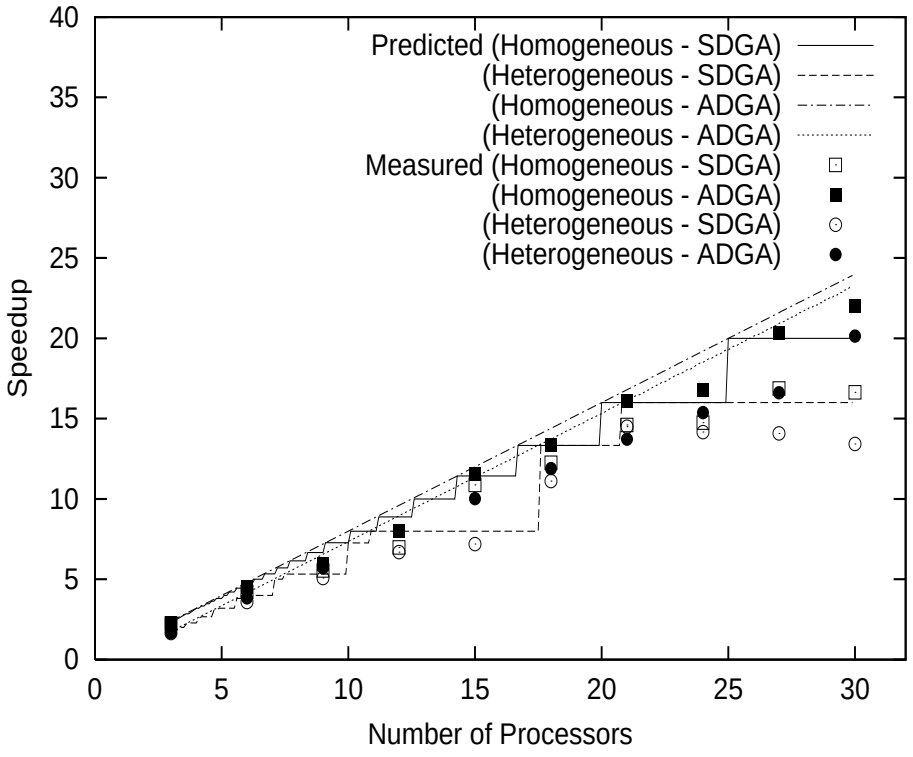


Figure 3.9: Comparison of measured speedups with the predicted values for SDGA and ADGA for the 0/1 knapsack problem

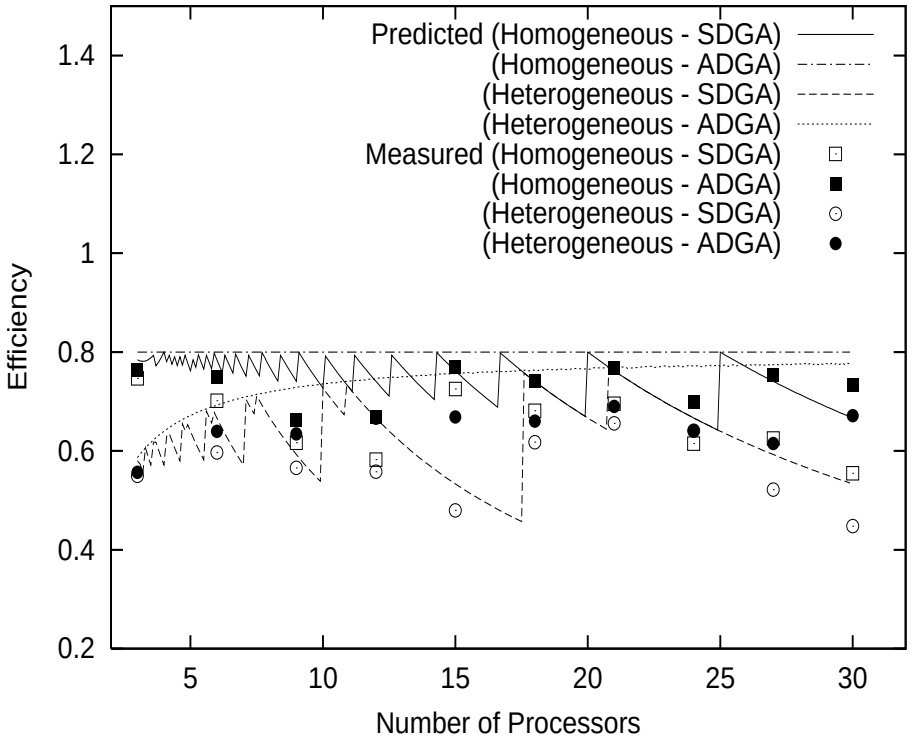


Figure 3.10: Comparison of measured efficiencies with the predicted values for SDGA and ADGA for the 0/1 knapsack problem

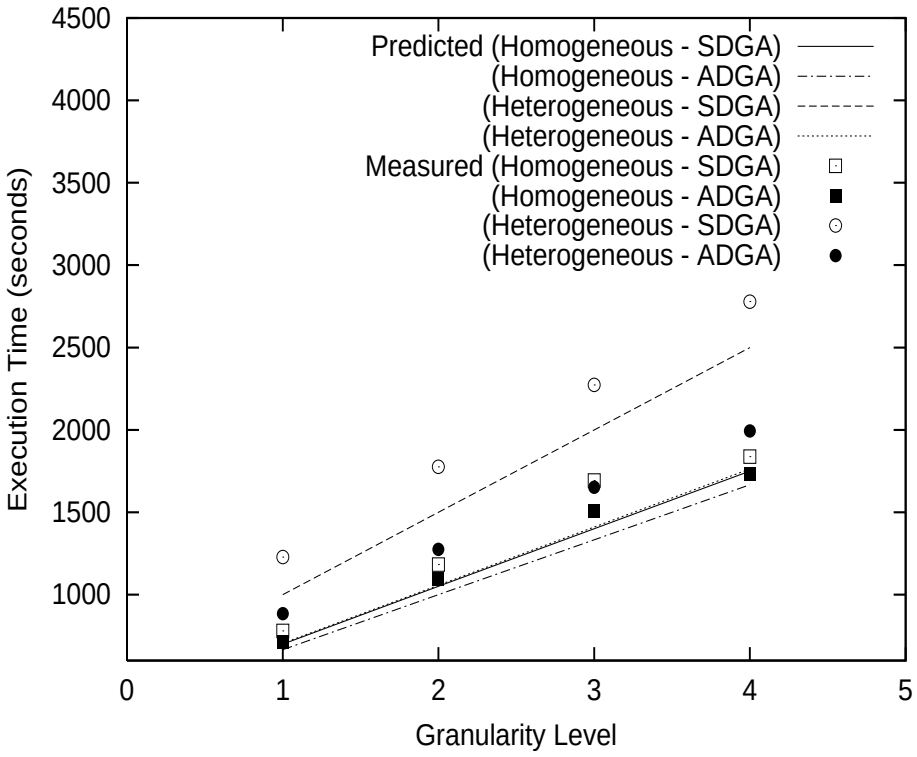


Figure 3.11: Execution times of ADGA and SDGA with increase in problem granularity for the knapsack problem. The sample curve is shown for the performance with number of processors = 15.

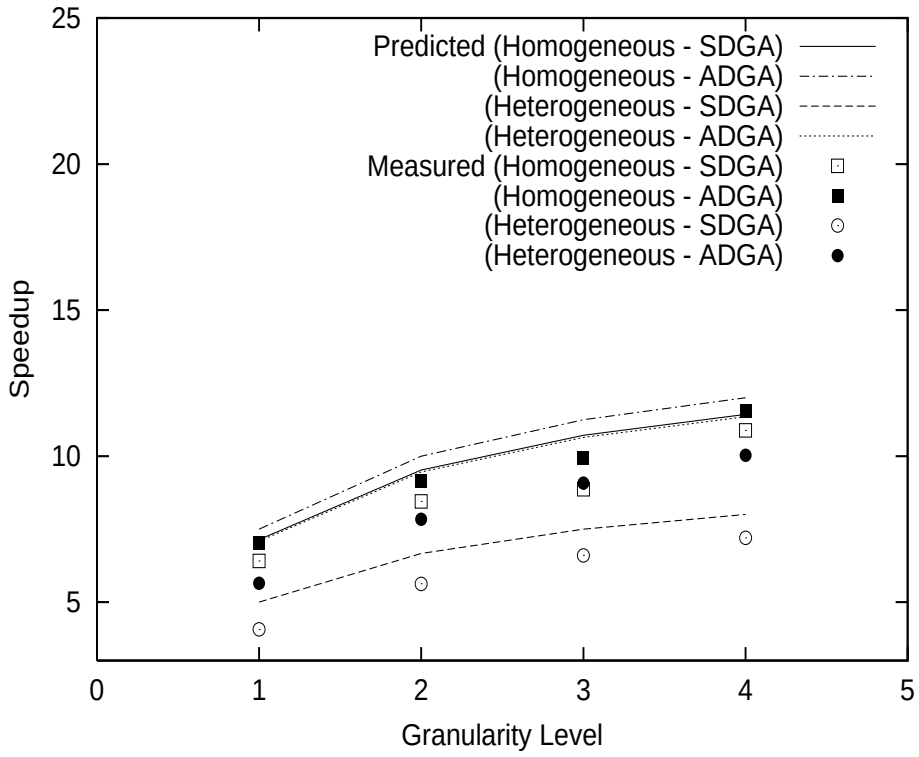


Figure 3.12: Speedups of ADGA and SDGA with increase in problem granularity for the 0/1 knapsack problem. The sample curve is shown for the performance with number of processors = 15.

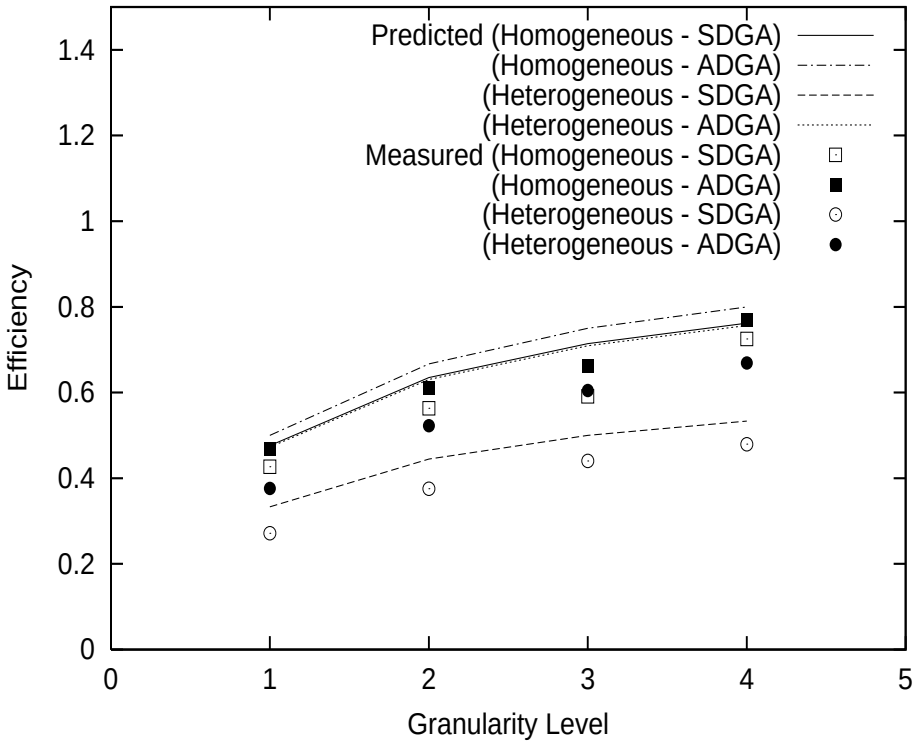


Figure 3.13: Efficiencies of ADGA and SDGA with increase in problem granularity for the knapsack problem. The sample curve is shown for the performance with number of processors = 15.

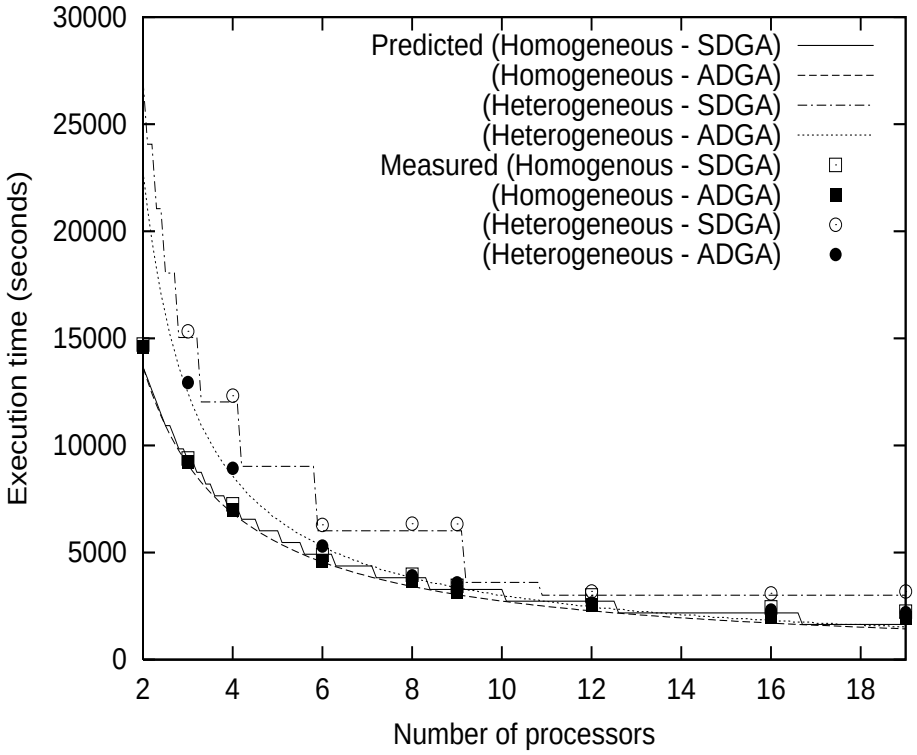


Figure 3.14: Comparison of the execution times of the homogeneous and the heterogeneous systems for the air quality optimization problem

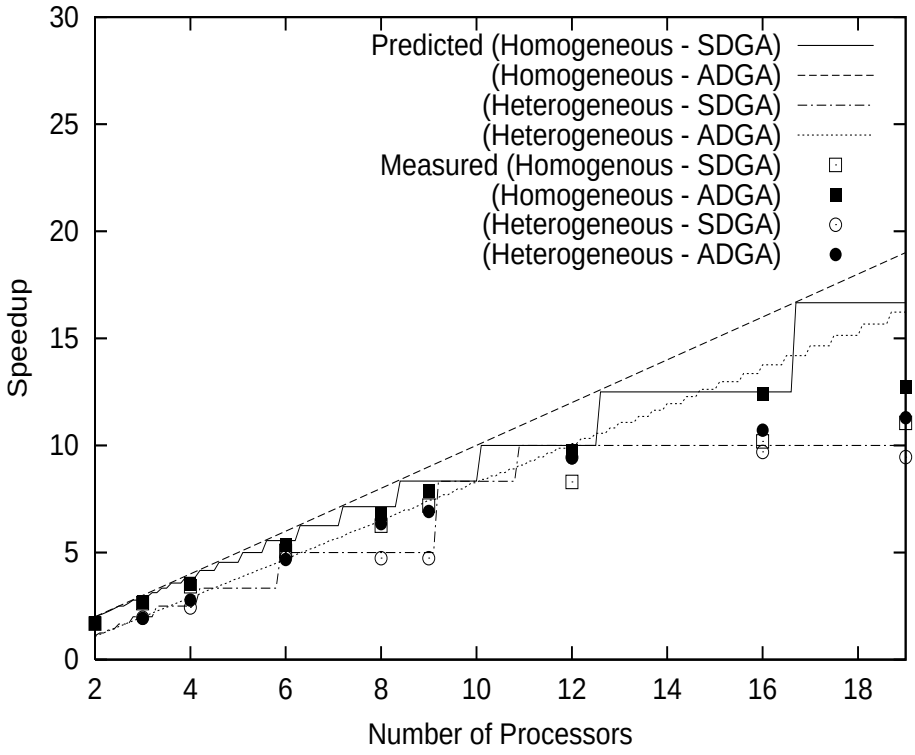


Figure 3.15: Comparison of the speedups of the homogeneous and the heterogeneous systems for the air quality optimization problem

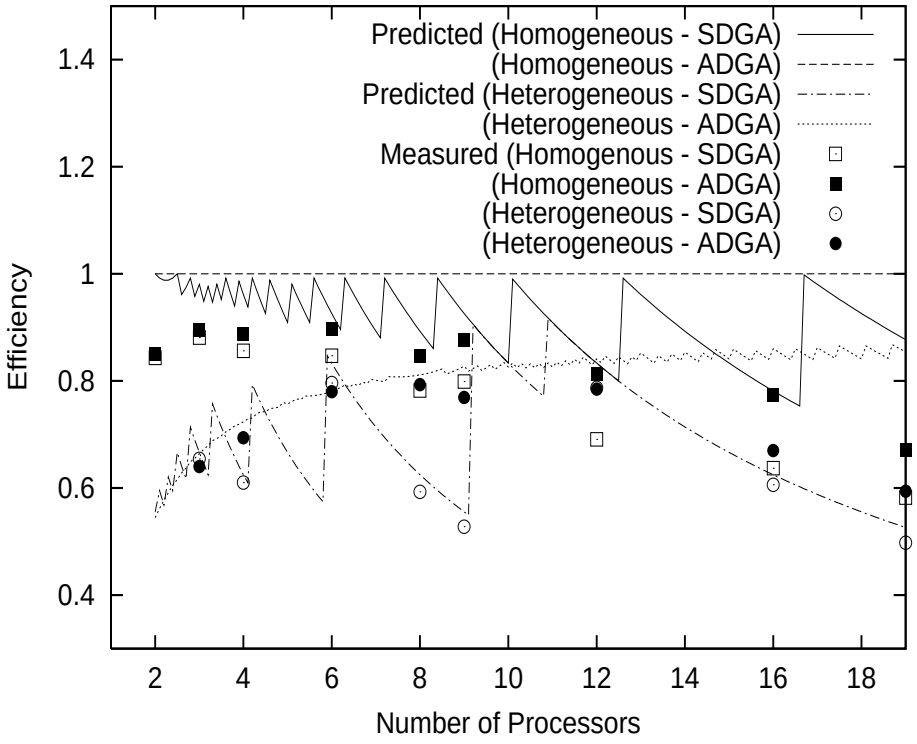


Figure 3.16: Comparison of the efficiencies of the homogeneous and the heterogeneous systems for the air quality optimization problem

References

[1] D. Abramson, R. Sosic, J. Giddy, and B Hall. Nimrod: A tool for performing parametised simulations using distributed workstations. *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing, Washington D.C.*, pages 112–121, 1995.

[2] H. E. Aguirre, K. Tanaka, and S. Oshita. Increasing the robustness of distributed genetic algorithms by parallel cooperative-competitive genetic operators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 195–202, San Francisco, CA., July 2001. Morgan Kaufmann.

[3] L. A. Anbarasu, V. Sundararajan, and P. Narayanasamy. Parallel genetic algorithm for performance-driven sequence alignment. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 1015, San Francisco, CA, July 2001. Morgan Kaufmann.

[4] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, 1995.

[5] Arvind and D. E. Culler. Dataflow architectures. *Annual Reviews in Computer Science*, 1:225–253, 1986.

[6] S. E. Atkinson and D. H. Lewis. A cost-effective analysis of alternative air quality control strategies. *Journal of Environmental Economics*, pages 237–250, 1974.

[7] A. Bevilacqua, R. Campanini, and N. Lanconelli. A distributed genetic algorithm for parameters optimization to detect microcalcifications in digital mammograms. In

E. J. W. Boers et al., editor, *EvoWorkshop*, pages 278–287, Heidelberg, 2001. Springer-Verlag. LNCS 2037.

[8] P. Calegari, F. Guidec, P. Kuonen, and D. Kobler. Parallel island-based genetic algorithm for radio network design. *Journal of Parallel and Distributed Computing*, 47:86–90, 1997.

[9] E. Cantu-Paz. Designing efficient master-slave parallel genetic algorithms. Technical report, University of Illinois at Urbana-Champaign, Urbana, IL, 1997.

[10] E. Cantu-Paz and D. E. Goldberg. On the scalability of parallel genetic algorithms. *Evolutionary Computation*, 7(4):429–449, 1999.

[11] H. S. Chadha and Jr. Baugh, J. W. Network-distributed finite element analysis. *Advances in Engineering Software*, 25:267–280, 1996.

[12] Computer. Special issue on data flow systems. 15(2), 1982.

[13] F. Easton, F. and N. Mansour. A distributed genetic algorithm for deterministic and stochastic labour scheduling problems. *European Journal of Operational Research*, 118:505–523, 1999.

[14] T. C. Fogarty and R. Huang. Implementing the genetic algorithm on transputer based parallel processing systems. *Parallel Problem Solving from Nature*, pages 145–149, 1991.

[15] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

[16] V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In *The 5th International Conference on Genetic Algorithms*, pages 177–183, Urbana-Champaign, IL, 1993. Morgan Kaufmann.

[17] V. S. Gordon, D. Whitley, and A. P. Wim Bohm. Dataflow parallelism in genetic algorithms. *Parallel Problem Solving from Nature*, pages 533–542, 1992.

[18] R. Hauser and R. Manner. Implementation of standard genetic algorithm on mimd machines. *Parallel Problem Solving from Nature*, pages 504–513, 1994.

[19] H. P. Hofstee, J. J. Likkien, and J. L. A. Van De Snepscheut. A distributed implementation of a task pool. *Research Directions in High-Level Parallel Programming Languages*, pages 338–348, 1991.

[20] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.

[21] J. Kim and P. Zeigler. A framework for multiresolution optimization in a parallel/distributed environment: Simulation of hierarchical GAs. *Journal of Parallel and Distributed Computing*, 32:90–102, 1996.

[22] A. Kumar, A. Srivastava, A. Singru, and R. K Ghosh. Robust and distributed genetic algorithm for ordering problems. *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, Syracuse, New York*, pages 253–262, 1996.

[23] Yu-Kwong Kwok and Ahmad Ishfaq. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47:58–77, 1997.

[24] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley and Sons, England, 1990.

[25] T. Maruyama, T. Hirose, and A. Konagaya. A fine-grained parallel genetic algorithm for distributed parallel systems. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 184–198, 1993.

[26] O. Pavel and S. Ivan. Multilevel distributed genetic algorithms. *Genetic Algorithms in Engineering Systems: Innovations and Applications*, September 1995.

[27] T. Reiko. Distributed genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms*, pages 434–439, 1991.

[28] M. G. Schleuter. ASPARAGAS an asynchronous parallel genetic optimization strategy. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427, 1989.

[29] L. Sekanina and V. Dvorak. A totally distributed genetic algorithm: From a cellular system to the mesh of processors. In *Proceedings of the 15th European Simulation Multiconference*, pages 539–543, Delft, The Netherlands, 2001. The SCS Publishing House.

[30] S. K. Sharma and Jr. Baugh, J. W. LAN ho! Structural analysis on a network. In B. J. Goodno and J. R. Wright, editors, *Proceedings of the Eighth Conference on Computing in Civil Engineering and Geographic Information Systems Symposium*, pages 639–646, 1992.

[31] P. A Skordos. Parallel simulation of subsonic fluid dynamics on a cluster of workstations. *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing, Washington D.C*, pages 6–16, 1995.

[32] J. E. Smith and T. C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computing*, volume 72, pages 318–323, 1999.

[33] A. Srivastava, A. Kumar and R. Pathak. Distributed approach to implementing genetic algorithms. *International Conference on Parallel Processing*, pages 106–109, 1994.

Chapter 4

Optimal Design of Redundant Water Distribution Networks using a Cluster of Workstations

(Chapter 4 is a reprint of the manuscript submitted to ASCE Journal of Water Resources Planning and Management.)

by Sujay V. Kumar, Troy. A. Doby, John. W. Baugh Jr., E. Downey Brill, and S. Ranji Ranjithan

Abstract

A genetic algorithm (GA)-based method for the least cost design of looped pipe networks for various levels of redundancy is presented in this paper. Redundancy constraints are introduced in the optimization model by considering the number of pipes assumed to be out of service at any one time. Using this approach, tradeoff relationships between cost and redundancy are developed. The GA-based approach is computationally intensive and implementations on a custom fault-tolerant distributed computing framework, called *Vitri*, are used to satisfy the computational requirements. The design methodology is applied to two water distribution networks of different sizes. A comparison of the performance of the distributed GAs for the design problems is also presented. We conclude that a GA-based approach to obtaining cost-effective, redundant solutions for the least cost design of looped pipe networks can be effectively used on a heterogeneous network of non-dedicated workstations.

Keywords: Genetic algorithms, optimization, water distribution networks, distributed computing, redundancy

4.1 Introduction

Water distribution systems constitute a vital part of civil infrastructure. The purpose of a water distribution system is to ensure the supply of water to users at specified demands. The design of urban water distribution systems typically involves the identification of least cost systems that meet the hydraulic requirements of flow and pressure.

Over the years, the design of least cost models for water distribution systems have progressed from designing branched to looped systems [1, 11]. Loops are included in a water distribution system to provide redundancy in flow paths. The loops may help in meeting the service demands in case of outages due to pipe failures or maintenance and also for critical needs such as fire fighting.

Numerous formal and informal approaches [14] have been developed for designing cost effective looped systems. The formal methods however, tend to produce cost effective looped network designs with minimal pipe sizes, which do not necessarily ensure meeting the flow and pressure requirements when there is an outage. An example of a formal approach is the enforcement of these hydraulic constraints for different pipe configurations that represent different combinations of pipe outages. A formal notion of redundancy considerations, using a GA-based water distribution design approach, has been demonstrated in a previous article [13].

The design of water distribution systems with loops is complex due to the consideration of a wide range of criteria and parameters. Requirements in a typical design model call for the system's ability to deliver a given flow at a specified pressure at the nodes of the system. The loop equations defining flow within the pipes and the cost equations of the pipes are typically nonlinear. Further, the least cost design of a water distribution system using discrete pipe sizes has been shown to be an NP-hard problem [28]. These, coupled with the consideration of additional design requirements, such as redundancy, introduce additional constraints in the design, making it even more complex.

Genetic algorithms (GAs) provide generic search capabilities that are suitable for dealing with such ill-behaved problems that are hard to solve by conventional methods. GAs typically do not rely upon local or gradient information, and are able to deal with complexi-

ties in the problem domains such as local optima and discontinuities. A GA can also handle discrete decision variables and non-linearity in the simulation models effectively, making it an attractive technique for problems such as the optimal design of water distribution systems.

GAs, however, typically require the evaluation of thousands of simulation runs to obtain a solution, making them computationally intensive. For example, in the least cost design of water distribution systems context, a number of different network designs need to be analyzed and checked for feasibility in terms of flow and pressure requirements. In addition, the formal approach for considering redundancy, as presented in this paper, requires the analysis of a number of pipe outage scenarios for each network design. The number of analyses required increases dramatically when higher levels of redundancy are considered, imposing significant computational requirements.

Distributed computing has been used widely in the scientific and engineering community as a tool to harness idle computational resources for dealing with problems that require considerable computing effort [25, 5]. To meet the computational requirements for the proposed design methodology, we use GA implementations in a distributed computing framework.

In this paper we describe the use of distributed GAs and a heterogeneous network of workstations to obtain cost-effective, redundant network designs. The distributed GAs are implemented using an existing high performance computing framework called *Vitri*, which provides basic support for distributed computing and communication for inherently parallel applications. The procedure is demonstrated using two problems from the literature: the "Hanoi problem" with 34 links and "Sioux Falls problem" with 253 links. Computational requirements are discussed, illustrating the potential of the approach as a tool in water distribution planning and design.

4.2 Related Work on water distribution system design

There have been numerous studies on the least cost design of water distribution networks. Lansey and Mays (1989) summarized a number of optimization approaches reported up to

1988, and Simpson et al. (1994) presented a review of subsequent works. The traditional methods of water distribution system design consist of trial-and-error search often aided by commercially available simulation packages. Gessler (1985) proposed a design approach using selective enumeration of possible combinations of pipe sizes for each pipe. Alperovits and Shamir (1977) and Quindry et al. (1979a) presented two linear programming approaches for the least cost design of pipe networks. A number of nonlinear programming approaches have also been reported, including approaches using the reduced gradient technique to identify a local optimum (Murtagh and Saunders 1987, Liebman et al. 1986). These methods, however, do not handle discrete pipe sizes and they have limitations on the size of the networks that can be analyzed.

Many efforts to incorporate reliability in the water distribution system design have also been reported. Su et al. (1987) defined reliability of the network as the probability of satisfying nodal demands and pressure heads for various pipe failures in the water distribution system. They applied a nonlinear programming formulation to optimize looped networks with reliability constraints. A separate model was used to compute the reliability of the distribution system. The method is limited because it can not incorporate network elements such as storage tanks, pumps and valves. Several methods have been developed to design a cost-effective system with alternate paths to each demand node, including heuristic and nonlinear approaches (Morgan and Goulter 1982, Morgan and Goulter 1985, Ostfield and Shamir 1996).

In recent years, GAs have been used as a heuristic search technique to solve the looped network design problem, and have been shown to perform well [24]. Goldberg and Kuo (1987) illustrated the application of GAs in the approximate solution of a pipeline engineering optimization problem. Savic and Walters (1997) developed a GA-based computer model for the least cost design of water distribution networks. As mentioned above, a GA-based approach for including redundancy considerations in the design formulation was presented in a previous article (Kumar et al. 2000).

4.3 Distributed GA Framework

GA implementations in a distributed computing framework are used in this study to support computationally intensive design simulations. A generic distributed computing framework, *Vitri*, solves a group of independent subproblems in a distributed manner by dividing its tasks among heterogeneous clusters of workstations and PCs. By defining the subproblems for a specific application, *Vitri* can be used for various approaches such as Monte Carlo simulation and simulated annealing, as well as the GA implementations used in this paper for designing redundant water distribution networks.

The design of *Vitri* is based on the *Pool of Tasks* paradigm [10], which divides the overall computation into a collection of tasks that are scheduled dynamically among a number of processors. It is implemented as a client-server program, where the client manages a set of tasks, and the servers retrieve tasks from the pool when idle. This paradigm is useful in achieving an efficient distribution of tasks to servers of unequal processing speeds since the faster servers request more tasks than the slower ones and hence balance loads overall.

In contrast to the traditional approach, where a client initiates connections in the servers, the roles of client and servers are reversed in *Vitri*. The client waits for servers to connect to it, and servers can connect or disconnect at any time. This feature allows the owners of potential servers to exercise control in the degree to which they are used in a given problem instance. The reversal of client and server roles also simplifies fault tolerance. Whenever a server fails, the task being performed by that server is simply returned to the pool where it waits to be retrieved by another active server.

Vitri is implemented in the Java programming language. Java sockets using the Transmission Control Protocol (TCP) over Internet Protocol (IP) are used for passing messages between clients and servers since that approach provides reliable connection-oriented communication. Information is passed between the client and the servers with non-blocking *writes* and blocking *reads*.

Vitri is used in this research to support a GA-based stochastic search process, which is based on the principles of Darwinian survival of fittest notion. GAs employ a population of potential solutions, each of which is represented by a set of values symbolizing

the problem's decision variables. A fitness value is associated with each solution, reflecting its ability to satisfy the constraints and objectives of the problem. GAs apply operators such as selection, recombination and mutation to the population of potential solutions. The selection operator simulates the "survival of the fittest" behavior. The individuals with higher fitnesses are preferentially selected to be present in the subsequent populations. As a result, solutions with good traits survive, eliminating the bad traits. The recombination operator creates two new solutions by combining the "genes" of the parent solutions. The mutation operator is used to infuse the population with gene values that may not be present in the population. GAs require no knowledge or gradient information about the decision space, and the discontinuities in the decision space have little effect on the overall search performance. GAs perform fairly well for large-scale problems and can be adapted easily to a wide variety of optimization problems.

The inherent parallelism in a GA is attributed to fact that fitness evaluations can be carried out independently. Distributed GAs can be implemented in several ways. One of the ways to parallelize GAs is through global parallelization, where the evaluations of individuals are done in parallel (Cantu-Paz 1997, Kumar et al. 1996). This method can achieve significant speedups if the communication costs are small compared to the computation costs.

In *Vitri*, distributed GAs are implemented as global parallel GAs. The degree to which GAs can exploit the resources of a networked environment depends on a number of design decisions that affect the parallelism available. In a synchronous distributed GA (SDGA) implementation, for instance, the program waits for computations in each generation to be completed before proceeding to the next generation. In a heterogeneous network, slower processors can impede the progress of the program at the synchronization points by leaving faster processors idle while the slower ones are finishing their computations. The speedup gained by parallel processing can be significantly constrained by these synchronizations between generations. An asynchronous approach that removes these bottlenecks has been developed by applying the dataflow model of computation [3] to GAs. The resulting asynchronous distributed GA (ADGA) [2] has been used effectively to solve the design problems discussed in this paper.

The ADGA is implemented by unrolling the loops in a typical global parallel GA and representing the synchronous instruction scheduling as a dataflow graph. Dataflow models use graphs to represent the flow of data and control, with the execution based solely on the availability and forwarding of data. Thus, the inter-generational data dependencies are captured so that the evaluation of individuals in subsequent generations is constrained only by the availability of data.

4.4 GA Formulation for Reliable System Design

The population of potential solutions undergoes a series of probabilistic operations in a GA. Figure 4.1 depicts a flow chart showing the order of execution of GA operations during optimization. In the selection operation, pairs of individuals are chosen from the GA population such that those individuals that are more fit are more likely to be selected. Each pair of selected individuals may then undergo recombination. From the two parent individuals, two new individuals are created that are random recombinations of the genes of their parents, and are placed in a new population. The mutation operator is used to infuse the population with gene values that may not have existed in the initial population or that may have been lost through selection and recombination. Mutation can be accomplished by randomly selecting a small number of genes in the population, and replacing their values with new, randomly-generated values. The recombination and mutation rates define the probability of crossover between any two pairs and the probability of a gene undergoing mutation, respectively. To ensure that the best solution in any generation is not lost through the probabilistic GA search operators, elitism is used after mutation. The elitist strategy ensures that the best solution from the previous generation is compared with the worst solution in the current generation, replacing the current generation's solution if better.

Each individual in the GA is a potential solution to the pipe network optimization problem. The solution consists of a vector of pipe sizes, corresponding to each link in the network. The representation of a potential solution is shown in Figure 4.2.

The least cost design of a water distribution network can be viewed as the process of determining the combination of pipe sizes that minimizes the overall cost subject to

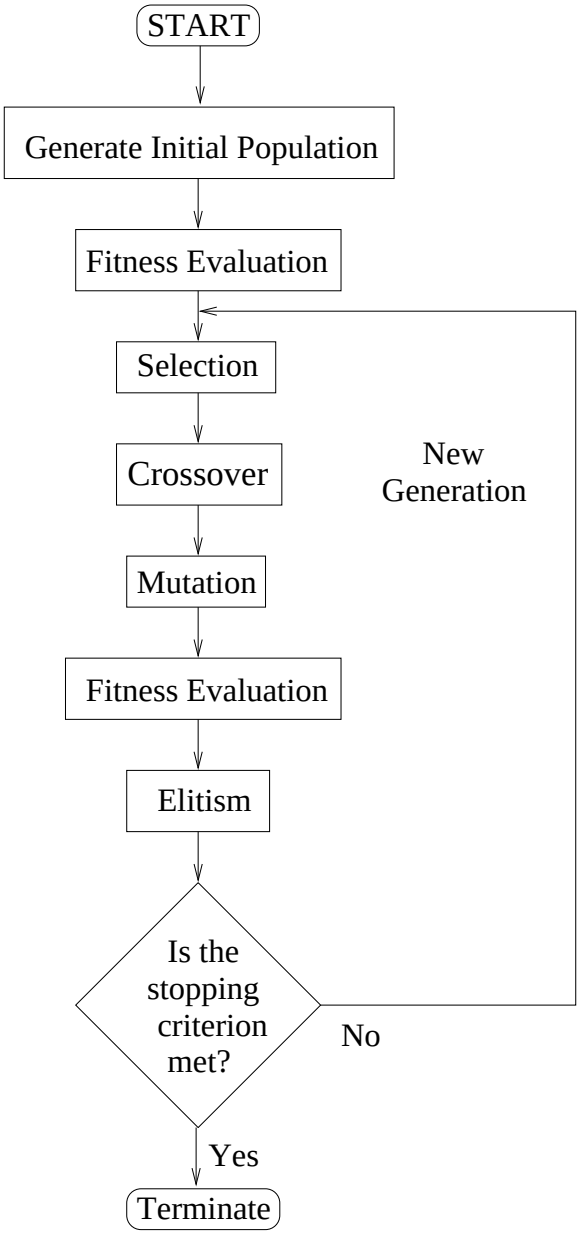


Figure 4.1: Steps in the GA search process

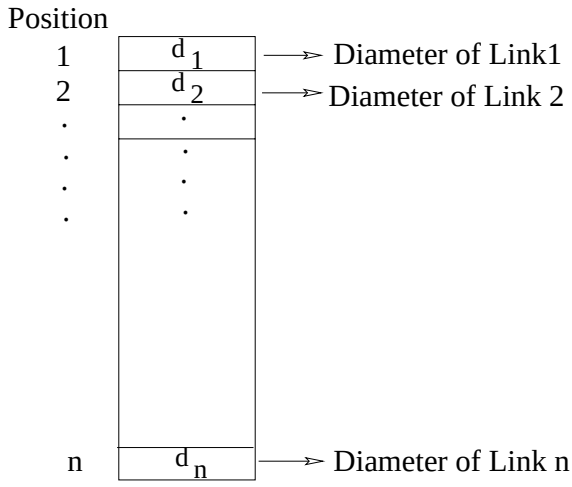


Figure 4.2: GA representation of a candidate solution

hydraulic constraints. The topology, connectivity, and the imposed flow and pressure requirements at different nodes are assumed to be known. The cost of a particular candidate solution is determined by summing the costs of all the pipes in the network. The pipe sizes of the component pipes vary within a specified discrete set.

The function to calculate the fitness of each individual gives a measure of how well a particular solution minimizes cost and meets all the constraints. The simulation of network flows is conducted using EPANET [22] for each potential solution and a penalty for any violation of the constraints is included in the fitness function. The form of the fitness function used is:

$$f = \frac{1}{TC + TP}$$

where f is the fitness value, and TC and TP are the cost and penalty terms, respectively. TC is defined as:

$$TC = \sum_{j=1}^{M} c(d_j) \times L_j$$

where M is the total number of links, $c(d_j)$ is the cost per unit length for link j with a diameter d_j, and L_j is the length of link j. TP is defined as:

$$TP = p \times \langle max_{i=1}^{N}[max(H_i^{min} - H_i, 0)]\rangle$$

where p is a penalty on the maximum violation of the head constraints among N nodes. H_i is the head at node i, H_i^{min} is the minimum head required at node i, and N denotes the total number of nodes.

The following constraints considered in the formulation are enforced by EPANET.

- For each junction node in the network, continuity of flow should be maintained.
  $\sum Q_i^{in} - \sum Q_i^{out} = D_i$
  where Q_i^{in} and Q_i^{out} are the flow in and out of node i respectively, and D_i is the external demand at node i. Steady state flow conditions in the network are assumed.
- The total head loss around a loop must be zero.
  $\sum h_f - \sum E_p = 0$
  where E_p is the energy put in by a pump and h_f is the head loss along a pipe.
- The head loss along a pipe is dependent on the hydraulic properties of the pipe, its length, diameter, and the flow in the pipe. The Hazen-Williams or Darcy-Weisbach equations [27] can be used to predict the head loss. Pumps can be used to provide positive head increases.

The minimum cost design for a given layout will drive the system towards a branched layout. However, because pipe sizes are selected from a discrete set (which does not contain a zero diameter), the existence of loops is ensured. Although in this case, the water distribution systems are designed with loops, many pipes will be of the minimum size allowed. As a result, the hydraulic requirements may be inadequate.

We define different levels of redundancy as follows: The minimum cost solution is termed a Level-0 redundant solution since it does not consider performance constraints in the event of a pipe failure. The GA formulation for the Level-0 case can be easily extended to consider redundancy requirements. A Level-1 redundant solution meets the flow and pressure requirements in the event of any single pipe outage in the network. Similarly, a Level-2 solution meets the requirements in the event of any two pipe outages. To obtain a minimum cost Level-1 solution using a GA, each of the cases with a single pipe out of

service must be analyzed to ensure that the hydraulic constraints are met. This is done by enumerating each feasible single pipe outage in the network. If the removal of a link causes nodes to become isolated, the possible removal of that link is not considered. As in the Level-0 case, the penalties for violations of head constraints for each scenario are incorporated in the fitness function.

To obtain a cost-effective Level-2 solution, every two possible pipe outages in the network are enumerated and analyzed, and appropriate penalties are included in the fitness functions. As in the Level-1 case, an outage of a given pair of pipes in the Level-2 case is not considered if it would isolate nodes. This working definition of redundancy makes it possible to incorporate a common understanding of the need for looped networks within a formal search procedure intended to obtain a cost effective design. A cost effective design is a minimum cost solution that will meet flow and pressure requirements with a link(s) out of service.

4.5 Example Applications and Results

The GA-based design process described above is illustrated using two water distribution networks presented in the literature. The first network is a planning problem for a water distribution trunk network for the city of Hanoi, Vietnam [7]. The second system is a skeletonized network for a planning problem based on information for Sioux Falls, South Dakota (Quindry et al. 1979b). The first network is a relatively small system. Small additions similar to this system are frequently appended to existing water distribution systems. The second network demonstrates our approach on a larger system.

The various GA analyses of both networks use an elitist strategy with a mutation rate of 0.005 and a recombination rate of 0.9. All GA runs have a population size of 100. Several test runs are used to determine an appropriate penalty for the fitness function. The convergence in a GA run is assumed when the number of generations exceed a fixed value.

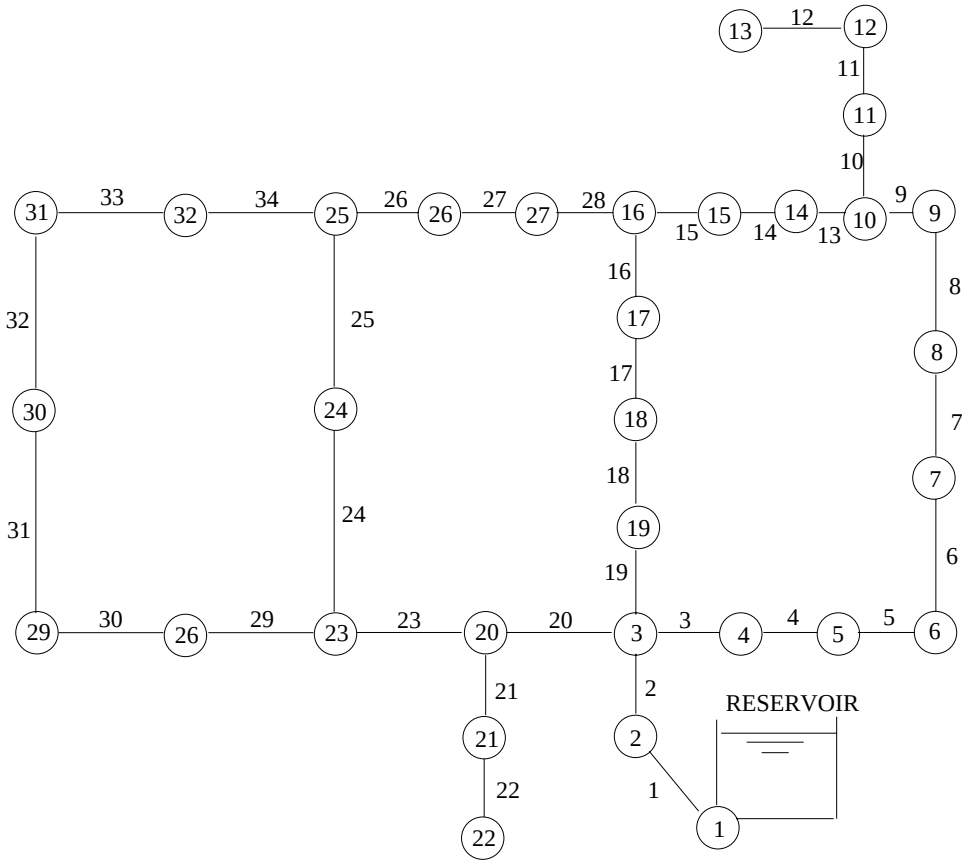


Figure 4.3: Hanoi water distribution network

4.5.1 Hanoi Example

Fujiwara and Khang (1990) considered the network planning problem shown in Figure 4.3. The network consists of 32 nodes, 34 pipes, and 3 loops. No pumping facilities are considered. The commercially available pipe sizes for the problem are 12, 16, 20, 24, 30, and 40 inches, and the cost of each pipe per unit length is defined as $1.1 \times D_i^{1.5}$. A Hazen-Williams coefficient of 162.5 is assumed. The acceptable pressure requirement for all the nodes is considered to be 30 m above ground level in all cases.

The Level-1 analysis considers 29 feasible single pipe outage scenarios, and there are 276 feasible Level-2 scenarios. Figure 4.4 shows a typical plot for the average and best fitnesses in each generation as the GA progresses. Figure 4.5 shows the average and best cost values in each generation. The best cost values obtained for various levels of

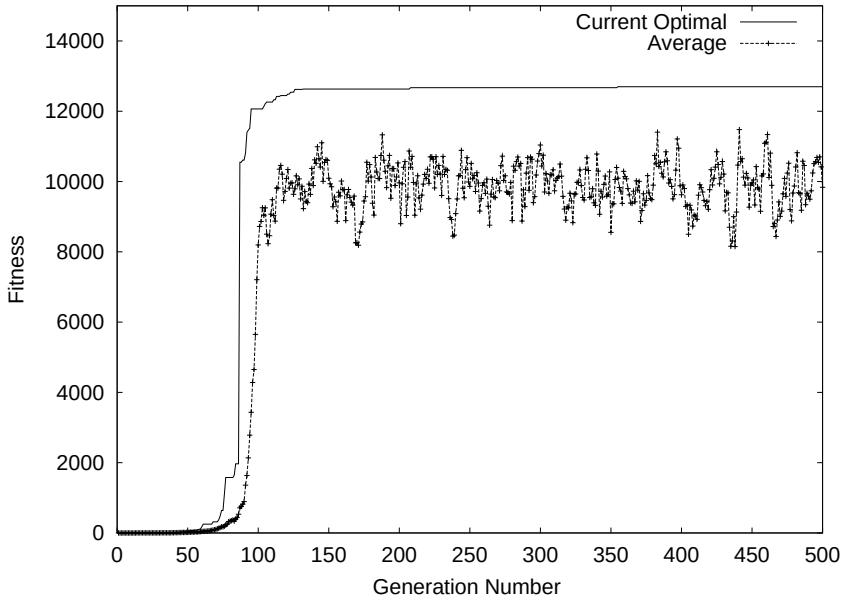


Figure 4.4: Fitness for the Hanoi network, Level-1

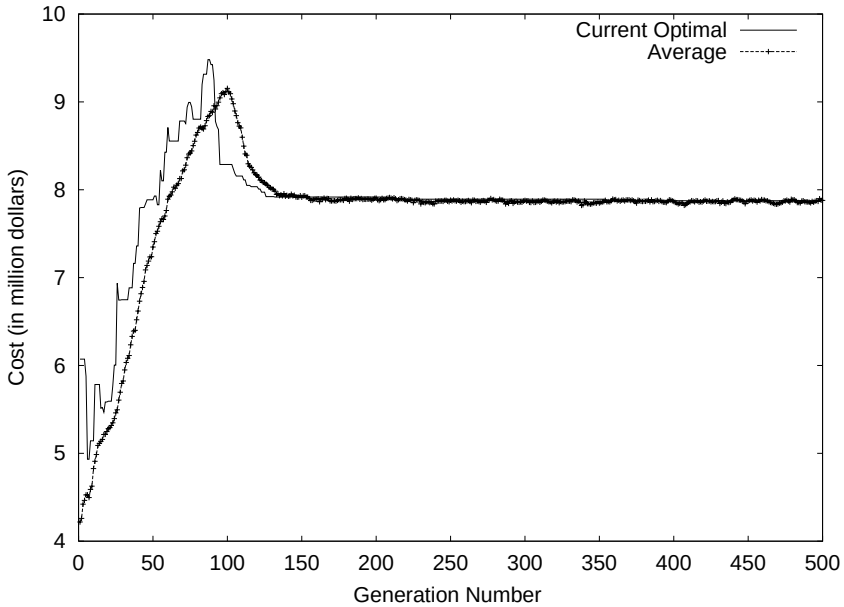


Figure 4.5: Cost for the Hanoi network, Level-1

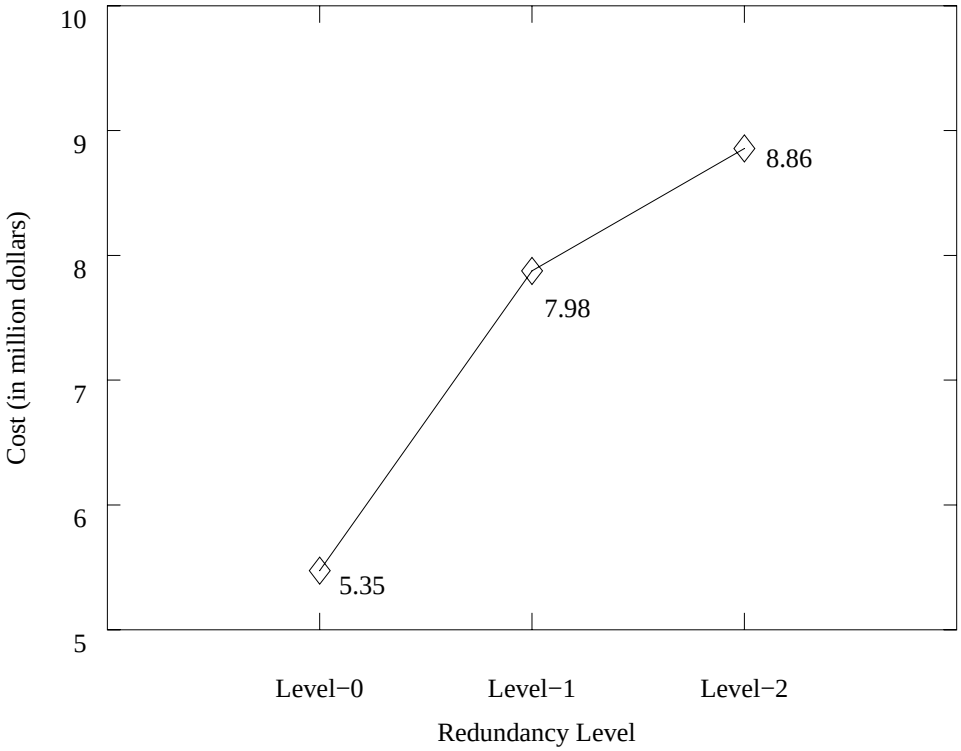


Figure 4.6: Tradeoff between cost and redundancy level for the Hanoi network

redundancy are shown in Figure 4.6. The optimum pipe diameters for these solutions are shown in Table 4.1. It can be observed that for each additional level of redundancy, there is a clear trend towards increased pipe diameters and cost. The optimum diameters of some of the links, however, actually decreased with an increase in redundancy level. For example, the diameter of pipe 31 decreased from 20 to 16 in going from Level-0 to Level-1, while the diameter of pipe 34 increased. The pipe sizes of links 16, 17, 18, and 19 increased in going from Level-0 to Level-1, but decreased from Level-1 to Level-2. In the latter case, the increase in the diameter of pipe 15 allows the demands at nodes 17, 18, and 19 to be met with smaller diameters in the other pipes. The application of these changes and analysis of corresponding systems with EPANET confirmed the changes to be cost-effective, and not simply artifacts of the heuristic GA search process.

The results of the Hanoi network problem are averaged over three runs using the ADGA. For comparison, results obtained using the SDGA are also presented. A network consisting of three Pentium III machines running Redhat 6.2 Linux on a 10BaseT Ethernet

Table 4.1: Pipe Diameters (in.) for Solutions to Hanoi problem

Link	Level-0	Level-1	Level-2
1	40	40	40
2	40	40	40
3	30	40	40
4	30	30	40
5	30	40	40
6	30	30	40
7	24	30	40
8	24	30	40
9	24	30	40
10	24	24	24
11	24	24	24
12	24	24	20
13	20	40	40
14	16	30	40
15	24	30	40
16	30	40	24
17	30	40	24
18	30	40	24
19	30	40	24
20	30	40	30
21	16	20	40
22	12	24	40
23	30	40	40
24	20	40	40
25	12	40	40
26	12	40	40
27	20	40	30
28	16	40	30
29	24	24	40
30	20	20	40
31	20	16	40
32	16	16	40
33	16	16	40
34	12	20	40

Table 4.2: Computational times for the Hanoi network analysis

Type of Analysis	Time taken by sequential GA (estimated)	Time taken by SDGA (measured)	Time taken by ADGA (measured)	Number of analysis
Level-0 (minutes)	66.68	27.3	25.07	500,000
Level-1 (hours)	32.22	14.98	12.55	14,500,000
Level-2 (days)	14.01	5.91	5.28	138,000,000

LAN was used for the analysis. The estimated time for a sequential GA and the measured times of the SDGA and ADGA runs are presented in Table 4.2. All the runs were conducted for 500 generations. Since the total number of analysis runs is known for a fixed number of generations, the required time for a sequential GA run can easily be estimated.

It can be observed that the computational requirements increase dramatically with higher levels of redundancy as expected given the large increase in the number of analyses required (Table 4.2). Figure 4.7 demonstrates the exponential increase in execution times and how the use of distributed GAs helps in reducing the total execution times considerably. The linear execution time shown in Figure 4.7 is the predicted time assuming the use of three machines. The results show that the execution times decrease nearly in proportion to the number of processors, implying that the speedups are nearly linear. The ADGA further reduces the execution times of the SDGA by approximately 10%.

4.5.2 Sioux Falls Problem

The network shown in Figure 4.8, which is taken from Quindry et al. (1979b), represents a planning model of a water distribution system for Sioux Falls, South Dakota, for the year 2010. The network consists of 253 pipes and 186 nodes.

Table 4.3 presents the list of commercially available diameters and the corresponding costs per meter of pipe length that are used in the analysis. The acceptable pressure requirements for all the demand nodes are again considered to be 30 m above ground level in all cases. The Level-1 analysis considered 227 possible single pipe outage scenarios and Level-2 considered 26,476 possible pairs of outages. As a result, the analysis of this problem requires considerable computational resources. The Sioux Falls problem was solved using a heterogeneous network of workstations running Linux, Solaris, and Windows operating

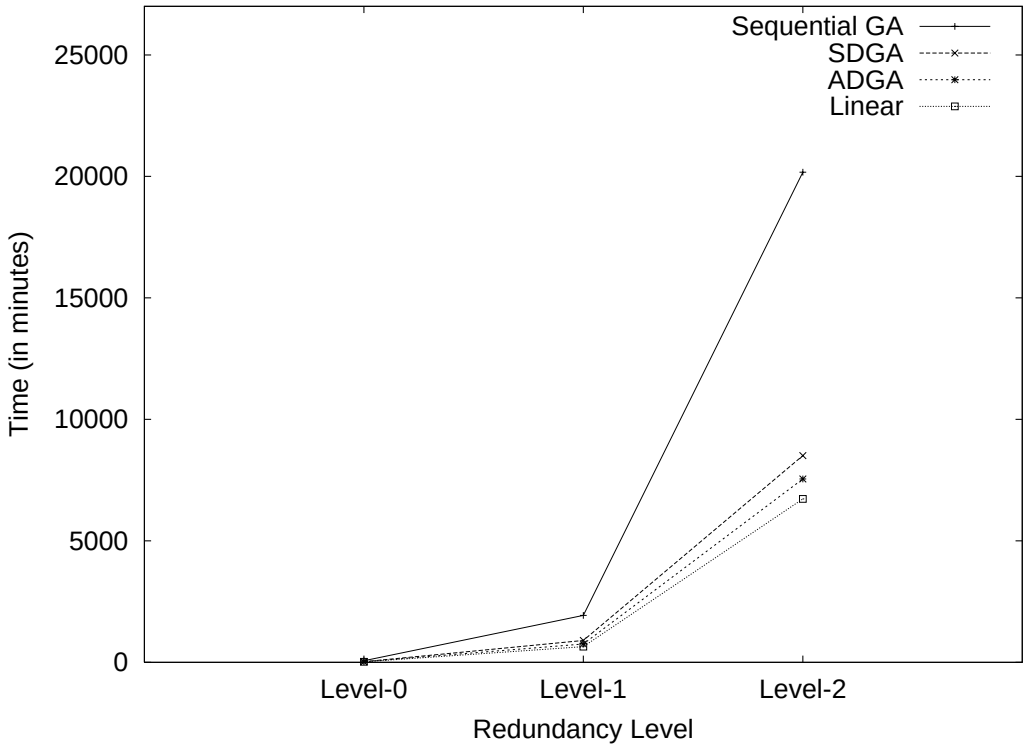


Figure 4.7: Computational times for the Hanoi network

systems. The number of machines involved in the analysis varied from 5 to 25. The experience with the Hanoi problem led to the use of ADGA for this problem. The estimated time for a sequential GA and actual computational times required for various runs are shown in Table 4.4. The Level-0 and Level-1 analysis was conducted for 500 generations, and the Level-2 analysis for 200 generations. It can be seen that the use of a distributed GA leads to considerable improvements in the execution times.

The optimum costs obtained for various levels of analysis are shown in Table 4.5, and the optimum pipe diameters are shown in Table 4.6. Once again, with an increase in redundancy requirements, the pipe diameters tend to increase, but many exceptions can be observed. The tradeoff between cost and various levels of redundancy is shown in Figure 4.9.

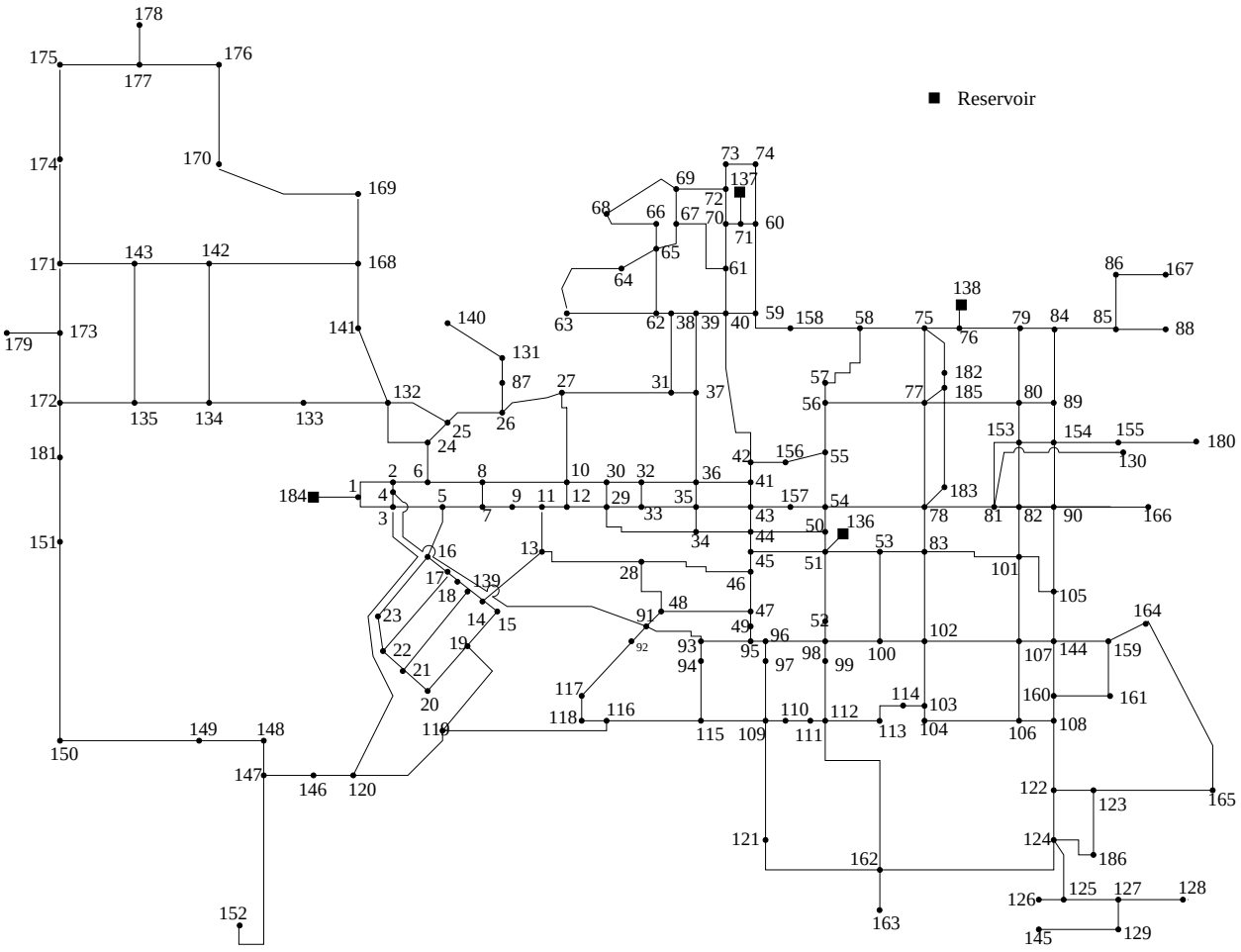


Figure 4.8: Sioux Falls water distribution network

Table 4.3: Cost data for pipes for the Sioux Falls system

Diameter (in.)	Cost (dollars)
1	2
2	5
3	8
4	11
6	16
8	23
10	32
12	50
14	60
16	90
18	130
20	170
22	300
24	550

Table 4.4: Computational time requirements for the Sioux Falls network design

Type of Design	Number of Processors	Time taken by sequential GA (estimated)	Time taken by ADGA (measured)
Level-0 (hours)	11	2.08	0.21
Level-1 (days)	5–10	19.70	2.86
Level-2 (months) (for 200 generations)	5-25	16.35	1.82

Table 4.5: Costs of solutions for the Sioux Falls problem

Type of Design	Cost (in Million Dollars)
Level-0 Redundant	1.113
Level-1 Redundant	1.974
Level-2 Redundant	3.035

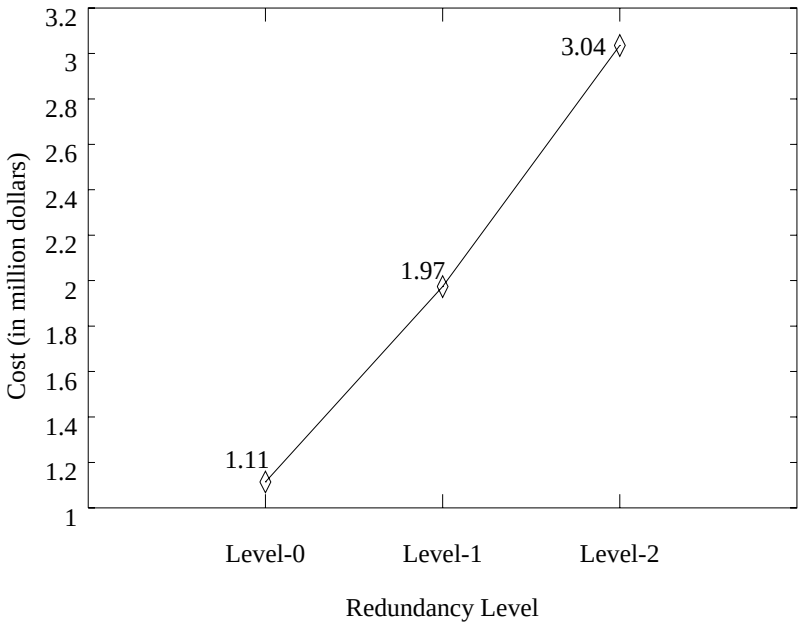


Figure 4.9: Tradeoff between cost and redundancy level for the Sioux Falls problem

4.6 Conclusions

A GA-based approach can be used to enforce commonly understood requirements for redundancy and to obtain cost effective designs. The approach is generic and can be applied to various types of networks. The two networks considered in this paper did not include hydraulic elements such as valves, pumps, and tanks. This, however, does not pose a limitation on the approach since EPANET, which is used for the hydraulic analysis, can model water distribution systems that contain these elements.

The results presented in this paper show that the design of water distribution networks under considerations of redundancy can be computationally intensive and can be effectively handled using distributed GA approaches. *Vitri* provides a multi-tasking, fault-tolerant, and scalable distributed platform which can be used effectively to harness idle CPU cycles of workstations and PCs. The *Vitri* platform is also flexible enough to incorporate a heterogeneous system of computers with various processing speeds, running different operating systems. It can be observed that the use of ADGA on the *Vitri* distributed platform has been used to meet some of the computational requirements. The use of a larger number of servers is expected to improve the execution times.

The technique developed in this paper provides a framework for incorporating other measures of redundancy or reliability in the analysis. The definition of an outage could be modified based on the locations of valves in an actual network. Probabilities of failures of component pipes could also be included if the information is available. A Monte Carlo simulation could be used to determine the level of reliability for a given candidate solution. Using the GA approach, the fitness function could be modified to include a requirement for a given level of reliability. By varying this level, a tradeoff between cost and reliability could be obtained. The introduction of such criteria will increase the computational requirements and the success of such design formulations is dependent upon the availability of adequate computational resources.

The *Vitri* framework could also be used to examine more complex variations of the water distribution design problem. For instance, including measures of water quality constraints is currently being investigated [6].

Table 4.6: Pipe Diameters (in.) for Solutions to the Sioux Falls problem

Link	Level-0	Level-1	Level-2	Link	Level-0	Level-1	Level-2
1	6	24	24	31	14	4	4
2	8	2	2	32	16	12	12
3	18	20	20	33	24	4	4
4	18	14	14	34	14	18	18
5	12	24	24	35	12	14	14
6	3	4	4	36	3	6	6
7	22	18	18	37	8	16	16
8	2	10	10	38	1	14	14
9	24	14	14	39	3	22	22
10	14	12	12	40	1	6	6
11	16	6	6	41	3	8	8
12	18	6	6	42	22	14	14
13	18	12	12	43	18	8	8
14	8	14	14	44	3	8	8
15	8	18	18	45	8	10	10
16	20	12	12	46	6	10	10
17	22	14	14	47	1	14	14
18	20	3	3	48	4	4	4
19	10	18	18	49	4	12	12
20	2	16	16	50	20	14	14
21	1	10	10	51	12	16	16
22	3	4	4	52	10	12	12

Table 4.6: Pipe Diameters (in.) for Solutions to the Sioux Falls problem

Link	Level-0	Level-1	Level-2	Link	Level-0	Level-1	Level-2
23	3	8	8	53	1	10	10
24	22	14	14	54	8	8	8
25	24	12	12	55	20	12	12
26	16	10	10	56	18	6	6
27	8	4	4	57	22	10	10
28	4	6	6	58	3	22	22
29	8	6	6	59	6	10	10
30	8	3	3	60	18	8	8
61	20	14	14	100	14	8	8
62	1	3	3	101	14	16	16
63	18	1	1	102	2	2	2
64	6	3	3	103	12	8	8
65	2	4	4	104	22	22	22
66	20	8	8	105	8	18	18
67	22	6	6	106	8	14	14
68	18	1	1	107	2	10	10
69	22	3	3	108	3	14	14
70	12	2	2	109	1	4	4
71	16	14	14	110	16	10	10
72	6	4	4	111	24	3	3
73	18	10	10	112	24	8	8
74	4	12	12	113	3	6	6
75	16	8	8	114	3	14	14
76	14	16	16	115	6	8	8
77	20	8	8	116	8	4	4
78	18	16	16	117	16	16	16
79	18	22	22	118	10	10	10
80	4	6	6	119	4	6	6
81	2	4	4	120	20	1	1
82	22	10	10	121	22	12	12
83	20	14	14	122	3	10	10
84	10	4	4	123	2	10	10
85	14	3	3	124	6	6	6
86	10	10	10	125	10	1	1
87	14	1	1	126	18	6	6
88	16	6	6	127	22	4	4
89	4	4	4	128	3	4	4
90	10	18	18	129	10	12	12
91	8	16	16	130	6	6	6

Table 4.6: Pipe Diameters (in.) for Solutions to the Sioux Falls problem

Link	Level-0	Level-1	Level-2	Link	Level-0	Level-1	Level-2
92	6	14	14	131	1	3	3
93	22	2	2	132	1	3	3
94	18	22	22	133	8	14	14
95	18	18	18	134	18	3	3
96	16	16	16	135	22	12	12
97	12	10	10	136	10	18	18
98	6	2	2	137	3	14	14
99	16	6	6	138	16	22	22
139	12	12	12	178	6	8	8
140	10	4	4	179	6	4	4
141	8	14	14	180	14	20	20
142	12	14	14	181	4	16	16
143	20	12	12	182	8	20	20
144	2	20	20	183	6	16	16
145	18	6	6	184	24	4	4
146	16	6	6	185	3	14	14
147	3	8	8	186	18	16	16
148	18	14	14	187	18	18	18
149	8	10	10	188	2	12	12
150	8	10	10	189	2	4	4
151	6	2	2	190	18	10	10
152	1	10	10	191	12	10	10
153	1	12	12	192	16	10	10
154	16	10	10	193	22	8	8
155	3	4	4	194	4	8	8
156	24	3	3	195	12	14	14
157	24	2	2	196	6	14	14
158	16	6	6	197	6	12	12
159	14	8	8	198	20	14	14
160	10	8	8	199	12	16	16
161	8	6	6	200	20	10	10
162	3	20	20	201	24	14	14
163	16	18	18	202	8	16	16
164	4	10	10	203	18	14	14
165	14	6	6	204	16	10	10
166	18	20	20	205	12	4	4
167	8	12	12	206	18	14	14
168	2	8	8	207	10	2	2
169	24	6	6	208	2	8	8

Table 4.6: Pipe Diameters (in.) for Solutions to the Sioux Falls problem

Link	Level-0	Level-1	Level-2	Link	Level-0	Level-1	Level-2
170	6	14	14	209	8	14	14
171	20	6	6	210	1	3	3
172	16	6	6	211	20	10	10
173	14	1	1	212	6	14	14
174	18	6	6	213	22	12	12
175	12	8	8	214	4	18	18
176	12	6	6	215	2	16	16
177	6	4	4	216	20	8	8
217	22	1	1	236	16	16	16
218	2	6	6	237	14	12	12
219	3	6	6	238	22	16	16
220	4	12	12	239	4	12	12
221	8	8	8	240	8	14	14
222	10	3	3	241	18	16	16
223	8	12	12	242	14	20	20
224	6	6	6	243	6	12	12
225	3	6	6	244	1	12	12
226	12	3	3	245	1	10	10
227	10	14	14	246	10	10	10
228	6	14	14	247	3	4	4
229	12	12	12	248	18	14	14
230	20	20	20	249	8	10	10
231	3	18	18	250	10	10	10
232	8	14	14	251	1	6	6
233	24	2	2	252	8	4	4
234	6	14	14	253	8	20	20
235	8	16	16				

References

[1] E. Alperovits and U. Shamir. Design of optimal water distribution systems. *Water Resources Research*, 13(6):885–900, 1977.

[2] J. W. Baugh and S. V. Kumar. Asynchronous genetic algorithms using coarse-grained dataflow. in preparation, 2001.

[3] J. W. Baugh and D. R. Rehak. Applications of coarse-grained dataflow in computational mechanics. *Engineering with Computers*, 8(1):13–30, 1992.

[4] E. Cantu-Paz. Designing efficient master-slave parallel genetic algorithms. Technical Report IlliGAL 97004, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1997.

[5] C. H. Cap and V. Strumpen. Efficient parallel computing in distributed workstation environments. *Parallel Computing*, 19:1221–1234, 1993.

[6] T. A. Doby, S. V. Kumar, J. W. Baugh, E. D. Brill, and S. R Ranjithan. Use of a distributed genetic algorithm for optimizing cost and water quality in a looped pipe network for various redundancy levels. In *Proceedings of the 2001 World Water and Environmental Resource Congress*. ASCE, 2001.

[7] O. Fujiwara and D. B. Khang. A two-phase decomposition method for the optimal design of looped water distribution networks. *Water Resources Research*, 26(4):539–549, 1990.

[8] J. Gessler. Pipe network optimization by enumeration. In *Proceedings, Computer Applications for Water Resources*, pages 572–581, ASCE, New York, N.Y., 1985.

[9] D. E. Goldberg and C. H. Kuo. Genetic algorithms in pipeline optimization. *Journal of Computing in Civil Engineering*, 1(2):128–141, 1987.

[10] H. Hofstee, J. Likkien, and J. Van De Snepscheut. A distributed implementation of a task pool. *Research Directions in High-Level Parallel Programming Languages*, pages 338–348, 1991.

[11] A. J. Kettler and I. C. Goulter. Reliability consideration in the least cost design of looped water distribution systems. In *International Symposium on Urban Hydrology, Hydraulics and Sediment Control*, pages 305–312, University of Kentucky, Lexington, Kentucky, 1983.

[12] A. Kumar, A. Srivastava, A. Singru, and R. K. Ghosh. Robust and distributed genetic algorithm for ordering problem. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 253–262, Syracuse, New York, 1996.

[13] S. V. Kumar, T. A Doby, J. W. Baugh, E. D. Brill, and S. R. Ranjithan. Method for least cost design of looped pipe networks for different levels of redundancy using genetic algorithms. In *In Proceedings of the 2000 Joint Conference on Water Resources Engineering and Water Resources Planning and Management*. ASCE, 2000.

[14] K. E. Lansey and L. W. Mays. Reliability analysis of water distribution systems. In L. W. Mays, editor, *Optimization Model for Design of Water Distribution Systems*, pages 37–84. ASCE, 1989.

[15] J. S. Liebman, L. Lasdon, L. Schrage, and A. Waren. *Modeling and Optimization with GINO*. The Scientific Press, Palo Alto, California, 1986.

[16] D. R. Morgan and I. C. Goulter. Least cost layout and design of looped water distribution systems. In *International Symposium on Urban Hydrology, Hydraulics and Sediment Control*, pages 27–29, University of Kentucky, Lexington, Kentucky, 1982.

[17] D. R. Morgan and I. C. Goulter. Optimal urban water distribution design. *Water Resources Research*, 21(5):642–552, 1985.

[18] B. A. Murtagh and M. A. Sanders. *MINOS 5.1 Users's Guide*. System Optimization Laboratory, Department of Operations Research, Stanford University, California, 1987.

[19] A. Ostfield and U. Shamir. Design of optimal reliable multiquality design criteria. *Journal of Water Resources Planning and Management*, 122(5):322–333, 1996.

[20] G. E. Quindry, E. D. Brill, and J. C. Liebman. Water distribution system design criteria. Technical Report NSF Grant APR-7619120, University of Illinois at Urbana-Champaign, Department of Civil Engineering, Urbana, Illinois, 1979b.

[21] G. E. Quindry, E. D. Brill, J. C. Liebman, and A. Robinson. Comment on the design of optimal water distribution systems by alperovits and shamir. *Water Resources Research*, 15(6):1651–1654, 1979a.

[22] L. A. Rossman. *EPANET, User's Manual*. U.S. Environmental Protection Agency, Cincinnati, Ohio, 1993.

[23] D. A. Savic and G. A. Walters. Genetic algorithms for least-cost design of water distribution networks. *Journal of Water Resources Planning and Management*, 120(4):423–443, 1997.

[24] A.R. Simpson, G.C. Dandy, and L.J. Murphy. Genetic algorithms compared to other techniques for pipe optimization. *Journal of Water Resources Planning and Management*, 120(4):423–443, 1994.

[25] P. A Skordos. Parallel simulation of subsonic fluid dynamics on a cluster of workstations. In *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing*, pages 6–16. Washington D.C., 1995.

[26] Y. C. Su, L. W. Mays, N. Duan, and K. E. Lansey. Reliability-based optimization model for water distribution systems. *Journal of Hydraulic Engineering*, 114(12):1539–1556, 1987.

[27] T. M. Walski. *Analysis of water distribution systems*. Van Nostrand Reinhold Co., Inc., New York, 1984.

[28] D.F. Yates, A.B. Templeman, and T.B. Boffey. The computational complexity of the problem of determining least capital cost designs for water supply networks. *Engineering Optimization*, 7(2):142–155, 1984.

Chapter 5

Comparative Study of the Constraint Method-based Evolutionary Algorithm (CMEA) with Other Evolutionary Algorithms for Multiobjective Optimization

(Chapter 5 is reprint of the manuscript submitted to IEEE Transactions on Evolutionary Computation)

by Sujay Kumar and Ranji Ranjithan

Abstract

In the operations research literature, the ϵ-Constraint Method has been demonstrated to be a useful approach for generating noninferior solutions for multiobjective problems. Further, the use of this method in conjunction with mathematical programming procedures has shown that the noninferior set can be generated more efficiently by using a noninferior solution as a starting point to find the adjacent noninferior solution. The Constraint Method-based Evolutionary Algorithm (CMEA) embodies these proven concepts into an evolutionary computation framework to offer a new multiobjective evolutionary algorithm (MOEA). This paper reports the results from a study comparing the performance of CMEA with those of other commonly reported MOEAs. A suite of 2-objective test problems, representing a range of complexities in the decision space as well as in the objective space, is chosen from the MOEA literature. In addition to comparing graphically the noninferior solutions obtained by the different MOEAs, quantitative performance metrics for accuracy, coverage, and spread are also used in the comparisons. For the problems considered in this paper, CMEA performs as well as or better than the other MOEAs tested.

Keywords: Multiobjective optimization, evolutionary algorithm, ϵ-constraint method, non-

inferior set, Pareto set.

5.1 Introduction

Many design optimization problems in engineering applications inherently involve consideration of multiple, non-commensurate, and often competing criteria that reflect various design specifications and constraints. For such multiobjective optimization problems, the examination of non-inferior tradeoffs among different objectives require the identification of efficient, Pareto-optimal solutions. Evolutionary algorithms (EAs), which are becoming increasingly more applicable in solving real world engineering problems [10], continue to be extended to incorporate multiobjective optimization.

An array of MOEAs have been reported in the literature, and extensive summaries are provided by Deb [7], Coello [3] and Van Veludhizen and Lamount [15]. More recently, Ranjithan et al. [14] presented an approach called the constraint method-based evolutionary algorithm (CMEA). This method is based on the underlying concepts used in the ϵ-constraint method that is described in the mathematical programming and multiobjective optimization literature([5], [1], [13]). CMEA is an evolutionary algorithm-based approach that achieves Pareto optimality via the ϵ-constrained method coupled with beneficial population seeding in the intermediate iterations of the algorithm.

While the previous work by Ranjithan et al. [14] tested CMEA using a few illustrative test problems, this paper presents an extensive testing and comparison of CMEA. Numerous 2-objective test problems representing different complexities (e.g., number and type of variables, and degrees of constraints) in the search space are used in this study. Several performance metrics are used to compare the performance of CMEA with those of commonly reported MOEAs, for example, non-dominated sorting genetic algorithm (NSGA-II) [8], strength pareto evolutionary approach (SPEA) [17], pareto envelope-based selection algorithm (PESA) [6], and micro-GA [4]. The results demonstrate that CMEA performs consistently well for this wide range of multiobjective problems.

5.2 Background

The description of the CMEA approach and its implementation is described in Ranjithan et al. [14]. CMEA is a procedure that integrates the ϵ-constraint method for multiobjective optimization (MO) within an evolutionary computation framework. In the traditional ϵ-constraint method, the MO problem is converted into a number of single objective optimization problems by treating all but one objective as constraints. The noninferior solutions are obtained by solving a series of these single objective optimization problems that are defined by varying levels of constraints corresponding to the other objectives. This approach of solving a number of independent single objective optimization problems to generate the noninferior set becomes less attractive if obtaining each noninferior solution is computationally intensive. Some mathematical programming procedures that implement the ϵ-constraint method improve the search for a new noninferior solution by seeding the starting solutions based on the previously generated adjacent noninferior solution. This approach works well for those classes of problems in which the noninferior solutions adjacent in the decision space map to adjacent points in the objective space.

CMEA couples this adjacency mapping property and the concepts of the ϵ-constraint method within an MOEA framework to generate the noninferior set. CMEA achieves Pareto optimality in an implicit manner by ensuring that the population migrates along the noninferior surface. At each iteration, the algorithm finds a noninferior solution by converging the population to the optimal solution to the following single objective optimization problem.

$$\text{Maximize } Z_h(\mathbf{x}) \tag{5.1}$$

$$\text{Subject to } g_i(\mathbf{x}) \leq 0 \; \forall i = 1, 2, ..., m \tag{5.2}$$

$$Z_l(\mathbf{x}) \geq Z_l^t \; \forall l = 1, 2, ..., k; l \neq h \tag{5.3}$$

$$\mathbf{x} \in X \tag{5.4}$$

Without loss of generality, it can be assumed that all the objectives in this problem are being maximized, where Z_h is one of the k objectives, $Z_l^t (l = 1, 2, ..., k; l \neq h)$ is the constraint value for objective $l (\neq h)$, $\mathbf{x} = \{x_j : j = 1, 2, ..., n\}$ represents the decision vector, X represents the decision space, m is the total number of constraints, and $g_i(\mathbf{x})$ is

the i^{th} constraint. Z_l^t is varied incrementally, thereby making the search migrate from one noninferior solution to an adjacent solution, eventually tracing the noninferior surface.

The steps involved in CMEA are shown in Figure 5.1. Any appropriate stopping criteria for single objective EA search can be used to determine convergence of the inner loop of the algorithm. In the study reported in this paper, convergence in each individual EA run is determined when the number of generations exceeds a maximum value or if the best solution does not improve within a specified number of generations. The outer loop of the algorithm drives the generation of a sequence of noninferior solutions by altering the ϵ-constraint values for all but one objective in the search problem solved in the inner loop. Starting from an extreme point (corresponding to a single objective solution), the constraint values are systematically incremented to trace the noninferior set.

5.3 Testing and Evaluation of CMEA

CMEA is applied to a number of test problems with different degrees of difficulty and characteristics. Deb et al. [9] presented a *tunable* test problem generator, which can be used to obtain constrained test problems with desired level of difficulty. They also presented simulation results obtained using the NSGA-II approach. The constrained test problems described by Deb et al. (2001) are used in this paper to compare the performance of CMEA with NSGA-II.

While Deb's test functions represent problems in continuous space, the extended 0/1 multiobjective knapsack problem presented by Zitzler and Thiele [17] represents a problem in a combinatorial search space. This problem is a constrained, binary problem. Performance comparisons of several MOEAs in solving this problem that were presented by Zitzler and Thiele [17] are used in comparing the performance of CMEA. In addition, a noninferior set is generated using a mathematical programming model for the extended 0/1 knapsack problem. This model is solved using the binary linear programming solver CPLEX®.

Further, the non-convex, unconstrained multiobjective optimization test problem defined by Kursawe [12] is used to compare the performance of CMEA with results obtained using the micro-GA presented by Coello and Pulido [4]. The applicability of CMEA in

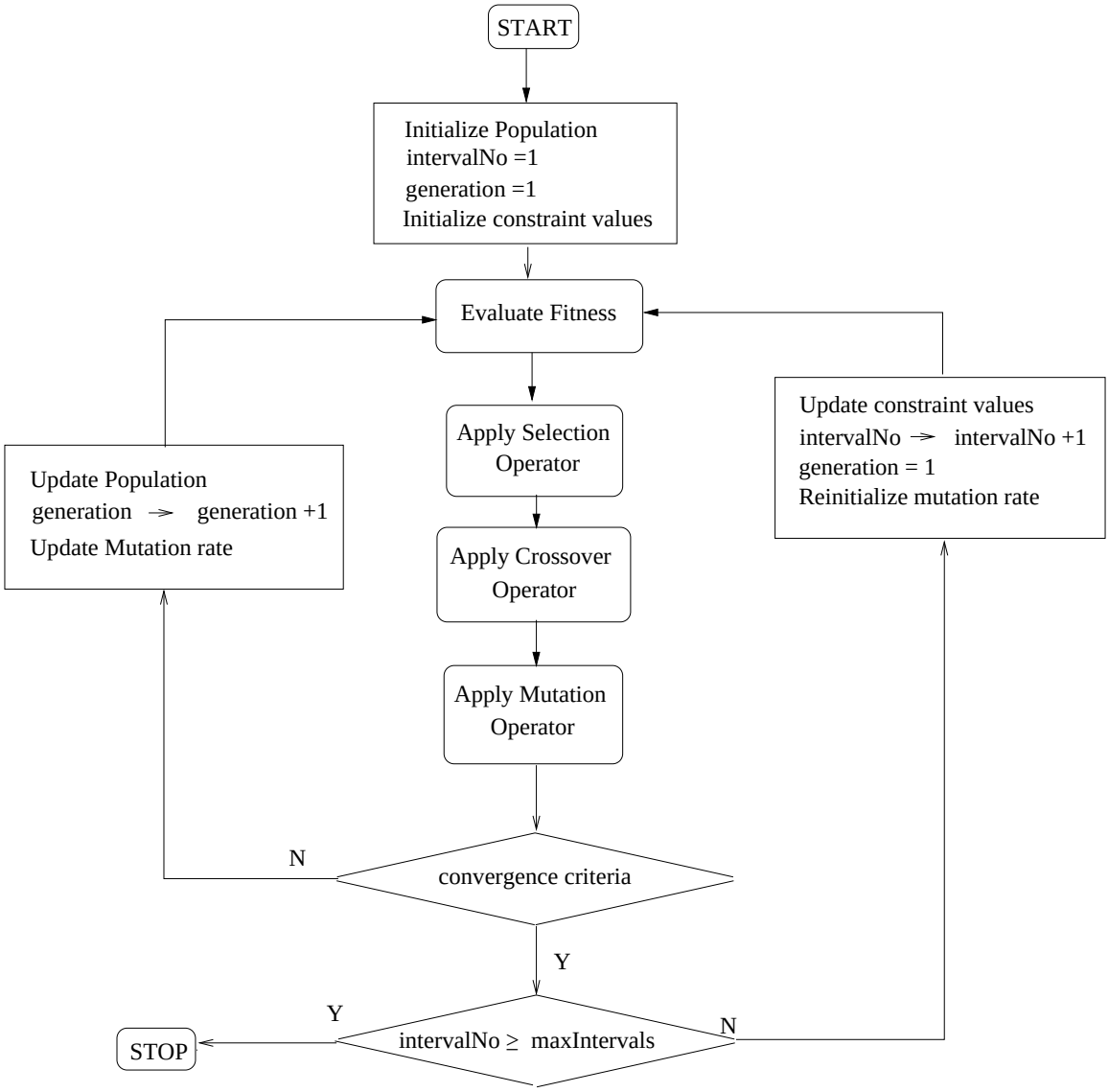


Figure 5.1: Flowchart for CMEA

solving problems with different characteristics (constrained vs. unconstrained, continuous vs. combinatorial, convex vs. nonconvex) is examined by applying it to these different problems. To test the robustness of the algorithm, the problems are solved repeatedly for different random seeds. As the results for the different random seeds were similar, a representative solution is used in the comparisons below.

5.4 Performance Metrics

The following performance criteria for 2-objective problems are used to evaluate CMEA, and to compare it with other approaches.

- *Accuracy* criterion is used to determine how close the generated noninferior solutions are to the best available prediction. Two different parameters are used to characterize the accuracy of an algorithm: The S factor used by Zitzler and Thiele [17] to compare accuracy, and the approach used by Knowles and Corne [11] to characterize the degree to which one noninferior set outperforms another. The same number of radial sampling lines used in the metric reported by Knowles and Corne [11] is used in the comparisons presented in this paper. An either-or criterion is used to determine if the noninferior set obtained by one MOEA dominates that obtained by another; the closeness of the two points of intersection are not differentiated statistically.
- *Spread* criterion determines the maximum range of the noninferior surface covered by the generated solutions. The spread parameters reported by Chetan [2] and Ranjithan et al. [14] are used in comparing CMEA with other MOEAs (a larger value of this metric indicates better spread of solutions). Using the illustration in Figure 5.2, points A and B refer to the two extreme noninferior solutions (corresponding to the single objective optima for each objective). The maximum range covered by the MOEA generated noninferior solutions represented by the ordered set $C = \{C_h, \forall h \in \{1, .., q\}\}$ is $(Z_1^{C_q} - Z_1^{C_1})$ and $(Z_2^{C_1} - Z_2^{C_q})$ in Z_1 and Z_2 objective space, respectively. The spread metrics in objective space Z_1 and Z_2 are defined as $(Z_1^{C_q} - Z_1^{C_1})/(Z_1^B - Z_1^A)$ and $(Z_2^{C_1} - Z_2^{C_q})/(Z_2^A - Z_2^B)$,

respectively.

- *Coverage* criterion represents how many different noninferior solutions are generated and how well they are distributed in the objective space. The Euclidean distance between adjacent noninferior points in the objective space is used as a quantitative measure to represent the distribution of the noninferior solutions generated by an MOEA. Two coverage metrics V1 and V2 are defined ([2],[14]) to characterize the coverage within the range of noninferior region defined by (1) the extreme noninferior solutions A and B, and (2) by the extreme solutions (C_1 and C_q) generated by that MOEA, respectively. Using the notations from Figure 5.2, V1 is defined as $Max\{d_h, \forall h \in \{0, 1, ..., q\}\}$, and V2 is defined as $Max\{d_h, \forall h \in \{1, ..., q-1\}\}$. A smaller value of V1 (or V2) implies more closely spaced noninferior solutions, thus indicating better coverage.

5.5 Results

The test problems considered in this paper are summarized in Table 5.1. These MO problems are solved using CMEA with algorithm-specific parameters shown in Table 5.2.

5.5.1 Comparison of Noninferior Solutions

Figures 5.3 to 5.8 compare the noninferior sets generated by NSGA-II and CMEA for CTP2-CTP7. The NSGA-II results are obtained from Deb et al. [9].

As shown in Figure 5.3, both algorithms are able to find solutions in the disconnected regions of the Pareto-optimal solutions for the CTP2 problem. CTP3 problem has only one solution in each disconnected Pareto-optimal region, and CMEA and NSGA-II are able to find solutions close to the true Pareto-optimal solutions. In CTP4, the problem definition parameters are chosen such that transition from continuous to discontinuous feasible region is far away from the Pareto-optimal region. It can be observed that NSGA-II did not perform well for CTP4, while CMEA is able to find solutions closer to the true noninferior set. For CTP5, the true Pareto-optimal solutions are scattered non-uniformly in the objective space, representing discrete and continuous regions. Both NSGA-II and

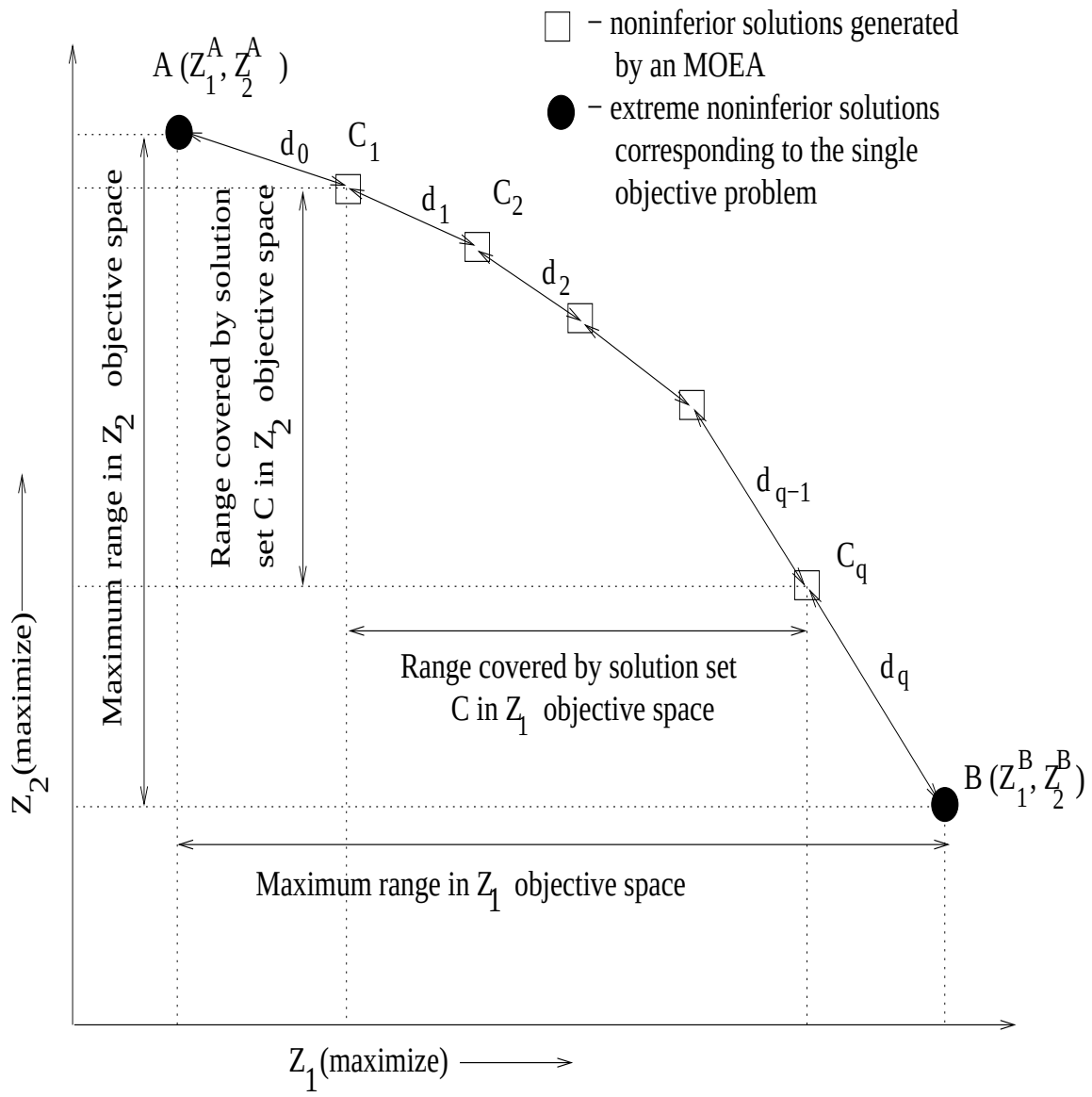


Figure 5.2: An example two-objective noninferior tradeoff to illustrate the computation of *Spread* and *Coverage* metrics.

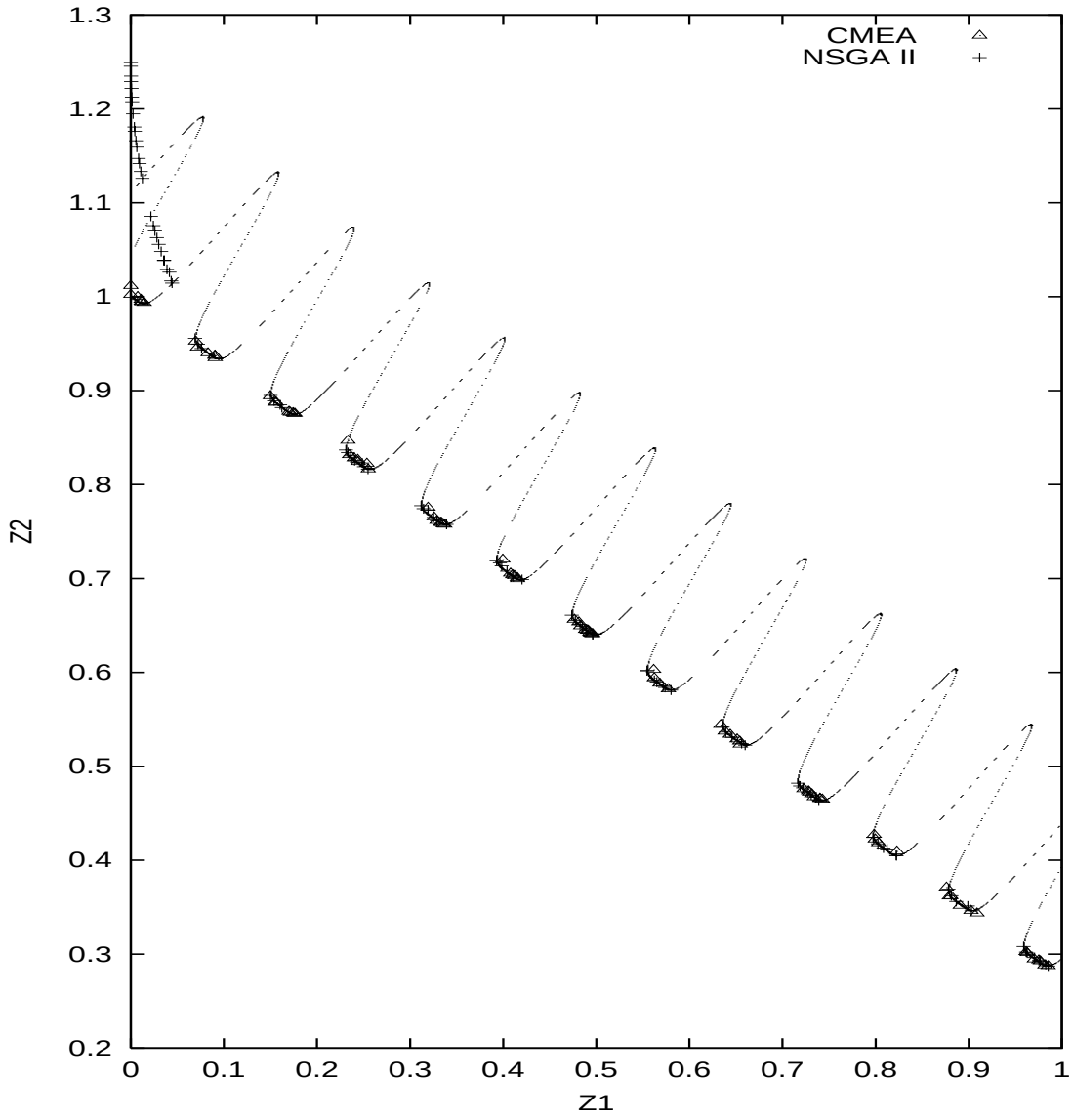


Figure 5.3: A comparison of the noninferior sets obtained using CMEA and NSGA-II for CTP2

CMEA are able to handle these complexities, and CMEA performs better at the extreme points in the objective space. CTP6 and CTP7 are problems where discontinuities in the search space are introduced. CTP6 has a search space with infeasible regions parallel to the Pareto-optimal front. Both MOEAs are able to converge to the correct feasible region and close to the true Pareto-optimal solutions. In CTP7, the infeasible regions in the search space are perpendicular to the Pareto-optimal front. Compared to NSGA-II, CMEA performs better in generating noninferior solutions more uniformly along the Parto-optimal front.

As shown in Table 5.1 and Figure 5.9, Kursawe [12] introduced a two-objective multiobjective optimization problem with discontinuous noninferior set that also includes concave and convex regions. The results of applying CMEA to this problem is shown in Figure 5.9 along with the results from Micro-GA developed by Coello and Pulido (2001).

Zitzler and Thiele [17] used in their work a knapsack problem that extends the traditional single objective knapsack problem by incorporating two knapsacks that can be filled by items selected from a larger collection of items. Similar to the traditional knapsack problem, the allocation of items to a knapsack is limited by its capacity, and each item in a knapsack has an associated payoff. The goal is to determine the allocation of items such that the the payoffs in each knapsack is maximized without exceeding the capacity constraints. This multiobjective problem is defined mathematically in Table 5.1. This binary MO problem is solved for two knapsacks (i.e., k = 2) with 250 items, and with 750 items. The results reported here correspond to $n = 750$ and $k = 2$. The data for the problems solved were adapted from Zitzler and Thiele [17].

The extended knapsack problem is solved by CMEA for the parameter setting shown in Table 5.2. In addition, by modeling the problem as a binary linear programming (BLP) model, the noninferior set is generated by solving this model using the binary linear programming solver, CPLEX®. In Figures 5.10 to 5.12, these results are shown along with the results reported by Zitzler and Thiele [17] for the following MOEAs: SPEA, NSGA-II, and PESA. The results from SPEA, NSGA-II and PESA required approximately 480,000 function evaluations (Zitzler and Thiele [16]), while CMEA required typically 270,000 function evaluations.

It can be seen that the solutions generated by CMEA, SPEA, NSGA-II and PESA are close to the BLP model-based estimate of the noninferior set, the best available for this problem. Although some of the noninferior solutions generated by SPEA, PESA, and NSGA-II dominate CMEA solutions in the middle region of the objective space, CMEA is able to provide better coverage by identifying good noninferior solutions that are broadly distributed over the objective space.

5.5.2 Comparison using Performance Metrics

The performance metrics for accuracy, coverage, and spread that were described earlier are computed for the noninferior solutions reported above. A summary of the performance metrics for the various problems discussed are shown in Tables 5.3 to 5.6. Based on the accuracy parameter defined by Knowles and Corne (2000), CMEA outperforms the other MOEAs in all comparisons (Table 5.3) considered in this paper. The S factor values generated by CMEA for the CTP problems are comparable or better than that of NSGA-II (Table 5.4). As shown in this table, CMEA outperforms Micro-GA in Kursawe's problem, and NSGA-II, SPEA, and PESA in the Knapsack problem.

The spread metric for the different problems is compared in Table 5.5. The values of CMEA and NSGA-II are comparable for CTP2, CTP3, CTP5, and CTP6. For problems CTP3 and CTP7, the spread of the noninferior sets generated by CMEA is considerably better than that of NSGA-II. The broader spread and even coverage of the CMEA solutions for the extended 0/1 knapsack problem compared to that of SPEA, NSGA-II, and PESA is reflected in the spread metric and the coverage metrics compared in Tables 5.5 and 5.6. When the coverage (V1) is measured in consideration of the extreme noninferior solutions, CMEA performs better in all instances. Measuring the coverage (V2) of an MOEA within the range of only the noninferior solutions generated by that MOEA, the other MOEAs provide better coverage compared to that by CMEA. This implies that while other MOEAs are able to find many closely spaced noninferior solutions within a narrower range, CMEA is better at finding more evenly distributed noninferior solutions over a broader range of the noninferior space. This result is reflected in the coverage metric values for the Knapsack problem, where SPEA, NSGA-II and PESA provide better distribution (based on V2 metric)

within the narrower noninferior range represented by their solutions, and CMEA provides the best coverage of solutions in the entire noninferior range (based on V1 metric).

5.6 Summary and Conclusions

This paper presents a comparative study of CMEA that is founded upon the ϵ-constraint method, which has been well established in the multiobjective optimization literature within operations research. The convergence of this method is enhanced by beneficial seeding of the population within the subiterations of the algorithm, a notion borrowed from mathematical programming-based MO analysis. Unlike other commonly reported MOEAs that attempt to converge the population of solutions simultaneously to the noninferior set, CMEA attempts to converge first the population of solutions to an extreme noninferior solution and then incrementally migrate the population to trace the noninferior surface. As no new algorithm-specific operators or special encoding are needed, the structure of the algorithm enables easy integration with existing implementation of evolutionary algorithms for an optimization problem. This is important when analyzing large-scale realistic problems for which much effort is already spent on configuring and implementing the base evolutionary algorithms.

To evaluate the performance of CMEA in solving MO problems, it was applied to a number of test problems with different characteristics and levels of difficulty. The test problems included problems involving continuous as well as combinatorial decision space, unconstrained as well as constrained optimization, real as well as binary variables, and concave as well as convex Pareto optimal sets. The performance of CMEA was compared with that of other commonly reported MOEAs (namely, NSGA-II, SPEA, PESA, and MicroGA). To characterize the performance of the algorithms, several metrics quantifying accuracy, spread, and coverage were used. Solutions were generated for several different random seeds, and CMEA showed robust behavior in generating the noninferior solutions. Overall, CMEA performed well with respect to these criteria and for the different problems tested. CMEA showed consistently good coverage and spread of solutions in the noninferior space. While SPEA, PESA, and NSGA-II performed better in a narrower range of the noninferior solutions for a few test problems, CMEA performed better over a broader

range of the noninferior set. For the problems tested in this paper, the number of function evaluations (used as an approximate surrogate for computational effort) was less than those reported for the other MOEAs.

Further testing is needed to evaluate the effectiveness of applying CMEA to higher-order MO problems (i.e., more conflicting objectives). Preliminary results from a few 3-objective test cases show highly promising outcome. Increases in computational requirements as the number of objectives are scaled up need further investigation. Scale up effect, however, is expected to be comparable with other MOEAs, which must increase the population size to handle higher dimensional MO problems. CMEA is currently being applied to several real-world engineering problems that require MO analysis. The practicality of these realistic applications is being assessed and will be reported later.

5.7 Acknowledgements

This research was supported in part by an NSF CAREER award (BES 9733788) to S. Ranji Ranjithan, and by the Department of Civil Engineering at North Carolina State University. The viewpoints presented here are those of the authors, and do not necessarily reflect those the funding agencies. The authors would like to thank also Drs. Kalyanmoy Deb, Eckart Zitzler, and Carlos Coello for sharing their results obtained using different algorithms for the various test problems reported in this paper.

Table 5.1: Test problems used in this study. The objective functions are denoted by $Z_l(x)$, $1 \leq l \leq k$, where k denotes the number of objectives and N the number of decision variables.

N	Domain	Objective Functions	Constraints
		CTP2-CTP7 by Deb et.al [9]	
5	$[-5.12{:}5.12]^N$	Minimize $Z_1(\mathbf{x}) = x_1$ Minimize $Z_2(\mathbf{x}) = g(\mathbf{x})(1 - \frac{Z_1(\mathbf{x})}{g(\mathbf{x})})$ where $g(x) = 10n$ $+ \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$	$c(\mathbf{x}) \equiv cos(\theta)(Z_2(\mathbf{x}) - e) -$ $sin(\theta)Z_1(\mathbf{x}) \geq a\lvert sin(b\pi$ $(sin(\theta)(Z_2(\mathbf{x}) - e)$ $+cos(\theta)Z_1(\mathbf{x}))^c)\rvert^d$
CTP2: $\theta = -0.2\pi$, $a = 0.2$, $b = 10$, $c = 1$, $d = 6$, $e = 1$			
CTP3: $\theta = -0.2\pi$, $a = 0.1$, $b = 10$, $c = 1$, $d = 0.5$, $e = 1$			
CTP4: $\theta = -0.2\pi$, $a = 0.75$, $b = 10$, $c = 1$, $d = 0.5$, $e = 1$			
CTP5: $\theta = -0.2\pi$, $a = 0.75$, $b = 10$, $c = 2$, $d = 0.5$, $e = 1$			
CTP6: $\theta = 0.1\pi$, $a = 40$, $b = 0.5$, $c = 1$, $d = 2$, $e = -2$			
CTP7: $\theta = -0.05\pi$, $a = 40$, $b = 5$, $c = 1$, $d = 6$, $e = 0$			
		Kursawe's Function by Kursawe [12]	
3	$[-5{:}5]^N$	Minimize $Z_1(\mathbf{x}) =$ $\sum_{i=1}^{n-1}(-10exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}))$ Minimize $Z_2(\mathbf{x}) =$ $\sum_{i=1}^{n}(\lvert x_i\rvert^{0.8} + 5sin(x_i)^3)$	Unconstrained
		Multiobjective Knapsack Problem (Zitzler and Thiele [17])	
750	$\{0,1\}^N$	Maximize $Z_l(\mathbf{x}) =$ $\sum_{j=1}^{n} p_{l,j}x_j \; \forall l = 1, 2, ..., k$	$\sum_{j=1}^{n} w_{l,j}x_j \leq c_l$ $\forall l = 1, 2, ..., k$
$Z_l(\mathbf{x})$ is the total profit associated with knapsack l, $p_{l,j}$ = profit of placing item j in knapsack l, $w_{l,j}$ = weight of item j when placed in knapsack l, c_l = capacity of knapsack l, $\mathbf{x} = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ such that $x_j = 1$ if selected and $= 0$ otherwise, n is the number of available items, and k is the number of knapsacks.			

Table 5.2: CMEA parameters and settings used in solving the test problems

Problem	Variable Type	CMEA parameters			
		No.of intervals	Pop. size	Encoding	Crossover
CTP2-CTP7	Real	100	100	20 bit Binary	Uniform
Kursawe	Real	100	100	20 bit Binary	Uniform
Knapsack	Binary	100	100	Binary	Uniform

Table 5.3: *Accuracy* comparison, based on the metric defined by Knowles and Corne [10], of CMEA with NSGA-II, SPEA, PESA, and Micro-GA for different test problems; The best is shown in bold.

The MOEAs Compared ($MOEA_1$ vs. $MOEA_2$)	Problem instance	(P_1,P_2): (Percentage number of times $MOEA_1$ outperforms $MOEA_2$, Percentage number of times $MOEA_2$ outperforms $MOEA_1$)		
		Number of Sampling Lines		
		108	507	1083
(CMEA vs NSGA-II)	CTP2	(**55.55**, 44.44)	(**55.82**, 44.18)	(**55.87**, 44.12)
(CMEA vs NSGA-II)	CTP3	(**78.57**, 21.43)	(**77.44**, 22.56)	(**77.20**, 22.80)
(CMEA vs NSGA-II)	CTP4	(**100.0**, 0.0)	(**100.0**, 0.0)	(**100.0**, 0.0)
(CMEA vs NSGA-II)	CTP5	(**69.70**, 30.30)	(**69.97**, 30.02)	(**69.97**, 30.02)
(CMEA vs NSGA-II)	CTP6	(**76.32**, 23.68)	(**74.57**, 25.42)	(**74.54**, 25.46)
(CMEA vs NSGA-II)	CTP7	(**100.0**, 0.0)	(**100.0**, 0.0)	(**100.0**, 0.0)
(CMEA, Micro-GA)	Kursawe	(**86.24**, 13.76)	(**89.94**, 10.06)	(**90.12**, 9.88)
(CMEA, SPEA)	Knapsack	(**95.37**, 4.63)	(**95.46**, 4.54)	(**95.94**, 4.06)
(CMEA, NSGA-II)	Knapsack	(**94.50**, 5.50)	(**94.28**, 5.72)	(**94.36**, 5.63))
(CMEA, PESA)	Knapsack	(**95.41**, 4.59)	(**95.07**, 4.93)	(**95.11**, 4.89)

Table 5.4: *Accuracy* comparison, based on the S factor (Zitzler and Thiele [14]), of CMEA with NSGA-II, SPEA, PESA and Micro-GA for different test problems. A larger value indicates better performance; the best is shown in bold.

The MOEAs Compared ($MOEA_1$ vs $MOEA_2$)	Problem instance	(S_1,S_2): (S factor for $MOEA_1$ data set, S factor for $MOEA_2$ data set)
(CMEA vs NSGA-II)	CTP2	(**0.6123**, 0.6075)
(CMEA vs NSGA-II)	CTP3	(**0.5931**, 0.5823)
(CMEA vs NSGA-II)	CTP4	(**0.7808**, 0.6004)
(CMEA vs NSGA-II)	CTP5	(**0.5997**, 0.5923)
(CMEA vs NSGA-II)	CTP6	(0.5593, **0.5663**)
(CMEA vs NSGA-II)	CTP7	(**0.6563**, 0.1543)
(CMEA vs Micro-GA)	Kursawe	(**0.4011**, 0.3976)
(CMEA vs SPEA)	Knapsack	(**0.7075**, 0.6068)
(CMEA vs NSGA-II)	Knapsack	(**0.7075**, 0.6235)
(CMEA vs PESA)	Knapsack	(**0.7075**, 0.6025)

Table 5.5: *Spread* comparison of CMEA with NSGA-II, SPEA, PESA and Micro-GA for different test problems. A larger value indicates better performance; the best is shown in bold.

The MOEAs Compared ($MOEA_1$ vs $MOEA_2$)	Problem instance	($Z1$ spread for $MOEA_1$, $Z1$ spread for $MOEA_2$)	($Z2$ spread for $MOEA_1$, $Z2$ spread for $MOEA_2$)
(CMEA vs NSGA-II)	CTP2	(**1.0045**, 1.0044)	(1.0173, **1.3510**)
(CMEA vs NSGA-II)	CTP3	(**0.9977**, 0.9962)	(1.1837, **1.3695**)
(CMEA vs NSGA-II)	CTP4	(**0.9830**, 0.4971)	(1.0351, **1.4375**)
(CMEA vs NSGA-II)	CTP5	(0.9927, **0.9952**)	(1.0203, **1.3756**)
(CMEA vs NSGA-II)	CTP6	(**0.9995**, 0.9744)	(**1.1453**, 0.9735)
(CMEA vs NSGA-II)	CTP7	(**0.9895**, 0.0364)	(**1.3353**, 0.2337)
(CMEA vs Micro-GA)	Kursawe	(0.9984, **0.9985**)	(0.9964, **0.9984**)
(CMEA vs SPEA)	Knapsack	(**0.9224**, 0.3428)	(**0.9207**, 0.3512)
(CMEA vs NSGA-II)	Knapsack	(**0.9224**, 0.2691)	(**0.9207**, 0.2960)
(CMEA vs PESA)	Knapsack	(**0.9224**, 0.2087)	(**0.9207**, 0.2610)

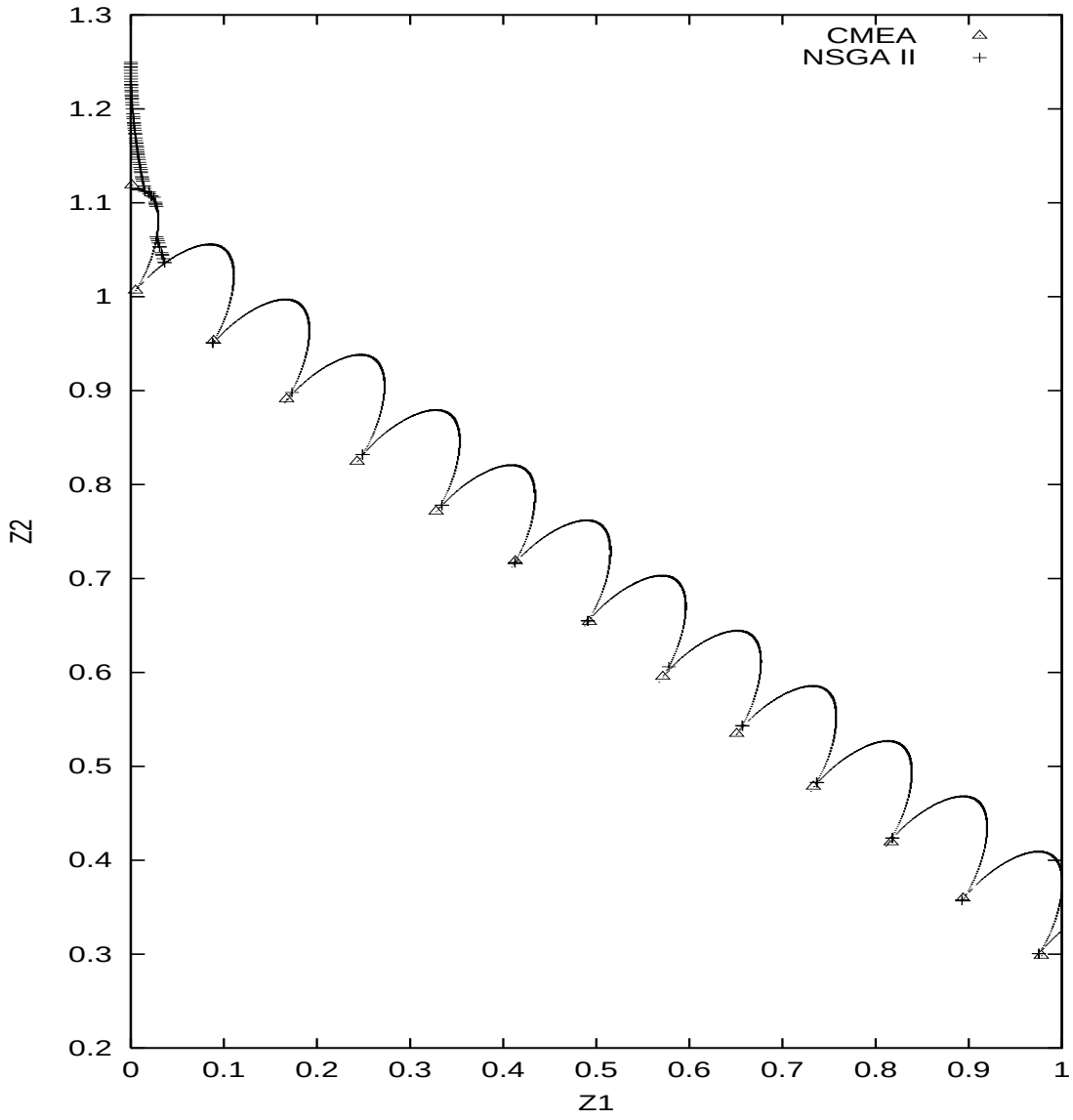


Figure 5.4: A comparison of the noninferior sets obtained using CMEA and NSGA-II for CTP3

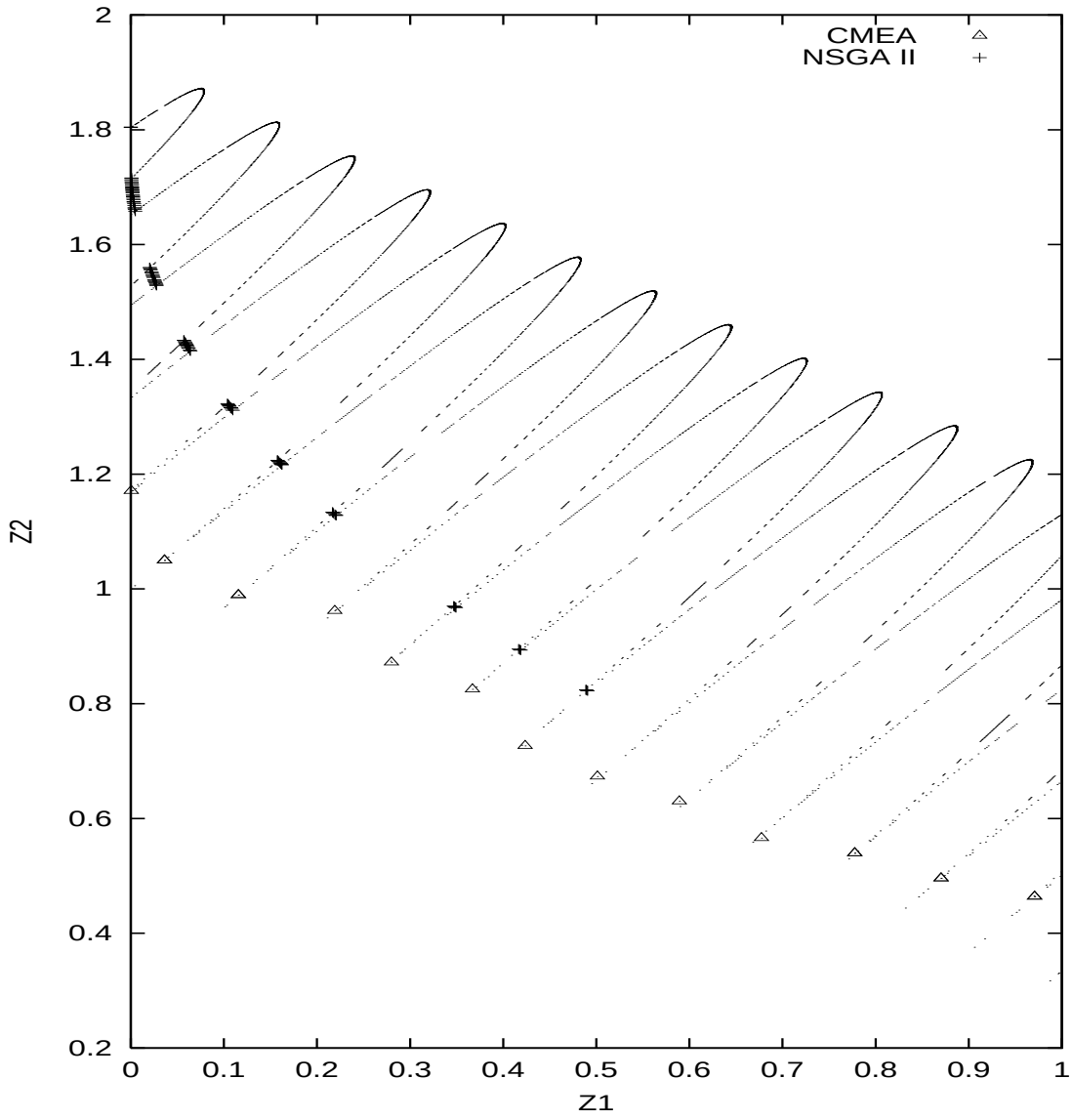


Figure 5.5: A comparison of the noninferior sets obtained using CMEA and NSGA-II for CTP4

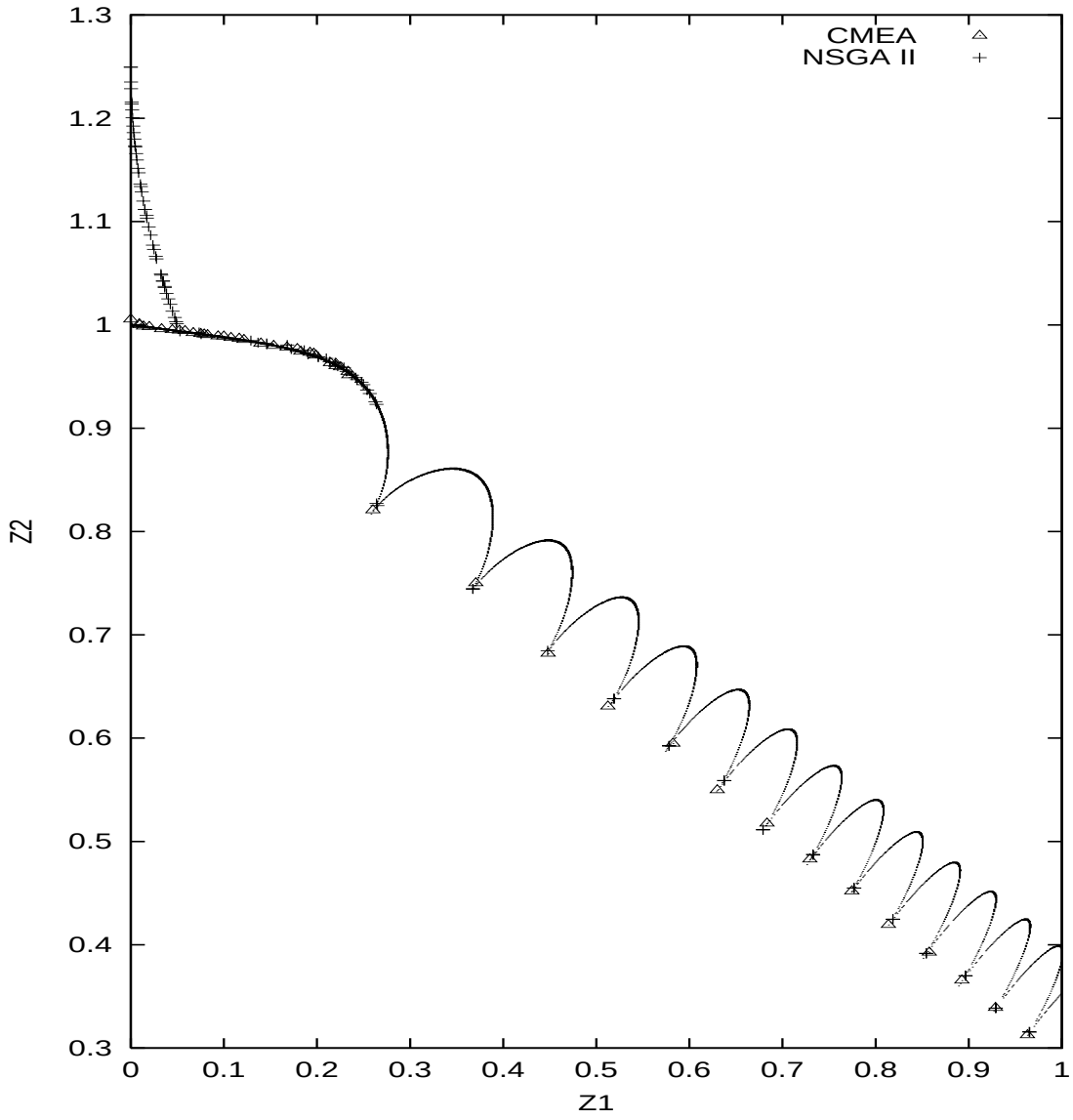


Figure 5.6: A comparison of the noninferior sets obtained using CMEA and NSGA-II for CTP5

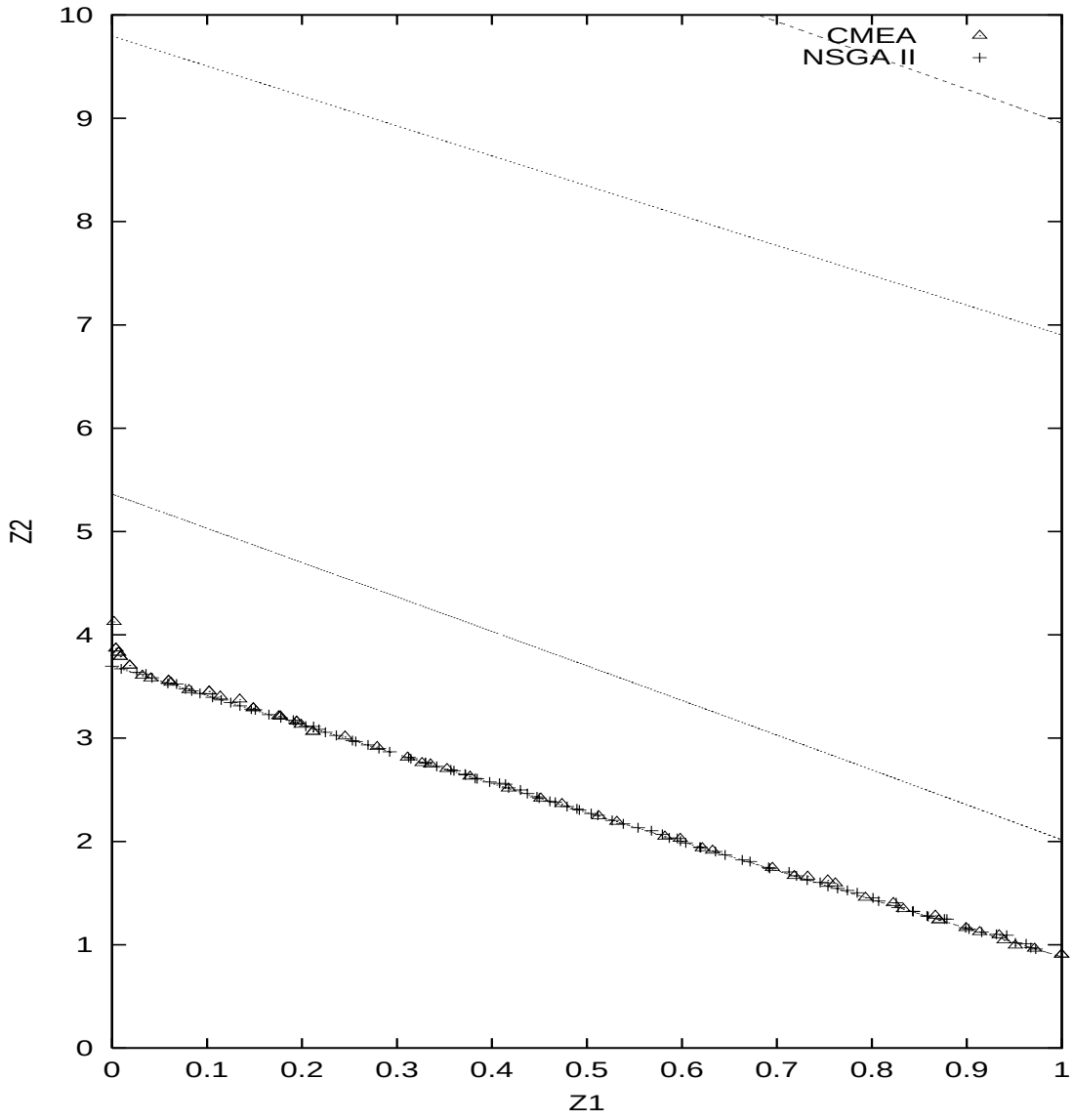


Figure 5.7: A comparison of the noninferior sets obtained using CMEA and NSGA-II for CTP6

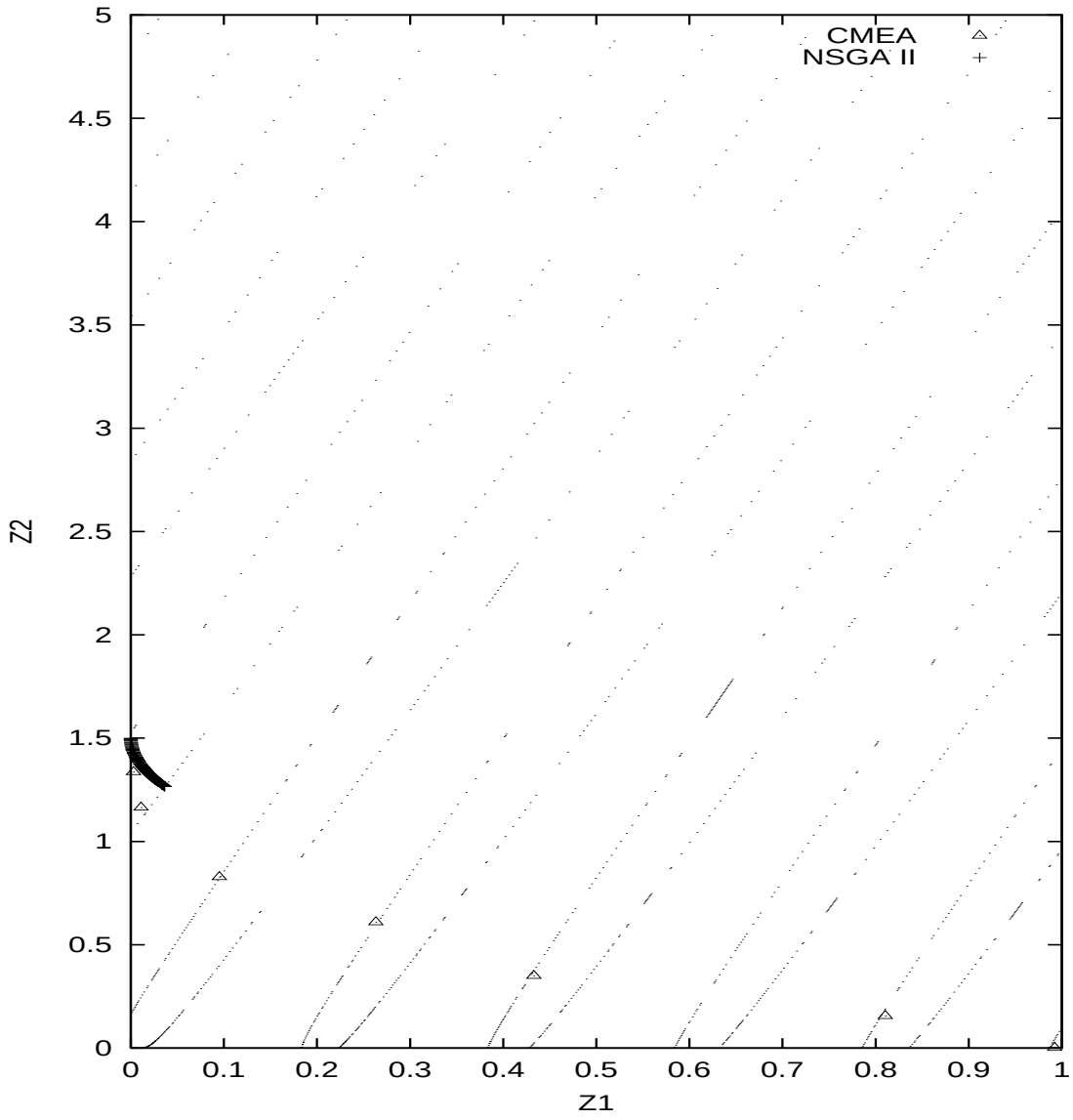


Figure 5.8: A comparison of the noninferior sets obtained using CMEA and NSGA-II for CTP7

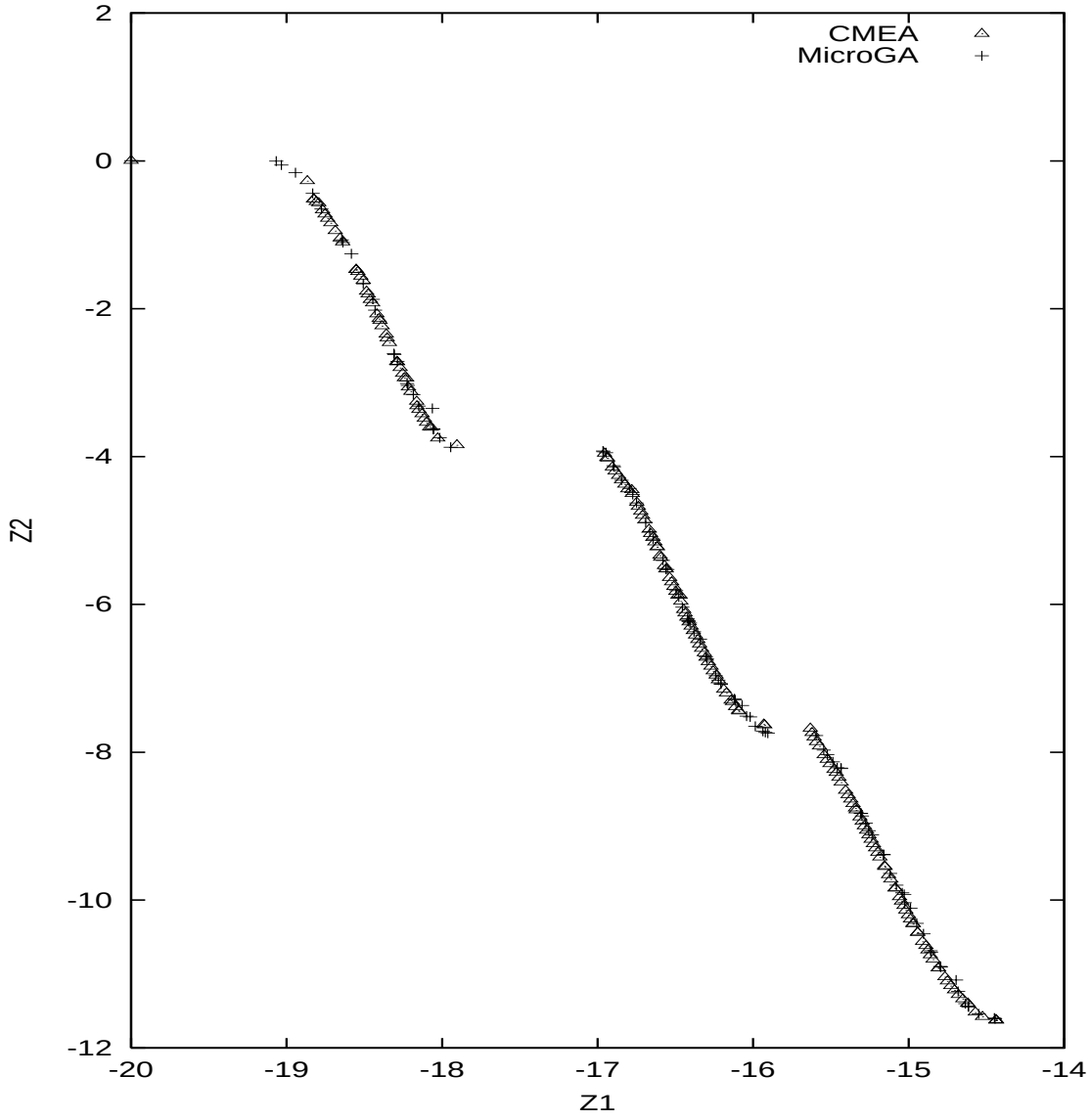


Figure 5.9: A comparison of the noninferior sets obtained using CMEA and Micro-GA for Kursawe's function

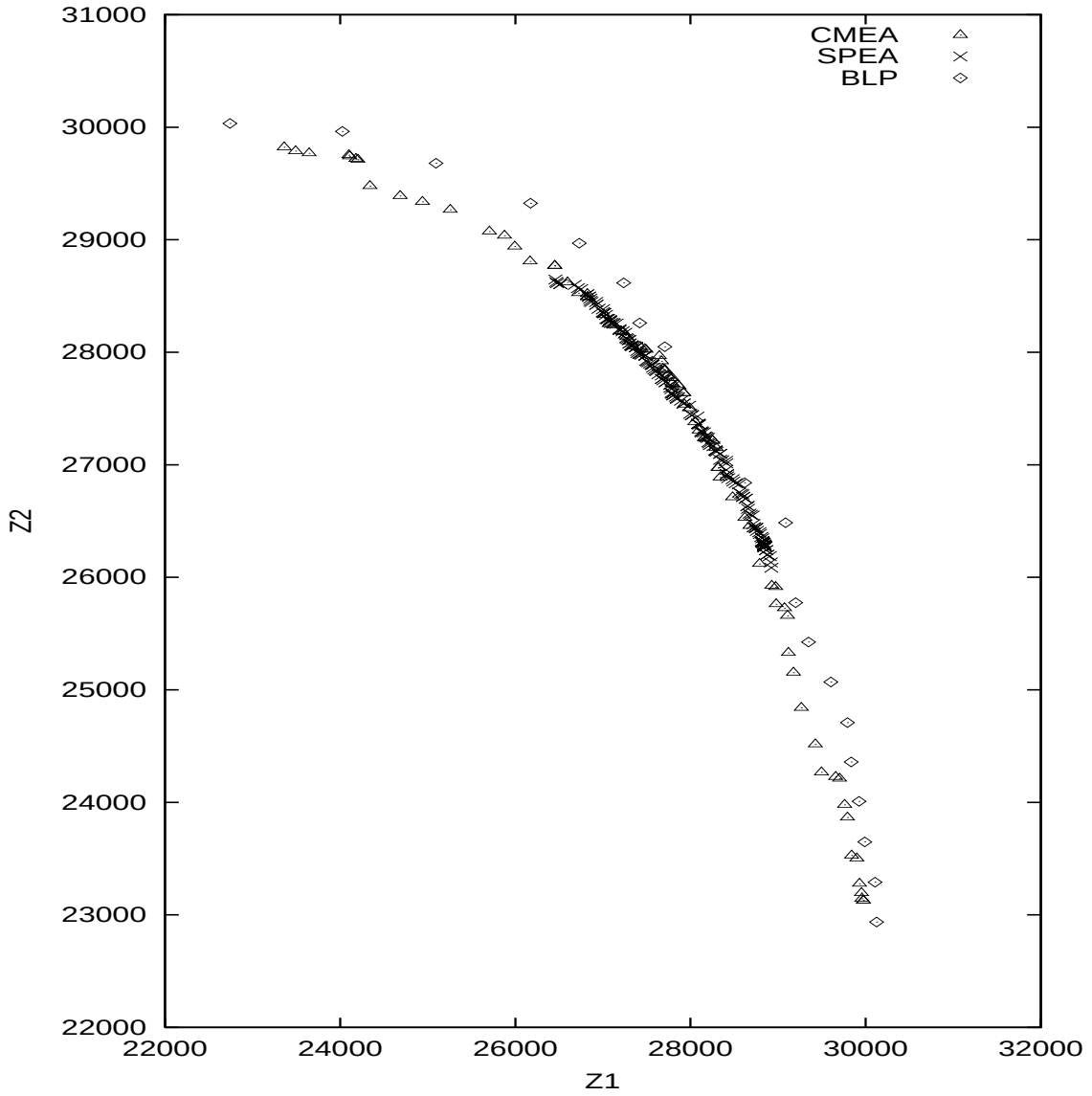


Figure 5.10: A comparison of the noninferior sets obtained using CMEA, NSGA-II, and BLP for the extended 0/1 multiobjective knapsack problem.

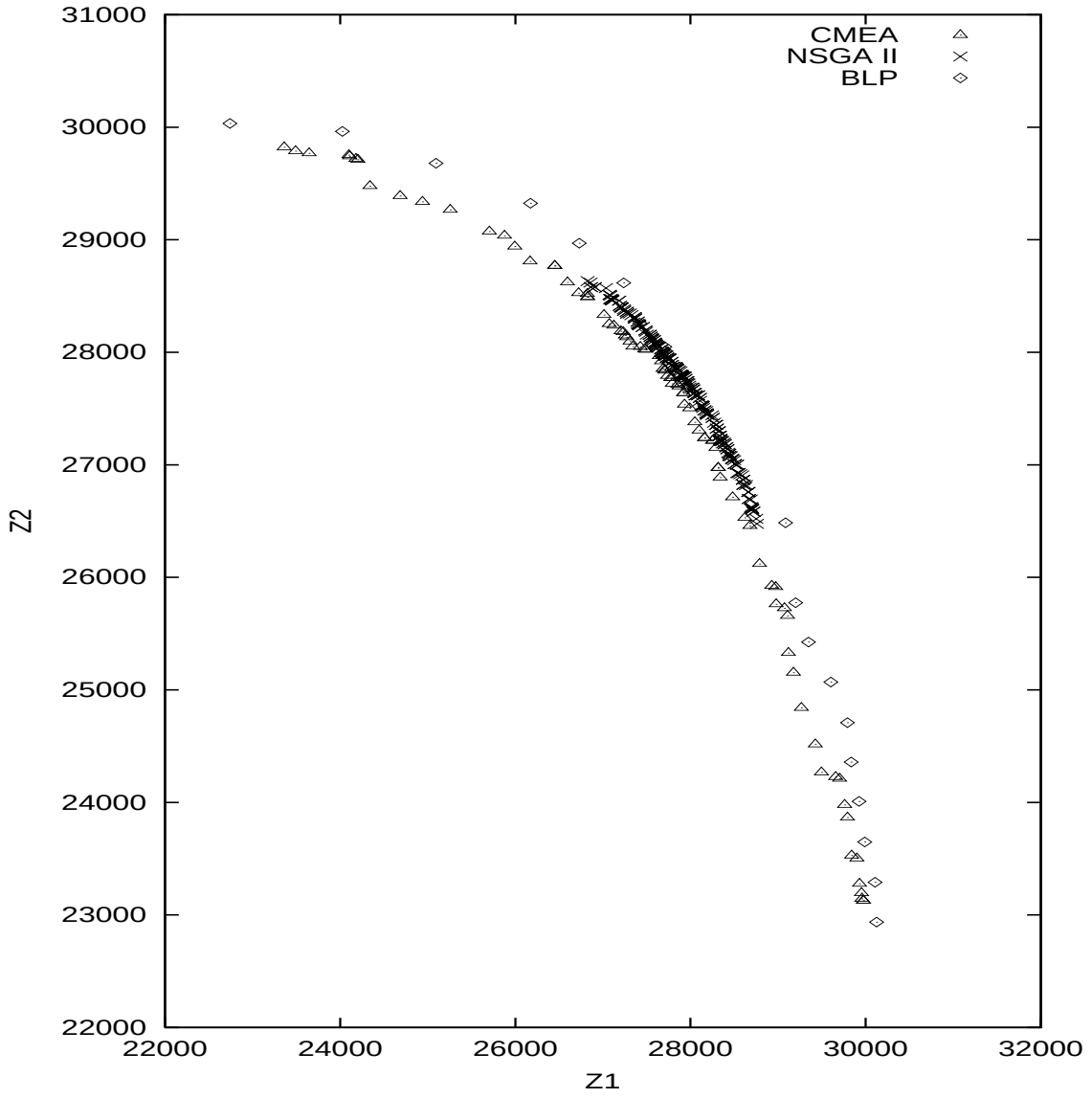


Figure 5.11: A comparison of the noninferior sets obtained using CMEA, SPEA, and BLP for the extended 0/1 multiobjective knapsack problem.

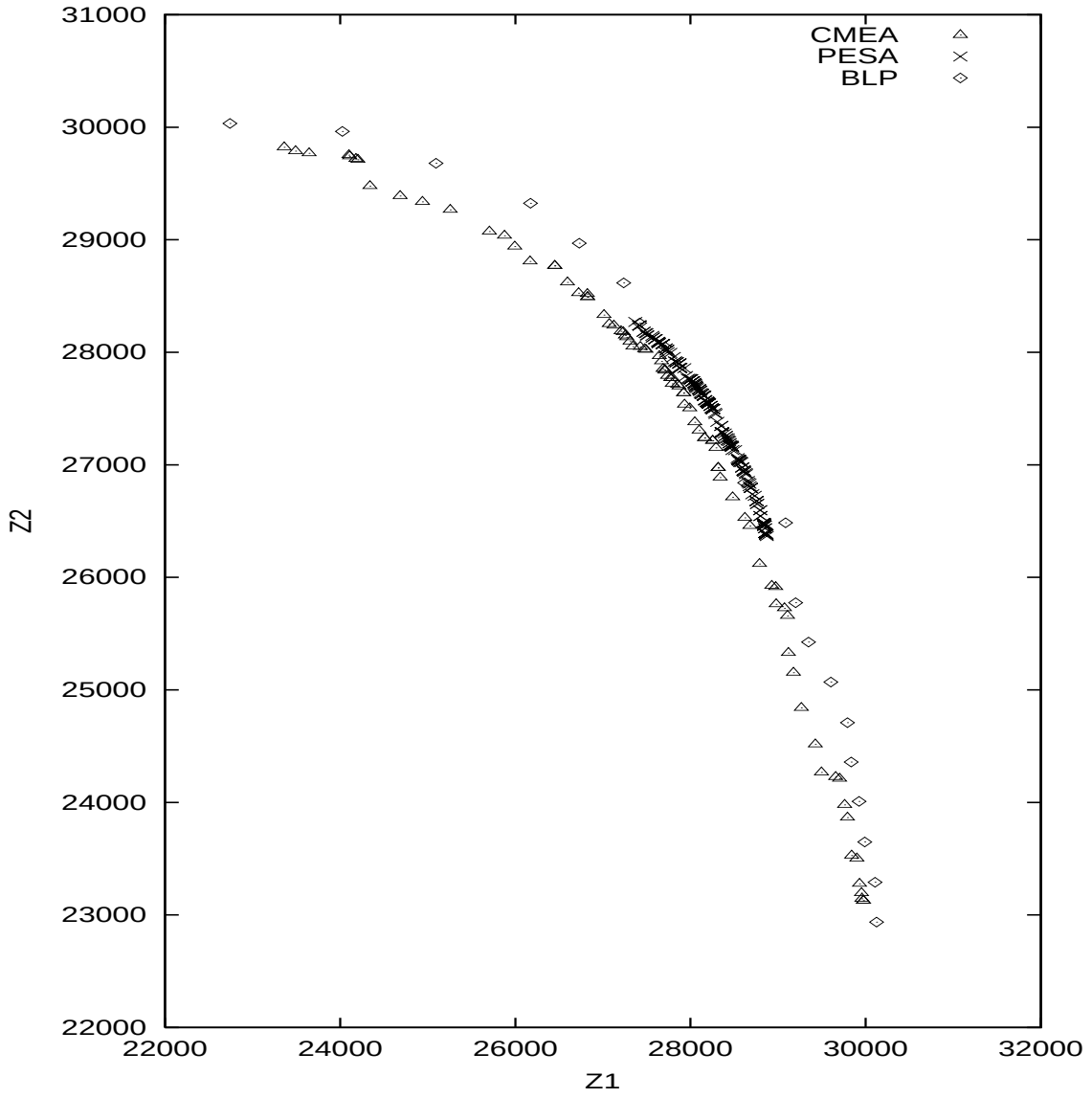


Figure 5.12: A comparison of the noninferior sets obtained using CMEA, PESA, and BLP for the extended 0/1 multiobjective knapsack problem.

Table 5.6: *Coverage* comparison of CMEA with NSGA-II, SPEA, PESA and Micro-GA for different test problems. A smaller value indicates better performance; the best is shown in bold.

The MOEAs Compared ($MOEA_1$ vs $MOEA_2$)	Problem instance	($V1$ for $MOEA_1$, $V1$ for $MOEA_2$) (includes the known extreme points for each objective)	($V2$ for $MOEA_1$,$V2$ for $MOEA_2$) (excludes the known extreme points for each objective)
(CMEA vs NSGA-II)	CTP2	(**0.0965**, 0.3500)	(**0.0965**, 0.1029)
(CMEA vs NSGA-II)	CTP3	(**0.1713**, 0.3601)	(0.1619, **0.1332**)
(CMEA vs NSGA-II)	CTP4	(**0.2501**, 1.1779)	(**0.1811**, 0.7159)
(CMEA vs NSGA-II)	CTP5	(**0.1942**, 0.3675)	(0.1942, **0.1592**)
(CMEA vs NSGA-II)	CTP6	(0.1525, **0.0383**)	(0.0916, **0.0355**)
(CMEA vs NSGA-II)	CTP7	(**0.4249**, 1.5907)	(0.4249, **0.0070**)
(CMEA vs Micro-GA)	Kursawe	(0.2050, **0.1762**)	(0.2050, **0.1762**)
(CMEA vs SPEA)	Knapsack	(**0.0890**, 0.5399)	(0.0666, **0.0224**)
(CMEA vs NSGA-II)	Knapsack	(**0.0890**, 0.5870)	(0.0666, **0.0175**)
(CMEA vs PESA)	Knapsack	(**0.0890**, 0.6743)	(0.0666, **0.0108**)

References

[1] V. Chankong and T. T. Haimes. *Multiobjective Decision Making: Theory and Methodology*. North-Holland, New York, 1983.

[2] S. K. Chetan. Noninferior surface tracing evolutionary algorithm (NSTEA) for multiobjective optimization. Master's thesis, North Carolina State University, Raleigh, NC., 2000.

[3] C. A. C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information System*, 1(3):269–308, 1999.

[4] C. A. C. Coello and G. T. Pulido. A micro-genetic algorithm for multiobjective optimization. In E. Zitzler et al., editor, *Evolutionary Multi-Criteria Optimization 2001*, Lecture Notes in Computer Science 1993, pages 126–140. Springer-Verlag, 2001.

[5] J. L. Cohon. *Multiobjective Programming and Planning*, volume 140 of *Mathematics in Science and Engineering*. Academic Press Inc., 1978.

[6] D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto-envelope based selection algorithm for multiobjective optimixation. In M. Schoenauer, Deb K., Rudolph G., X. Yao, Lutton E., J. J. Merelo, and H-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI*, Lecture Notes in Computer Science, pages 869–878. Springer Verlag, 2000.

[7] K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, Ltd., England, 2001.

[8] K. Deb, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multiobjective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI*, pages 849–858, 2000.

[9] K. Deb, A. Pratap, and T. Meyarivan. Constrained test problems for multi-objective evolutionary optimization. In E. Zitzler et al., editor, *Evolutionary Multi-Criteria Optimization 2001*, Lecture Notes in Computer Science 1993, pages 284–298. Springer-Verlag, 2001.

[10] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[11] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[12] F. Kursawe. A variant of evolution strategies for vector optimization. In H. P. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature - PPSN I*, Lecture Notes in Computer Science 496, pages 193–197. Springer Verlag, 1991.

[13] K. M. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, MA., 1999.

[14] S. R. Ranjithan, S. K. Chetan, and H. K. Dakshina. Constraint method-based evolutionary algorithm (CMEA) for multiobjective optimization. In E. Zitzler et al., editor, *Evolutionary Multi-Criteria Optimization 2001*, Lecture Notes in Computer Science 1993, pages 299–313. Springer-Verlag, 2001.

[15] Veldhuizen Van and G. B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147, 2000.

[16] E. Zitzler, M. Laumanns, and L. Thiele. Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Communications Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Gloriastrasse 35, CH-8092, 2001.

[17] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 2(4):257–271, 1999.

Chapter 6

Noninferior Surface Tracing Evolutionary Algorithm in *Vitri*

6.1 Introduction

This chapter describes the Noninferior Surface Tracing Evolutionary Algorithm (NSTEA) in *Vitri*. Chetan [1], in addition to introducing the technique, also presented the evaluation of NSTEA to a number of multiobjective optimization problems. This chapter presents the results of applying NSTEA to a suite of constrained test problems presented by Deb et al. [3]. The performance of the algorithm is evaluated by using the quantitative metrics presented in the previous chapter.

6.2 Background

The NSTEA algorithm is implemented by incorporating the concepts of the mathematical programming-based weighting approach into an evolutionary algorithm framework. Similar to an objective aggregation approach, a linearly weighted function of all objective functions is used to evaluate fitness at each intermediate step to enforce Pareto optimality of a solution. To maintain generality, normalized objective function values are used. NSTEA attempts to identify the noninferior set by varying the weight vectors throughout the execution of the evolutionary algorithm. A linearly weighted fitness function Z_{ag}, computed

as:

$$Z_{ag} = \sum_{i=1}^{k} w_i \bar{Z}_i \tag{6.1}$$

where, $\mathbf{w} = \{w_i : i = 1, 2, ...k\}$ is the weight vector, w_i is the i^{th} weight and $\bar{Z}_i$ is the i^{th} normalized objective function value. The weight w_i is a fractional number such that

$$\sum_{i=1}^{k} w_i = 1 \tag{6.2}$$

The key steps of NSTEA is shown as a flowchart in Figure 6.1. Similar to the CMEA, NSTEA exploits the basic concept that for some class of problems, adjacent solutions in the decision space map to adjacent solutions in the objective space. This enables the beneficial use of the final population corresponding to the current noninferior solution to seed the search of an adjacent noninferior solution. The new search is conducted with updated weight vector **w** to represent an adjacent noninferior point in the objective space. When the new selection pressure manifesting from the updated weight vector is applied on the previous population, the population quickly migrates to an adjacent noninferior solution. A systematic update of the weight vector enables an efficient mechanism for incrementally tracing the noninferior set. This incremental population migration approach significantly reduces the computational burden compared to that required when solving each single objective EA as independent search problems.

6.3 Testing and Evaluation of NSTEA

In the previous chapter, a suite of test problems presented by Deb et al [3] was used to compare the performance of CMEA. The same suite of Deb's functions are used in this chapter to evaluate the performance of NSTEA. The results obtained by the NSTEA are compared with those reported using the NSGA-II approach.

Figures 6.2 to 6.7 compare the noninferor set of solutions obtained by NSGA-II and NSTEA for problems CTP2-CTP7. Table 6.1 represents the algorithm-specific parameters used in solving the problems.

As shown in Figure 6.2, 6.3, and 6.5, the NSGA-II was able to find all disconnected noninferior solutions for the CTP2, CTP3, and CTP5 problems, whereas the noninferior

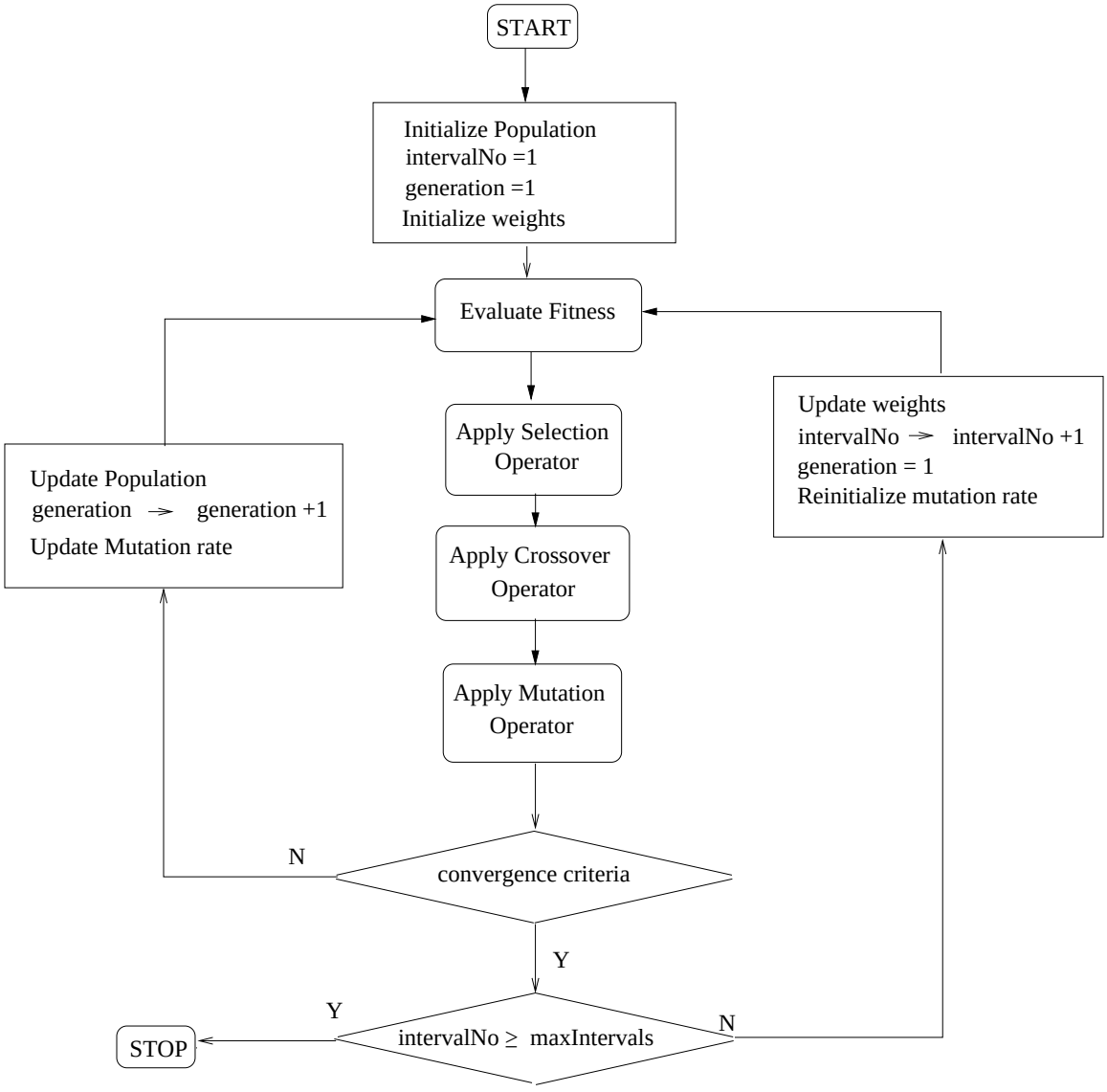


Figure 6.1: Flowchart for NSTEA

Table 6.1: NSTEA parameters and settings used in solving the test problems

Variable Type	NSTEA parameters			
	No.of intervals	Pop. size	Encoding	Crossover
Real	100	100	20 bit Binary	Uniform

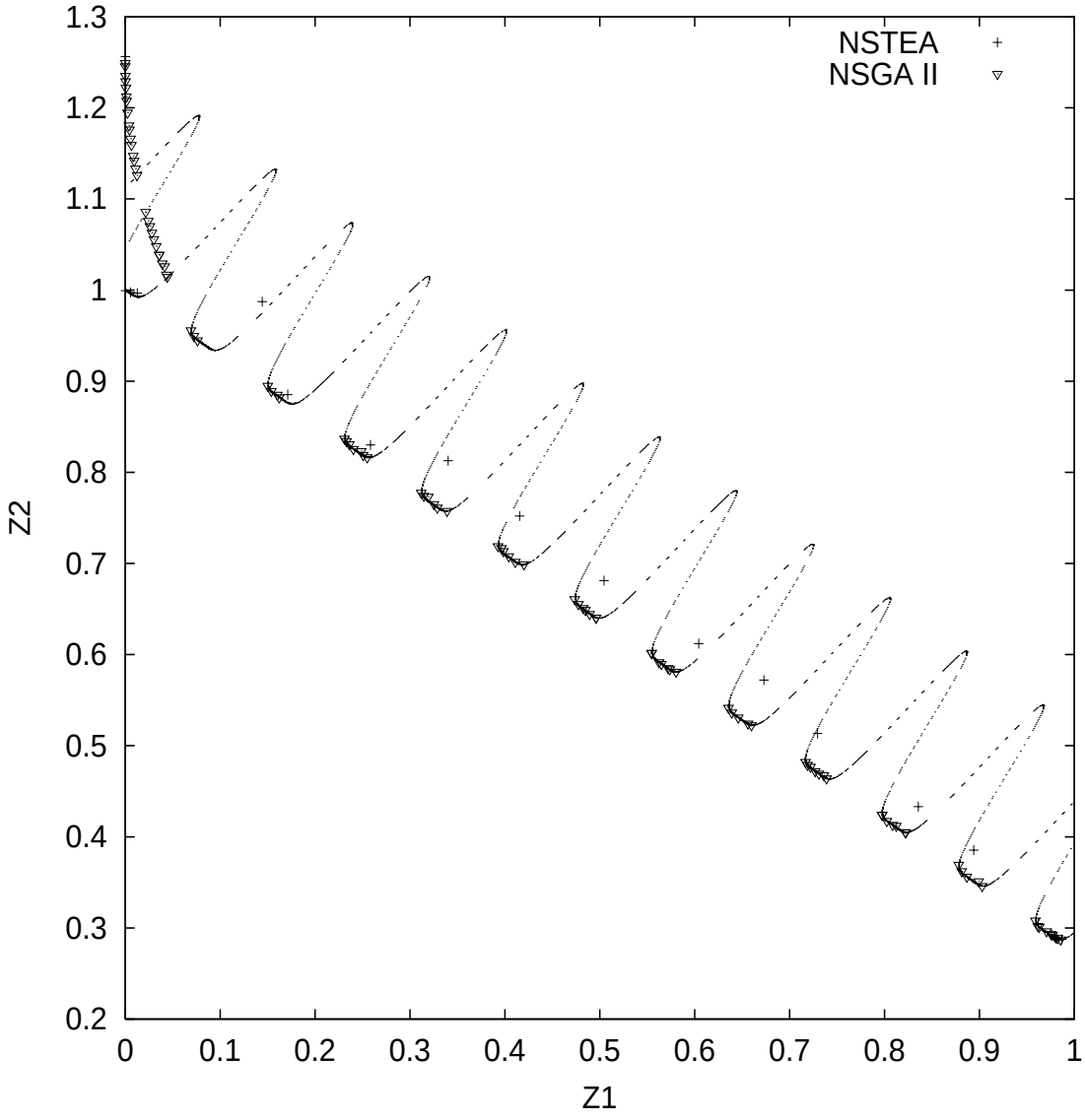


Figure 6.2: A comparison of the noninferior sets obtained using NSTEA and NSGA-II for CTP2

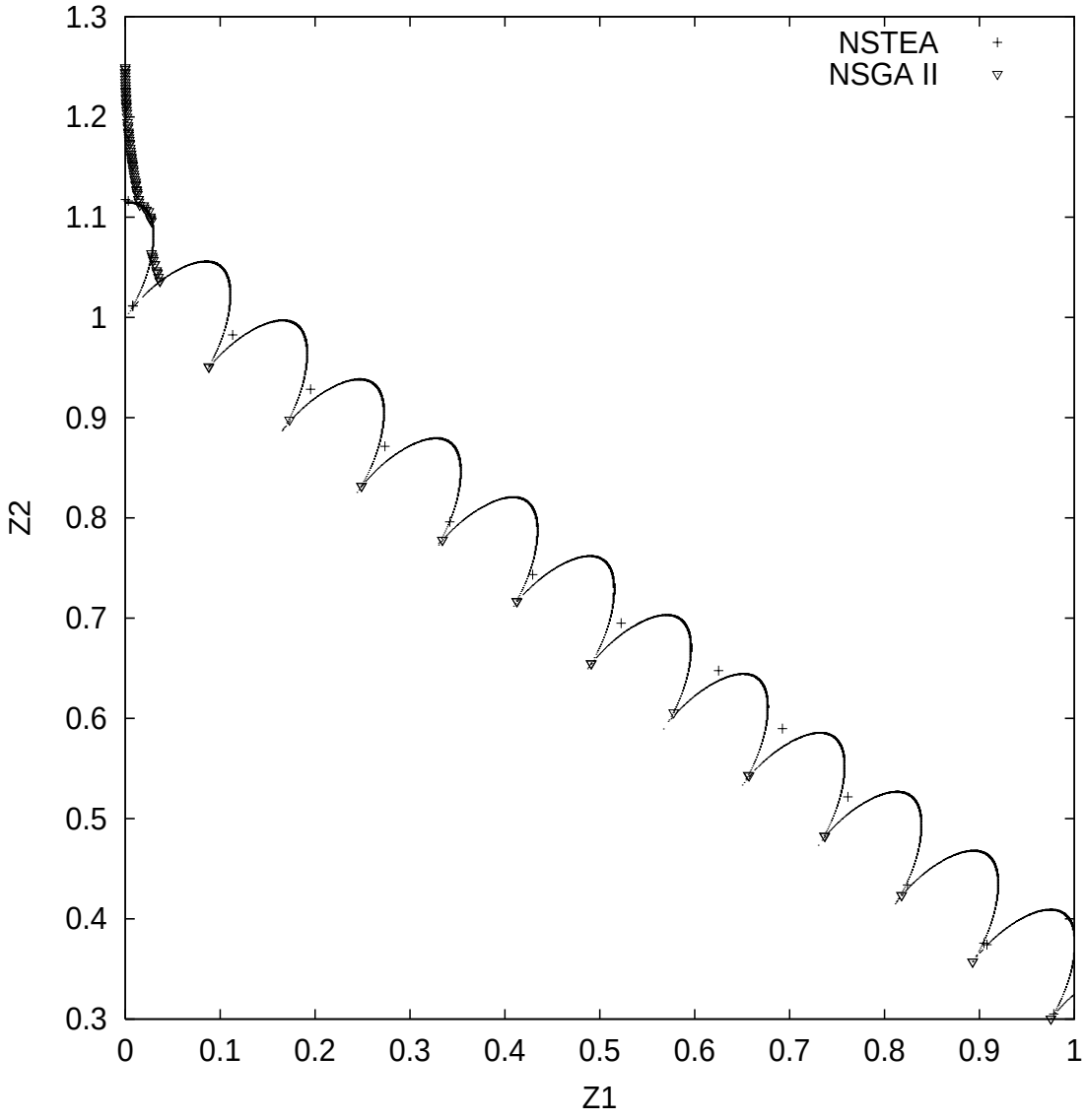


Figure 6.3: A comparison of the noninferior sets obtained using NSTEA and NSGA-II for CTP3

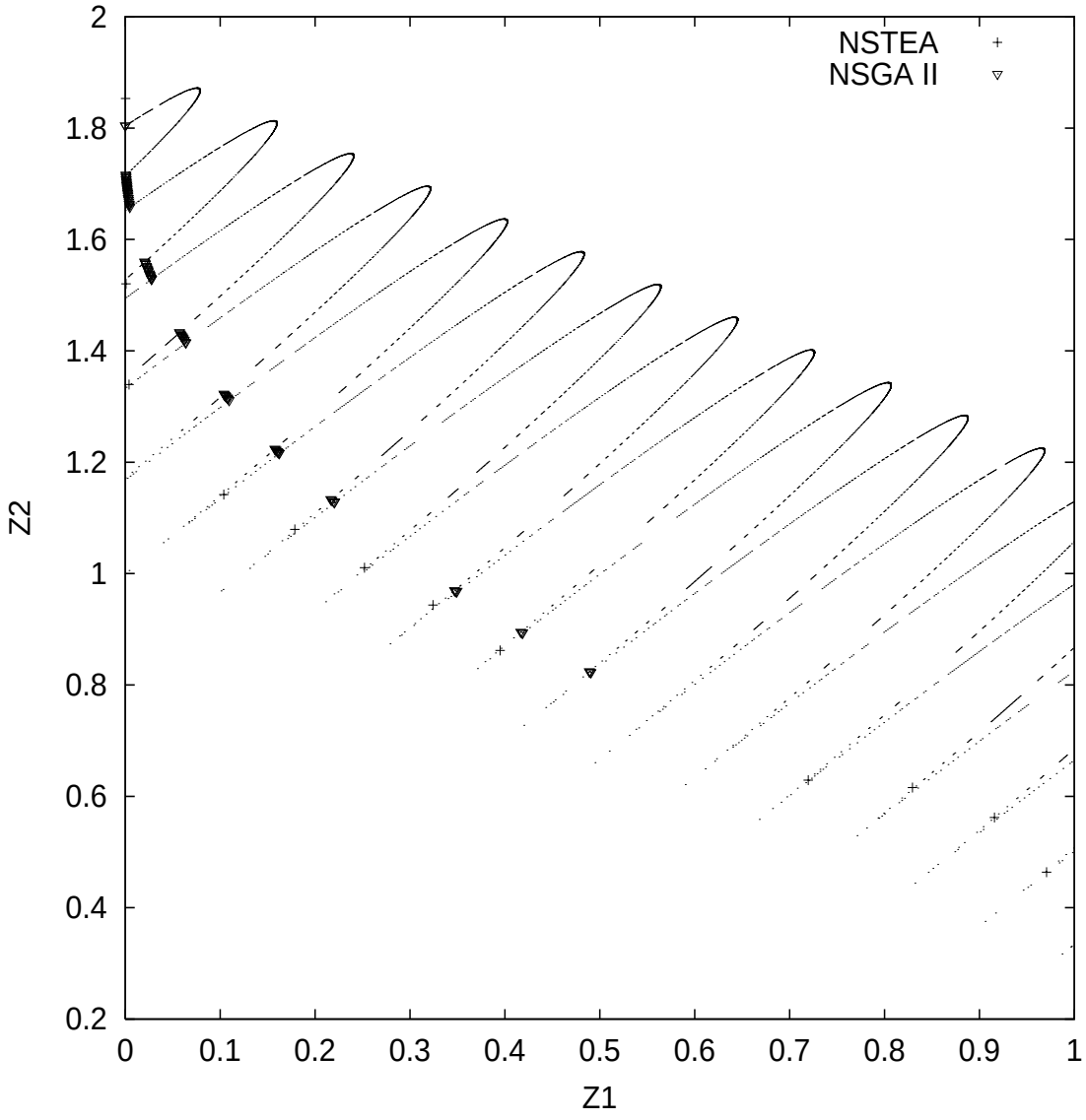


Figure 6.4: A comparison of the noninferior sets obtained using NSTEA and NSGA-II for CTP4

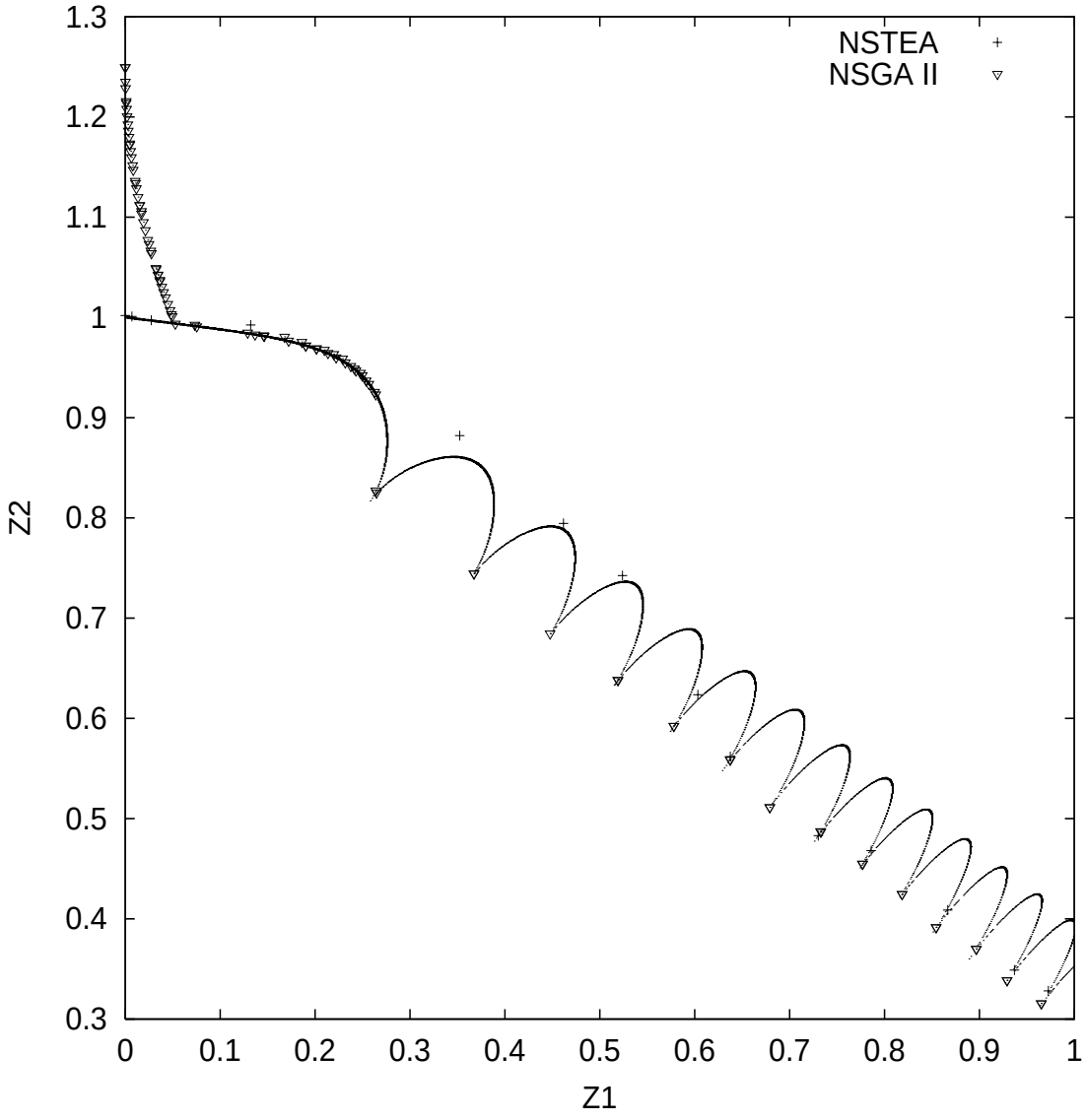


Figure 6.5: A comparison of the noninferior sets obtained using NSTEA and NSGA-II for CTP5

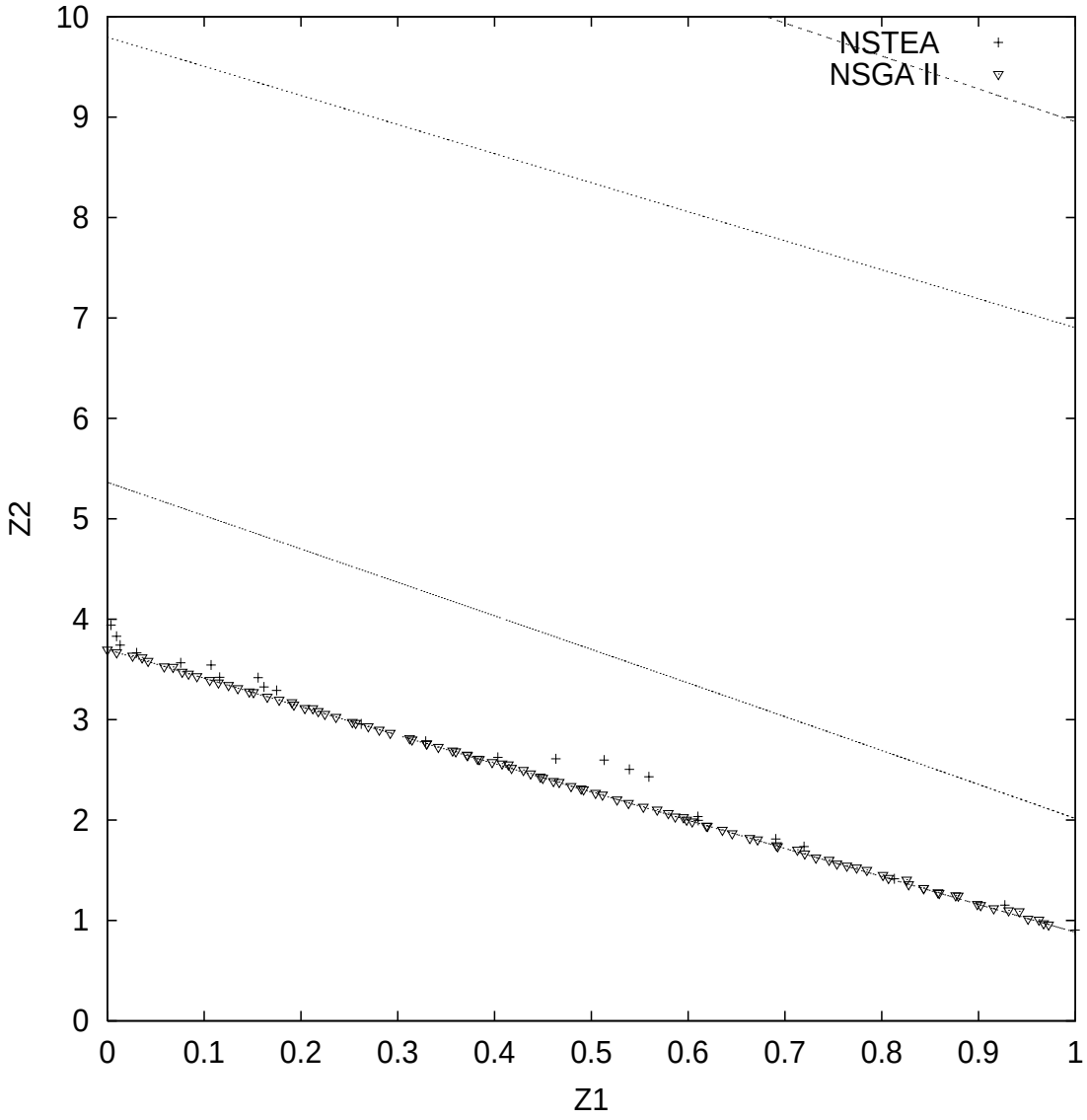


Figure 6.6: A comparison of the noninferior sets obtained using NSTEA and NSGA-II for CTP6

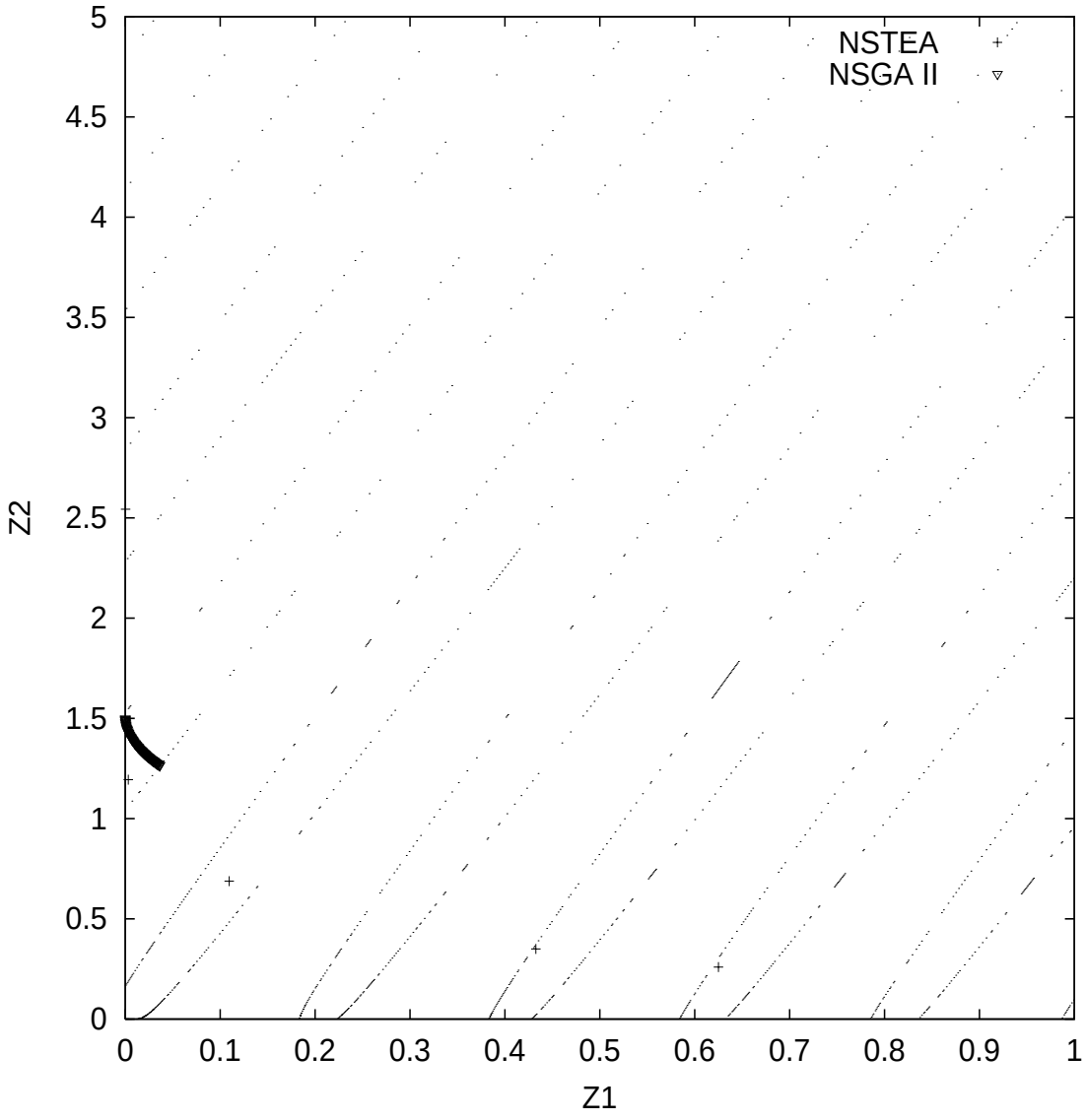


Figure 6.7: A comparison of the noninferior sets obtained using NSTEA and NSGA-II for CTP7

Table 6.2: *Accuracy* comparison, based on the S factor (Zitzler and Thiele [4], of CMEA with NSGA-II, for the test problems. A larger value indicates better performance; the best is shown in bold.

Problem instance	(S_1,S_2): (S factor for NSTEA data set, S factor for NSGA-II data set)
CTP2	(0.5617, **0.6075**)
CTP3	(0.5418, **0.5823**)
CTP4	(**0.6741**, 0.6004)
CTP5	(0.5365, **0.5923**)
CTP6	(0.5161, **0.5663**)
CTP7	(**0.6525**, 0.1543)

solutions generated by NSTEA was slightly away from the true noninferior solutions. It can be observed from Figure 6.4 that NSGA-II did not perform well for the CTP4 problem. The NSTEA, however, was able to generate solutions closer to the true noninferior set. CTP6 and CTP7 are problems where discontinuities in the search space are introduced. Further, these problems have search spaces with infeasible regions parallel and perpendicular to the search space, respectively. As can be seen from Figure 6.6, both NSGA-II and NSTEA are able to find solutions closer to the correct feasible region and closer to the true noninferior set of solutions. For CTP7, NSTEA produced a set of noninferior points more uniformly distributed in the objective space compared to NSGA-II.

6.3.1 Performance Comparison

A summary of the performance metrics for Deb's suite of multiobjective optimization problems is presented in Tables 6.2 to 6.5.

The S factor [4] values generated by NSTEA and NSGA-II are very similar (Table 6.2), with the values of NSTEA being marginally better. However, in comparing the Knowles and Corne [2] parameter for accuracy, the NSGA-II outperforms NSTEA in all problems except CTP4 and CTP7 (Table 6.3). The spread values generated by both algorithms, shown in Table 6.4, are similar for problems CTP2, CTP3, CTP5, and CTP6. For

Table 6.3: *Accuracy* comparison, based on the metric defined by Knowles and Corne [2], of NSTEA with NSGA-II, for the test problems; The best is shown in bold.

Problem instance	(P_1,P_2): (Percentage number of times NSTEA outperforms NSGA-II, Percentage number of times NSGA-II outperforms NSTEA) Number of Sampling Lines		
	108	507	1083
CTP2	(15.29, **84.71**)	(13.97, **86.04**)	(13.80, **86.20**)
CTP3	(2.83, **97.17**)	(2.24, **97.76**)	(3.80, **97.62**)
CTP4	(**100.0**, 0.0)	(**100.0**, 0.0)	(**100.0**, 0.0)
CTP5	(30.00, **70.00**)	(29.20, **70.80**)	(29.38, **70.62**)
CTP6	(40.00, **60.00**)	(44.72, **55.28**)	(45.19, **54.81**)
CTP7	(**75.00**, 25.0)	(**100.0**, 0.0)	(**100.0**, 0.0)

Table 6.4: *Spread* comparison of CMEA with NSGA-II, for the test problems. A larger value indicates better performance; the best is shown in bold.

Problem instance	($Z1$ spread for NSTEA, $Z1$ spread for NSGA-II)	($Z2$ spread for NSTEA, $Z2$ spread for NSGA-II)
CTP2	(0.9967, **1.0044**)	(1.0100, **1.3510**)
CTP3	(**0.9990**, 0.9962)	(0.8608, **1.3695**)
CTP4	(**0.9912**, 0.4971)	(1.0356, **1.4375**)
CTP5	(**1.0026**, 0.9952)	(0.7251, **1.3756**)
CTP6	(**0.9959**, 0.9744)	(0.9432, **0.9735**)
CTP7	(**0.6247**, 0.0364)	(**1.522**, 0.2337)

Table 6.5: *Coverage* comparison of CMEA with NSGA-II for the test problems. A smaller value indicates better performance; the best is shown in bold.

Problem instance	($V1$ for NSTEA, $V1$ for NSGA-II) (includes the known extreme points for each objective)	($V2$ for NSTEA, $V2$ for NSGA-II) (excludes the known extreme points for each objective)
CTP2	(**0.2678**, 0.3500)	(0.2678, **0.1030**)
CTP3	(**0.1116**, 0.3601)	(0.1349, **0.1332**)
CTP4	(**0.3743**, 1.1779)	(**0.3743**, 0.7159)
CTP5	(**0.2563**, 0.3675)	(0.2671, **0.1592**)
CTP6	(0.1405, **0.0383**)	(0.1405, **0.0355**)
CTP7	(**0.8992**, 1.5907)	(0.8992, **0.0070**)

problems CTP6 and CTP7, the spread of noninferior sets generated by NSTEA was better, indicating a better spread of solutions in the objective space. The coverage metric for the two algorithms is compared in Table 6.5. When the coverage (V2) is measured excluding the extreme noninferior points, NSGA-II outperforms NSTEA in all cases. However, when the extreme noninferior points are considered in measuring coverage (V1), NSTEA performs better in most cases. This indicates that NSTEA is able to find solutions in a broader range in the objective space.

6.4 Summary

The results presented in this chapter compares the performance of NSTEA on a set of constrained multiobjective test problems. Several random trials were performed when solving each problem. Overall, NSTEA performed well for all problems tested with respect to different quantitative measures considered. The spread and coverage of noninferior solutions obtained using NSTEA were better than those of NSGA-II and the NSTEA outperformed NSGA-II for those problems, where NSGA-II had difficulties.

The NSTEA approach has some known limitations. The computational efficiency gain obtained in NSTEA is premised on the existence of similarities in noninferior solutions that correspond to adjacent points in the objective space. For problems where this may

not hold true strongly, the search implemented by NSTEA becomes analogous to solving a number of independent single objective optimization problems, and therefore, may not realize any significant computational gain. As the underlying search mechanism for a Pareto optimal solution uses an incrementally varying aggregate function, the amount of each weight increment would dictate the number of noninferior solutions found. If this increment is relatively large, it is possible to miss some of the noninferior solutions, thus affecting the coverage. As a result, NSTEA with relatively large weight increments will likely miss noninferior solutions that lie within any linear segment of the noninferior tradeoff. For a problem with more than two objectives, incrementally updating the weight vector to obtain an adjacent point is not necessarily as straightforward as is for the two-objective cases presented here.

References

[1] S. K. Chetan. Noninferior surface tracing evolutionary algorithm (NSTEA) for multi-objective optimization. Master's thesis, North Carolina State University, Raleigh, NC., 2000.

[2] D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto-envelope based selection algorithm for multiobjective optimixation. In M. Schoenauer, Deb K., Rudolph G., X. Yao, Lutton E., J. J. Merelo, and H-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI*, Lecture Notes in Computer Science, pages 869–878. Springer Verlag, 2000.

[3] K. Deb, A. Pratap, and T. Meyarivan. Constrained test problems for multi-objective evolutionary optimization. In E. Zitzler et al., editor, *Evolutionary Multi-Criteria Optimization 2001*, Lecture Notes in Computer Science 1993, pages 284–298. Springer-Verlag, 2001.

[4] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 2(4):257–271, 1999.

Chapter 7

MGA Tool in *Vitri*

Many real world problems involve competing objectives and constraints that are difficult to quantify in a mathematical model. Further, the existence of unmodeled and unquantifiable objectives may make results of mathematically determined optimal solutions less than optimal in a practical sense. These unmodeled issues, however, can affect the eventual acceptance of a solution from a decision making perspective. Examples of such unmodeled issues can be factors such as aesthetics, socio-economic impacts, and political issues.

MGA is a technique developed to address such issues. The basic idea behind MGA is to generate a small number of different solutions when dealing with such complex, incompletely defined problems. The solutions produced by MGA are allowed to be slightly sub-optimal with respect to modeled objectives and are forced to be different from each other in decision space. MGA solutions provide a decision maker with a number of alternatives so that human judgment can be used to determine which alternative best satisfies unmodeled objectives and constraints.

This chapter presents the modules provided by *Vitri* to generate MGA solutions using specialized operators in conjunction with a GA. The implementation is based on the approach presented by Loughlin et al. [9]. The GA-based approach uses a restrictive mating scheme along with a number of specialized operators to force the evolution of a GA population to diverge into a small number of subpopulations. The approach presented in this chapter also includes a specialized operator [8] that encourages the evolution of different niches in a GA population. The implementation and the evaluation of the technique using

two problems are also described in the following sections.

7.1 Background

The applications of MGA tools to many engineering problems have been reported [1, 7]. These approaches were based on mathematical programming. However, modeling a complex system with mathematical programming tools is not always feasible. As emphasized in earlier chapters, heuristic techniques such as GAs offer effective means to solve complex problems by incorporating domain knowledge to improve the efficiency of search procedures.

A number of GA-based MGA techniques have been reported in the literature. Harrell and Ranjithan [6] presented a GA-based approach to identify different reservoir management scenarios. They followed an iterative methodology to find MGA solutions by solving different optimization problems that explicitly force the search to find different solutions.

Since a GA deals with a number of solutions simultaneously, some researchers have explored the possibility of forcing a GA to converge to multiple, different solutions. These approaches mainly included niching schemes such as sharing [5] and crowding [3]. These approaches were mainly focussed on developing techniques to handle multiobjective optimization using a GA.

In this chapter, a population-based approach that forces a GA to evolve into different, slightly sub-optimal niches is presented. The MGA technique is implemented by using an approach that is similar to the neighborhood constraint method (NCM), encouraging and maintaining diversity within a GA population by combining special operators with a restrictive mating scheme.

7.2 MGA implementation

The MGA approach is embedded in a GA by using a number of specialized operators, such as: identification of potential MGA solutions, boosting the fitness of MGA organisms, strategic placement of the MGA solutions in a population, and a neighborhood restrictive

mating scheme. These operators force the GA population to converge to distinct subpopulations and thus help in exploring different areas of decision space in search of alternative solutions. The order of execution of these operators in a GA is shown in Figure 7.1.

7.2.1 Population Indexing

The individuals in a population are given an index number from 1 to n, where n is the size of a population. The population can be considered to be a linear array of candidate solutions with an index representing the position of a particular solution. The population indexing is done as the initial population is created.

7.2.2 Neighborhood Mating Scheme

Neighborhood mating schemes restrict the mating of individuals to those within a specified "neighborhood" [9] of each other. The population is divided into m subpopulations, where m is the number of MGA solutions required. The individuals in a subpopulation mostly mate with other individuals in the same subpopulation, with mating across different subpopulations occurring only with a low frequency. This selective mating scheme encourages the GA population to converge into distinct and different subpopulations.

7.2.3 Identification of Alternative Solutions

The purpose of this step is to identify a set of m good, yet different solutions in the population. From the individuals in the population, the set of solutions that perform nearly as well as the current best solution are identified. A number of criterion can be used to identify the set of solutions. For instance, if the fitness value is used as the criteria, all the solutions having fitness values within a specified percentage (say 90%) of the best, can be selected. Other problem specific measures such as cost can also be used as the selection criteria. Once the best solutions are assembled, m solutions different from each other are selected. Phenotypic or genotypic formulations can be used to measure the difference between two solutions. The candidate solution most different from the current best is considered to be "MGA-1", the solution most different from the current best and MGA-1 is considered to

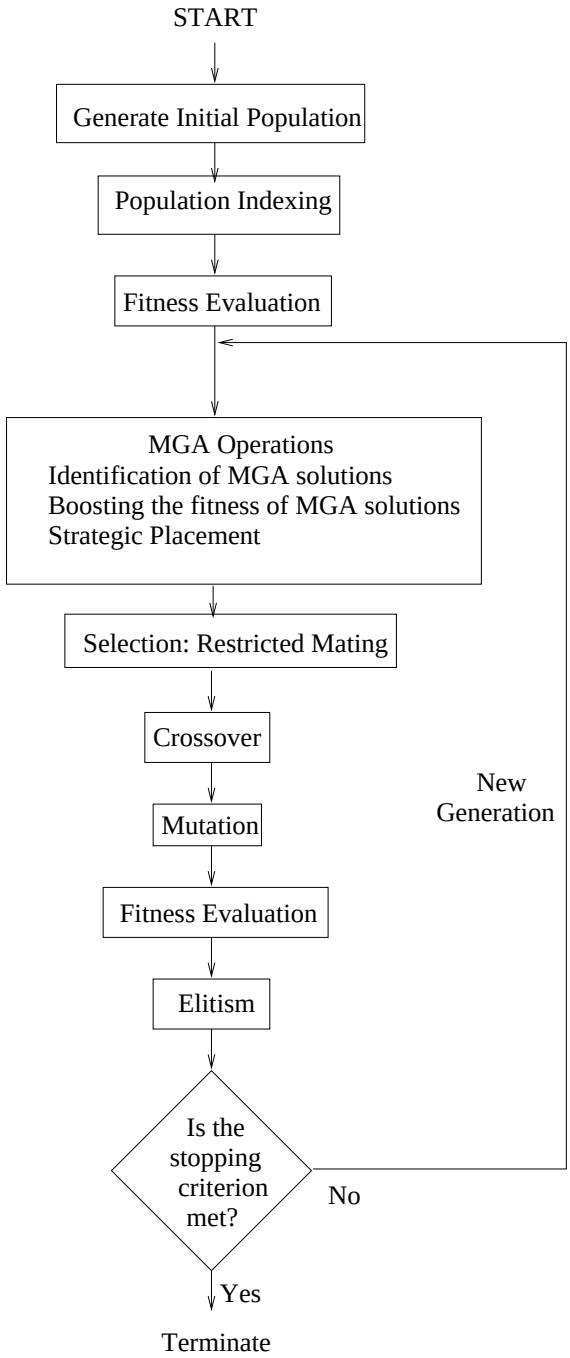


Figure 7.1: MGA operators in a GA execution

be MGA-2 and so on.

7.2.4 Boosting the Fitness of MGA solutions

Once the MGA solutions are identified, their fitnesses are increased to that of the current best solution. This operation increases the probability of MGA solutions getting selected over other solutions.

7.2.5 Strategic Placement of MGA solutions

The MGA solutions are strategically placed in different locations in the population. The locations are chosen such that each MGA solution is placed in its own "neighborhood", where it is the best solution. The placement of the solutions is done in two different ways depending on the progress of the GA.

7.2.6 Placement Method 1

Once the MGA solutions are identified, the index of the placement locations are determined as follows: The population is divided into m equal subpopulations. Let p denote the size of a subpopulation. MGA-1 is placed at location $p/2$. MGA-2 is the one which is most different from the current optimal solution and MGA-1, and it is placed at $(m-1)\times p+p/2$. In general, the MGA-i (where $i \neq 1$), is placed at $(m-i+1)\times p+p/2$ (Figure 7.2). This placement is designed so that the solutions which are most different from each other will be furthest apart.

7.2.7 Placement Method 2

This scheme is used after the subpopulations are allowed to develop and take shape. The scheme is designed so that the placement of MGA solutions will create the minimum disturbance in terms of diversity in the subpopulations. Once the MGA solutions are determined, each solution is compared with every individual in the population for difference. The cumulative difference of MGA-i with subpopulation j is termed as d_{ij}. For m MGA solutions, there are $m!$ ways of placing them in subpopulations. Each of these scenarios is enumer-

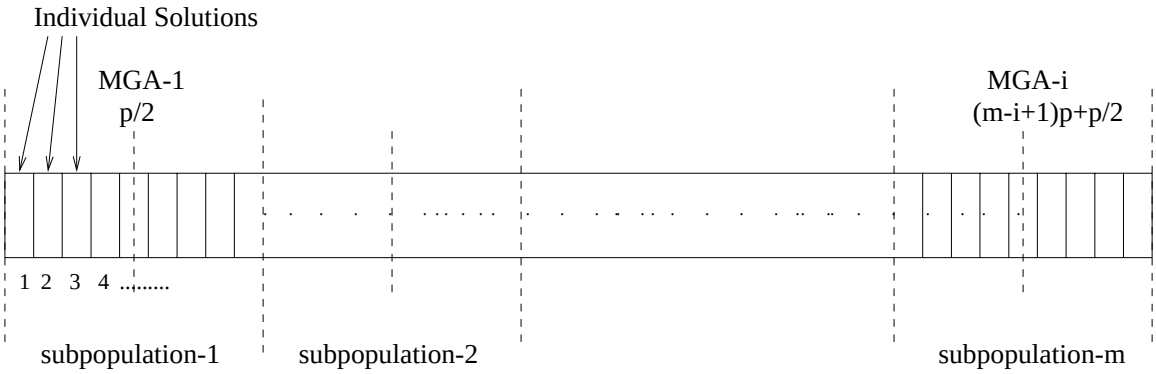


Figure 7.2: Placement scheme-1 for the MGA solutions

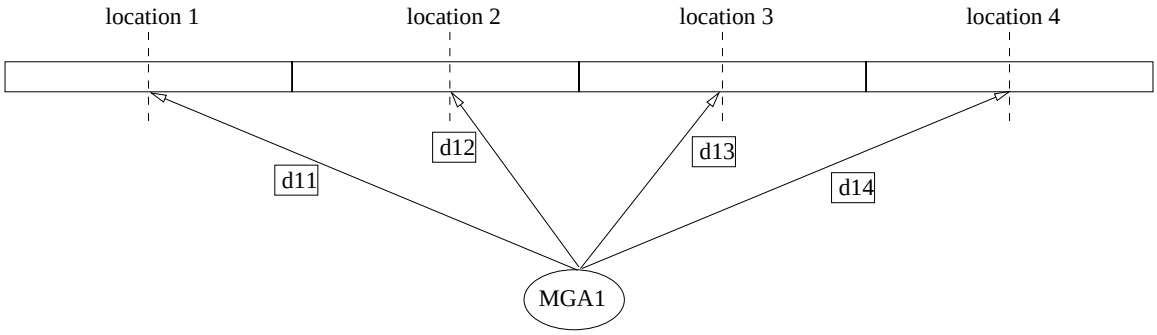


Figure 7.3: Placement scheme-2 for the MGA solutions

ated and the cumulative difference for each scenario is calculated. The scenario with the minimal cumulative difference is chosen as the placement scheme. The scheme is illustrated in Figures 7.3 and 7.4 for obtaining four MGA solutions. The population is divided into four subpopulations. The placement possibilities for four MGA solutions are emulated in Figure 7.4. For example, if the scenario with the minimal difference turns out to be $D11$, then MGA-1 is placed at location 3, MGA-2 at location 1, MGA-3 at location 4 and MGA-4 at location 2. This scheme is followed in each subsequent population.

The MGA implementation in *Vitri* GAs are applied to two different problems: (1) A simple multimodal optimization problem and (2) application to seismic performance evaluation.

1	2	3	4	d11 + d22 + d33 + d44 = D1
1	2	4	3	d11 + d22 + d43 + d34 = D2
1	3	2	4	d11 + d32 + d23 + d44 = D3
1	3	4	2	d11 + d32 + d43 + d24 = D4
1	4	2	3	d11 + d42 + d23 + d34 = D5
1	4	3	2	d11 + d42 + d33 + d24 = D6
2	1	3	4	d21 + d12 + d33 + d44 = D7
2	1	4	3	d21 + d12 + d43 + d34 = D8
2	3	1	4	d21 + d32 + d13 + d44 = D9
2	3	4	1	d31 + d32 + d43 + d14 = D10
2	4	1	3	d21 + d42 + d13 + d34 = D11
2	4	3	1	d21 + d42 + d33 + d14 = D12
3	1	2	4	d31 + d12 + d23 + d44 = D13
3	1	4	2	d31 + d12 + d43 + d24 = D14
3	2	1	4	d31 + d22 + d13 + d44 = D15
3	2	4	1	d31 + d22 + d43 + d14 = D16
3	4	1	2	d31 + d42 + d13 + d24 = D17
3	4	2	1	d31 + d42 + d23 + d14 = D18
4	1	2	3	d41 + d12 + d23 + d34 = D19
4	1	3	2	d41 + d12 + d33 + d24 = D20
4	2	1	3	d41 + d22 + d13 + d34 = D21
4	2	3	1	d41 + d22 + d33 + d14 = D22
4	3	1	2	d41 + d32 + d13 + d24 = D23
4	3	2	1	d41 + d32 + d23 + d14 = D24

Figure 7.4: Placement scheme-2 for the MGA solutions

7.3 MultiModal Problem

The MGA tools are applied to a simple multimodal optimization [9] problem, as defined below:

$$\begin{aligned} \text{Maximize } F &= 2 + sin(19\pi X) + sin(19\pi Y) + X/1.7 + Y/1.7 \\ \text{s.t.} & \\ 0.0 < X &< 1.0 \\ 0.0 < Y &< 1.0 \end{aligned}$$

The function is shown in Figure 7.5 has 100 peaks that are separated by valleys. The optimal solution is ($X = 0.974$, $Y = 0.974$), where the objective function has a value of 5.145.

The GA approach was used to identify four maximally different solutions. The MGA alternatives were restricted to being within 15% of the fitness of the best solution. The MGA solutions generated are shown in Table 7.1. The Euclidean distance between two solutions is used as the parameter to measure the genotypic difference. Figure 7.6 shows some of the MGA solutions generated during the run. It can be observed that these solutions are very different from each other and the optimum solution in the decision space, although their objective function values are within 15% of the optimum solution. Further, the peaks in Figure 7.5 to the left of the diagonal connecting the points (0.974, 0.026) and (0.026, 0.974) do not meet the 15 % relaxation constraint.

7.4 Application of MGA Tools to Seismic Performance Evaluation

Evaluation of the performance of heterogeneous structural systems during earthquakes is a complex problem due to competing design objectives, uncertainties in design data, and large simulation models. The modeling of structural performance involves analysis of subsystem

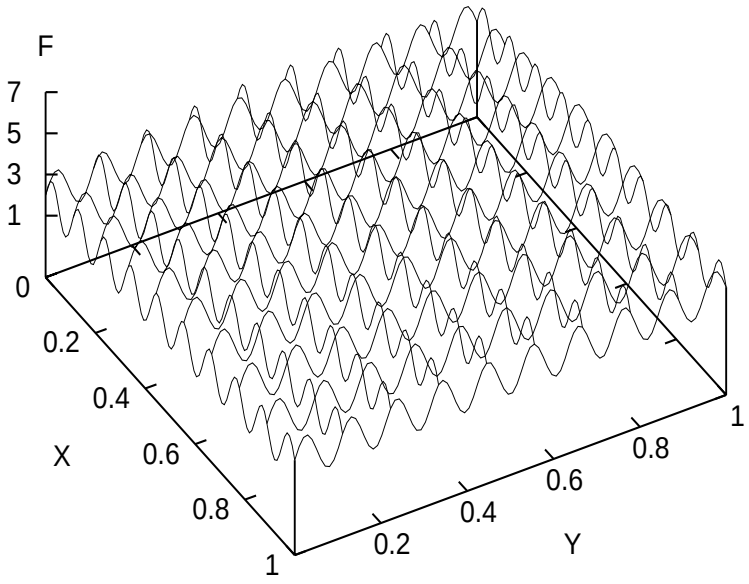


Figure 7.5: Multimodal function F

Table 7.1: MGA solutions for the multimodal problem

	Objective function value	X	Y
MGA-1	4.588	0.974	0.026
MGA-2	4.588	0.026	0.974
MGA-3	4.564	0.450	0.550
MGA-4	4.564	0.559	0.450

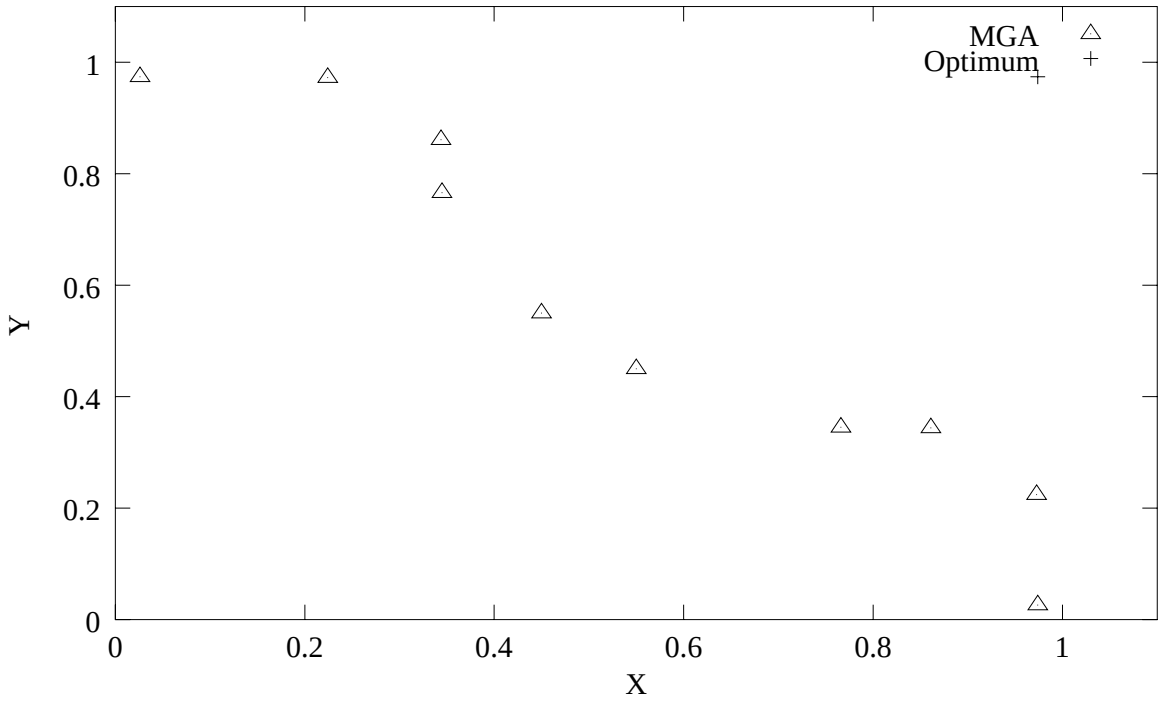


Figure 7.6: MGA solutions for the multimodal problem

interactions such as those between soil, building, equipment, and piping. Historically, due to the lack of computational resources, conservative decisions were made leading to high-cost designs. Simulation models that predict structural behavior can be used in the analysis of piping systems. However, a brute force approach in which various design scenarios are analyzed in an enumerative manner becomes computationally prohibitive. The heuristic optimization techniques in *Vitri* are used here as part of a formal procedure for generating optimal as well as alternative solutions.

The MGA tools are applied to a simple individual piping system to demonstrate the applicability of the technique. For multiply supported piping systems, optimization of support locations is a highly complex and iterative problem that can be time and cost intensive. The piping system in the present study represents a multiply supported straight pipe anchored at the two ends (Figure 7.7). The pipe is modeled using three-dimensional beam elements with one degree of freedom at each node. Several valves and other equipment located on the pipe are modeled as lumped masses. The design earthquake input is considered to be a response spectrum corresponding to the El Centro-1940 S00E record [2] uniformly applied at all support locations. The GA tools in *Vitri* are used to determine the support type and location such that the total cost of all the supports in the system is minimized. The objective function in the GA formulation incorporates a penalty for any violation of the constraints. The formulation includes constraints on displacements, stresses and excess of loads at the supports over the corresponding support capacities.

Traditionally, structural engineers spend significant effort and time to find alternative solutions in which additional supports are provided for redundancy. The localized vibrations in valves and other equipment can cause pipe failures in the vicinity of such equipment. Such failures are avoided in practice by providing supports directly under the equipment, especially the heavy ones. MGA is used to develop alternate solutions for providing insights with respect to redundancy and the support locations under heavy equipment.

7.4.1 Results and Analysis

PIPESTRESS [4], a commercial piping analysis program was used to carry out the finite element analysis of the piping system discussed above. The simulations were carried out

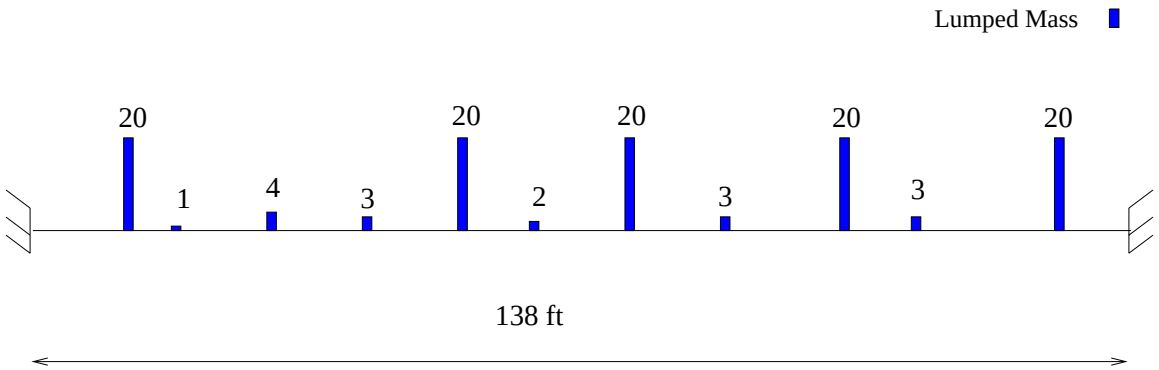


Figure 7.7: Pipe System. The values of lumped masses (in Kilo lbs) are indicated in the figure on top of corresponding location

using a network of Solaris ULTRA10 workstations.

The MGA implementation requires identification of a metric that is used to characterize the difference between alternative solutions. The following difference parameter is used to differentiate between MGA solutions

$$\delta = \sum_{i=1}^{nl} |b1_i - b2_i| \tag{7.1}$$

$$b1_i, b2_i = \begin{cases} 1 & \text{if there is a support at location } i \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

where nl is the total number of possible support locations, $b1_i$ and $b2_i$ are binary values indicating the presence of supports at location i, for solutions 1 and 2, respectively.

A GA run is performed to determine the optimal solution in terms of cost for the above piping system. A second GA run is also performed in which the placement of supports is considered to be under the lumped masses while optimizing for cost. The two solutions are shown in Figure 7.8 (SLC refers to this " support location constraint"). Table 7.2 indicates the actual capacities and costs corresponding to the indices on top of each support shown in Figure 7.8. It can be observed that the cost of the solution with all the supports under lumped masses is approximately 30% higher than the cost without that constraint.

The MGA tools are applied to generate alternatives to the cost optimal solution without the support location constraint. Some of the MGA alternatives generated are shown

Table 7.2: Capacities and Costs of Supports

Index	Capacity (lbs)	Cost (in dollars)
1	8610	21500
2	13750	26000
3	19150	24600
4	20100	24800
5	23450	32000
6	37300	35500
7	62000	36000
8	70350	42000

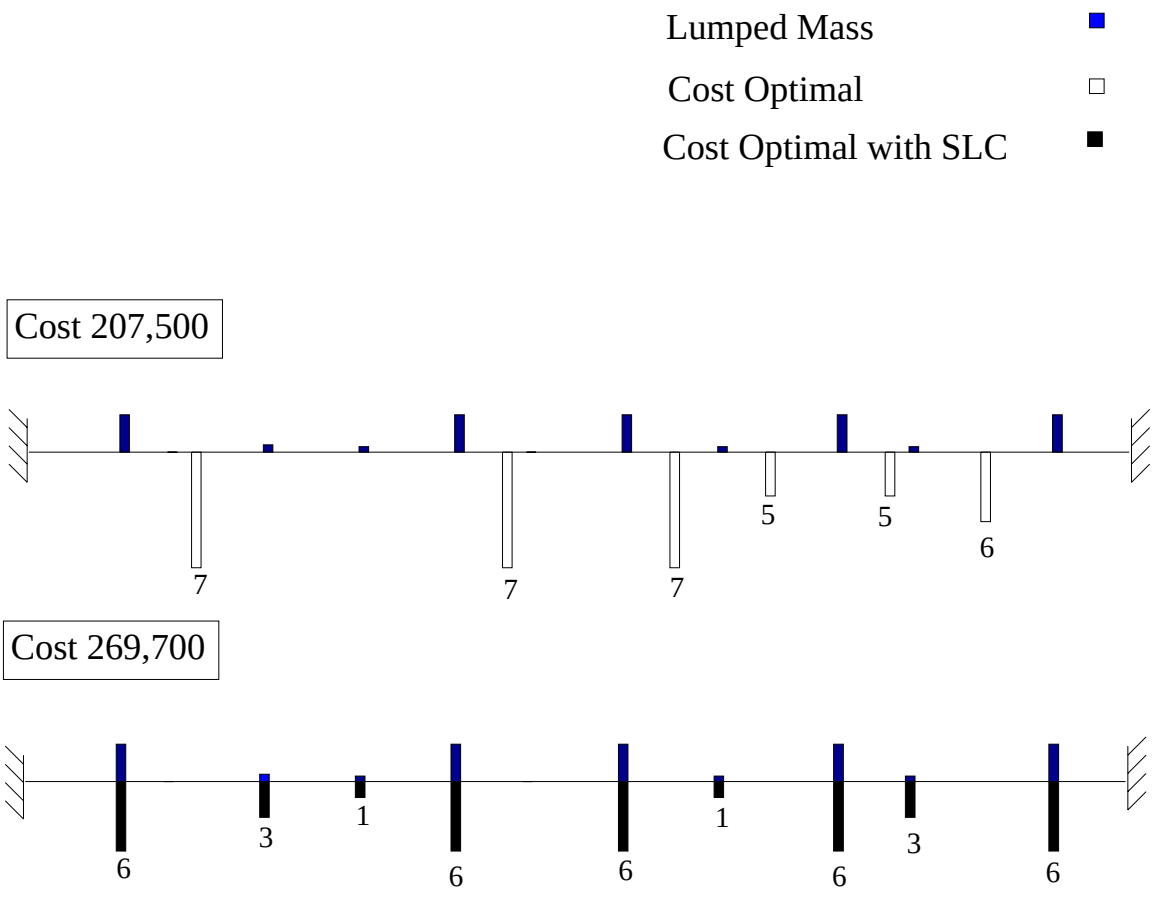


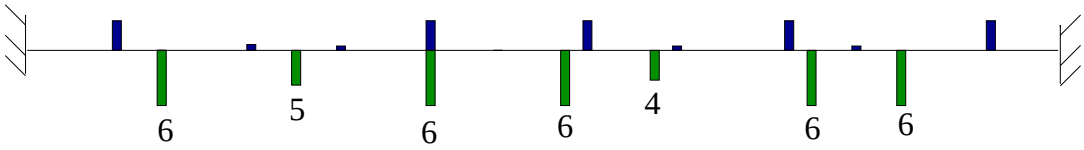
Figure 7.8: The optimum solutions

in Figure 7.9. The Figures show the location and capacities of the supports chosen, along with the value of the difference metric compared to the cost optimal solution. It can be observed that the MGA solutions are not only different from the cost optimal solution, but also different from each other. These MGA solutions provide a pool of small number of feasible solutions for the designer to work with.

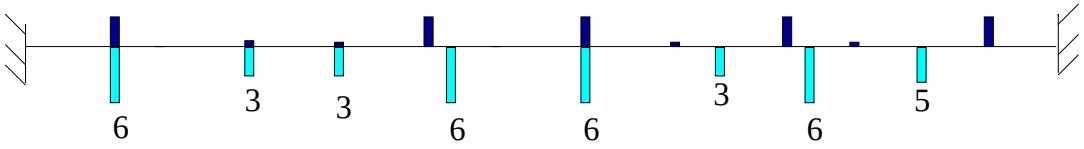
Although the fitness function includes a penalty for any violation of displacements, stresses, or capacity loads over the specified allowable limits, the degree of stressing in the piping or support loads is not explicitly modeled in the the fitness function. Figures 7.10, 7.11, and 7.12 display the ratio of support loads to capacities, the maximum displacements, and the maximum stresses, respectively, for different solutions. It can be observed that the least cost solution has high displacements and stresses compared to other solutions. The MGA-3 solution also displays high stresses at one end. From the Figure 7.10 it can be seen that all the solutions contain at least one support that has a ratio of support -load-to-capacity ratio close to 1. These ratios are low for most supports in MGA-4, followed by MGA-2. A designer may prefer a solution with somewhat higher cost (244,800, or 247,800), with relatively lower values of displacements, stresses, and support loads because of lower chances of exceeding the allowable values due to uncertainty or loads greater than the design loads.

It can be observed that the parameters such as the displacement, stresses, and support capacity ratios can be explicitly modeled to obtain the corresponding optimal solutions. However, the MGA solutions presented in Figure 7.9 can still be used to provide insights into the trends of various design parameters. A designer can also use these solutions as starting points for further exploration.

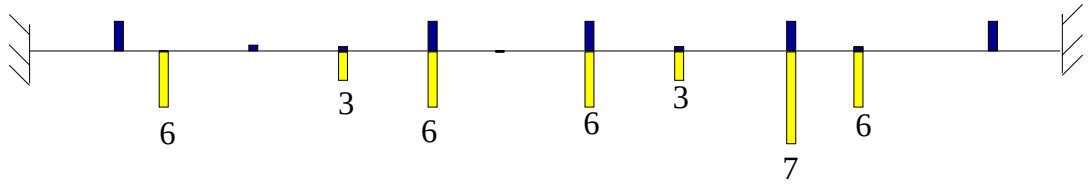
One unmodeled issue in these MGA solutions is the aesthetics. The location of supports under lumped masses is preferred in practice due to its increased reliability against observed failures. It can be seen from the Figure 7.9 that the MGA solutions performs better than the cost optimal solution with respect to the placement of supports under lumped masses. MGA-2 and MGA-3 solutions have 4 and 6 supports under lumped masses, respectively. Further, the MGA solutions are more symmetrical in their placement of supports and their capacities.



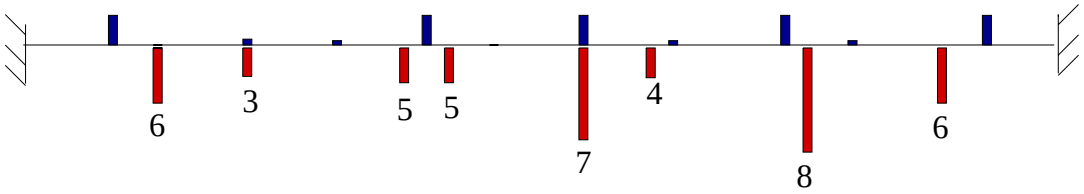
MGA-1 Cost 234,300, $\delta = 13$



MGA-2 Cost 247,800 $\delta = 10$



MGA-3 Cost 227,200, $\delta = 13$



MGA-4 Cost 244,800 $\delta = 14$

Figure 7.9: MGA Solutions for the Pipe System

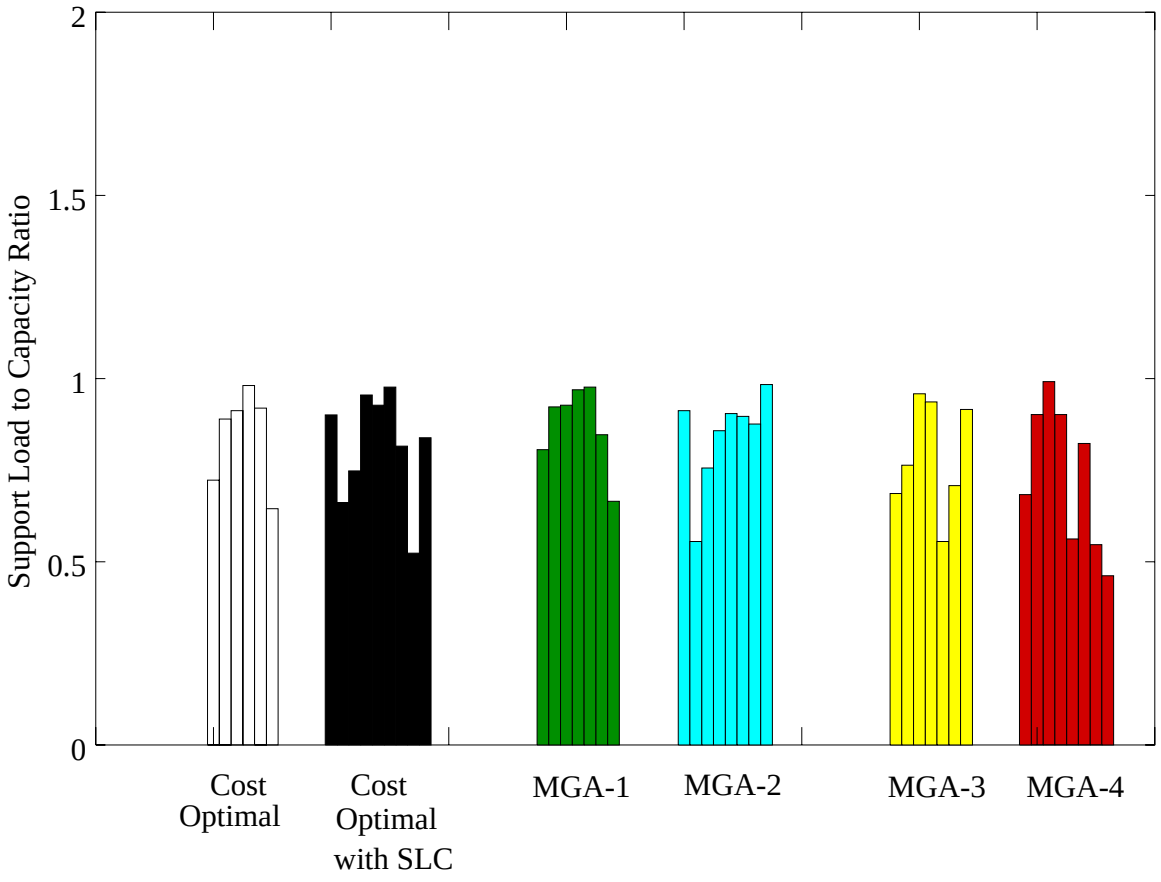


Figure 7.10: Ratio of Support Loads to Capacities for Solutions to the Piping Problem

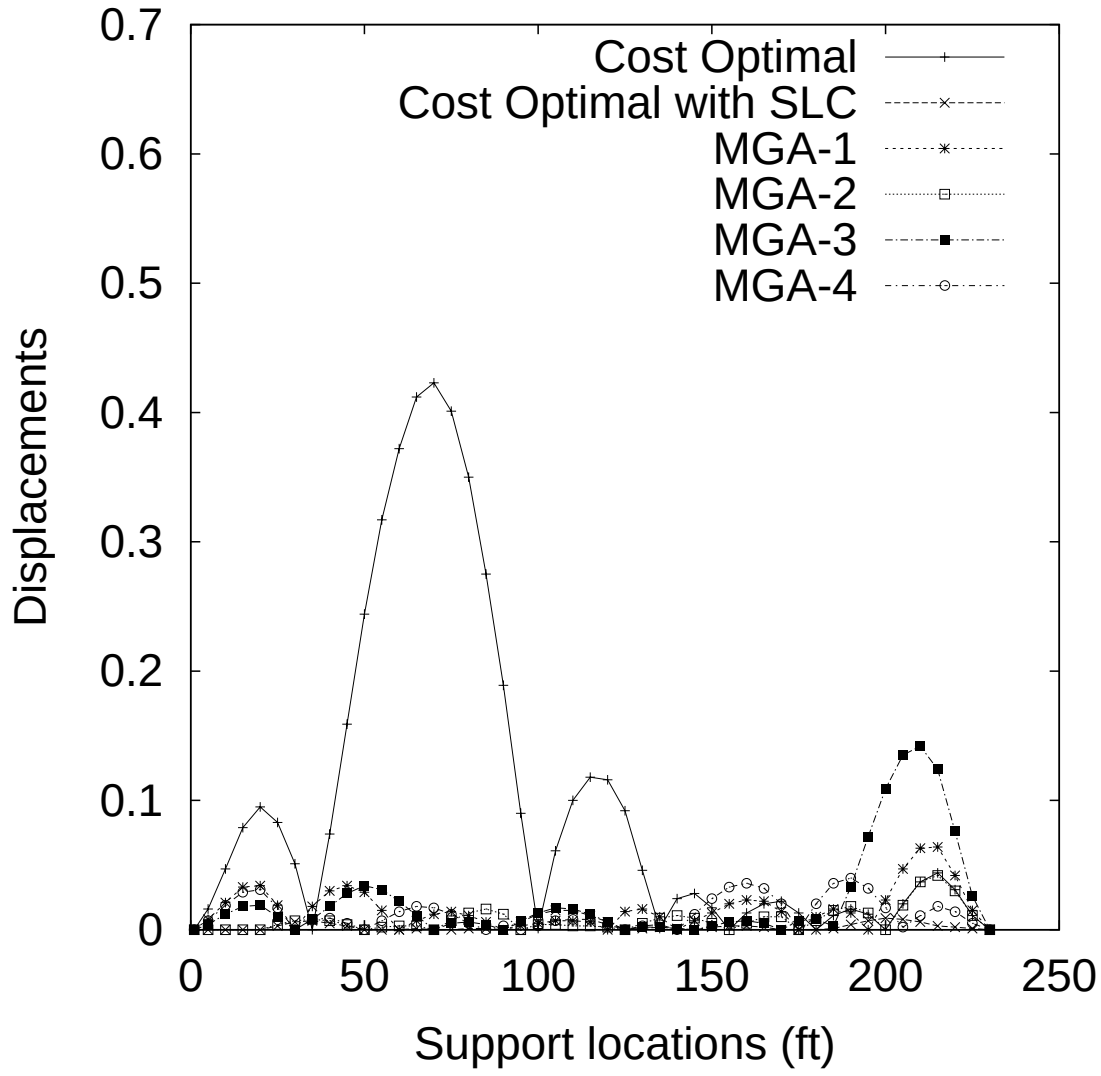


Figure 7.11: Displacements at Each Support for the Solutions to the Piping Problem

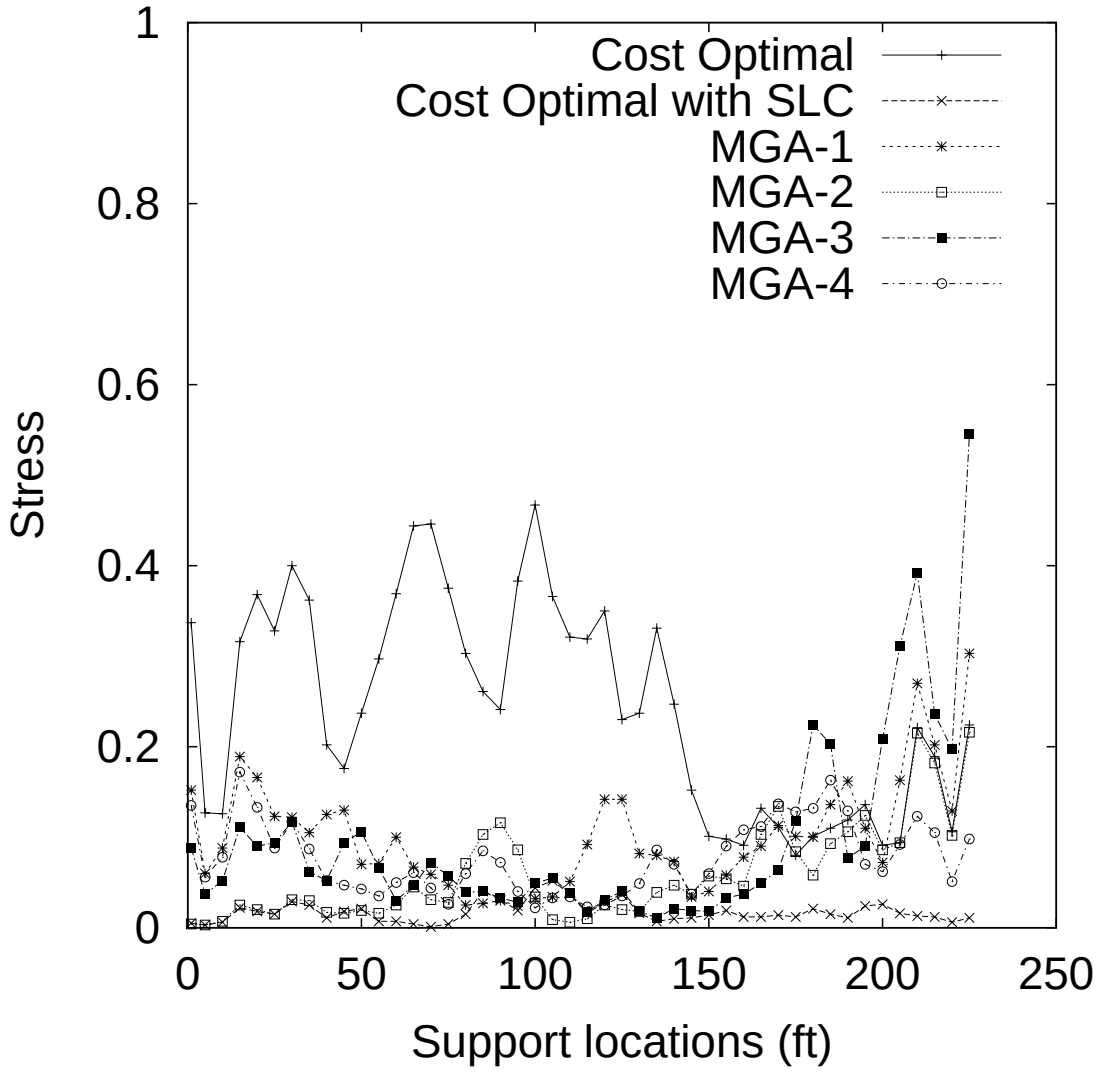


Figure 7.12: Stresses at Each Support for the Solutions to the Piping Problem

7.5 Summary

The MGA technique in *Vitri* is currently implemented only within the SDGA. However, the method can also be implemented within the ADGA by implementing the MGA operations as separate nodes and incorporating it within the existing dataflow framework.

The results illustrate the usefulness of MGA tools in *Vitri* for generating solutions that are good with respect to modeled objectives, and yet different in decision space for complex problems with unmodeled issues. Other measures to determine the difference between two solutions can also be used to generate solutions. This approach illustrates the utility of MGA in helping a decision maker use experience and professional judgment to choose among from a number of alternatives.

References

[1] J. W. Baugh, S. C. Caldwell, and E. D. Brill. A mathematical programming approach for generating alternatives in discrete structural optimization. *Engineering Optimization*, 28:1–31, 1997.

[2] A. K. Chopra. *Dynamics of Structures, Theory and Applications to Earthquake Engineering*. Prentice Hall, New Jersey, 1995.

[3] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[4] DST Computing Services, S.A. *PipeStress Users's Manual Version 3.4.09*.

[5] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 4–49, 1987.

[6] L. J. Harrell and S. R. Ranjithan. Generating efficient watershed management strategies using a genetic algorithm-based method. In *Proceedings of the 24th Annual Water Resources Planning and Management Conference*, pages 272–277, Houston, TX., 1997.

[7] L. D. Hopkins, E. D. Brill, and B. Wong. Generating alternative solutions for dynamic programming models of water resources problems. *Water Resources Research*, 18(4):782–790, 1982.

[8] D. H. Loughlin. Personal Communication, 2001.

[9] D. H. Loughlin, S. Ranjithan, E. D Brill, and J. W. Baugh. Genetic algorithm approach

for addressing unmodeled objectives in optimization problems. *Engineering Optimization*, 33:549–569.

Chapter 8

Conclusions and Future Directions

This thesis describes the design and application of a computational framework that supports the engineering design and analysis process. The design of *Vitri* using object-oriented principles enables the reuse of components that can be customized to meet new application requirements as demonstrated by its use on a wide variety of applications. *Vitri* integrates a number of tools within a framework that supports a set of applications in diverse areas, from different domains, and with different design requirements. *Vitri* provides a platform for computational prototyping and experimentation with new approaches to scientific and engineering design. The modularity, adaptability, and scalability of the framework have been used effectively in the development of solution methods and in the transparent use of underlying software and hardware.

The software architecture of *Vitri* consists of a number of basic components that facilitate the use of a distributed system and a number of components loosely built on top it. These loosely built components are various algorithmic tools such as GAs, multiobjective GAs, MGA tools, etc. These components share and reuse generic abstractions in *Vitri*. The framework structure allows the components to be dynamically linked at runtime so that the user is not bogged down by the size of a large number of components.

The modules in *Vitri* are based on existing, well-established algorithms. However, the traditional methods and algorithms have been explored to remove their limitations. New approaches to improve their algorithmic efficiencies have also been studied. For instance, as described earlier, an asynchronous distributed GA approach that eliminates the end-

of-generation bottlenecks associated with a synchronous GA has been developed. The improvements in GA-based MGA techniques have been explored by implementing new schemes in a GA. The multiobjective optimization algorithms have also been evaluated by testing them using a number of problems from literature.

In addition to improvements in distributed and optimization routines, application of tools in *Vitri* to various problems have led to the development of new design and analysis approaches. The use of *Vitri* has enabled the development of a new approach for the least cost design of water distribution networks with redundancy constraints. The approach is computationally intensive, but has been effectively handled using the tools in *Vitri*. The approach also provides a framework for incorporating other design measures such as water quality and more complex variations in the analysis. *Vitri* has also been used to experiment and learn about a problem incrementally. For instance, tools in *Vitri* were used to investigate the potential tradeoffs between the total number of piping supports and the cost of the system for the seismic performance evaluation problem. The maintenance or lifetime cost of the system is proportional to the number of supports, whereas the installation costs of the system is assumed to be dependent on the capacities of the supports. The knowledge on lifetime cost was extremely difficult to obtain and communication with practicing engineers revealed that it was highly site specific. The initial optimization studies revealed no multiobjective relationships between the hardware cost and the number of supports. When the expected lifetime cost was approximated as power functions of the initial hardware cost, the relationship between cost and the number of supports was found to be competing, resulting in a multi-objective scenario. This analysis was conducted for a small system, and the increased knowledge from this study can be used in analyzing more complex systems. Though the analyses were computationally intensive, the use of *Vitri*'s distributed computing environment helped in solving problems in a reasonable amount of time. The MGA tool, as mentioned in Chapter 7 is also used as part of the study to develop and evaluate design scenarios for the seismic performance evaluation problem.

Some of the limitations of *Vitri* can be identified as follows: *Vitri* has been designed mainly for perfectly parallel applications that have a set of tasks that require little or no communication. *Vitri* is also suitable for data parallel applications that have the same

operations performed on many data elements simultaneously. There is another class of parallelism called control parallelism that have different operations performed simultaneously on different processors. Control parallel applications would require the allocation of specific tasks to specific servers and usually are done on a dedicated network of workstations. The pool-of-tasks structure in *Vitri* would require modifications to incorporate control parallel applications effectively. The algorithms in *Vitri* are more suitable for coarse-grained problems with low communication costs compared to the computational costs. The performance of the algorithms when applied to fine grained problems could suffer because of the communication overhead. Analytical studies such as the one presented in Chapter 3 can be used to determine the suitability of using the distributed algorithms in *Vitri*.

Vitri has been used for rapid prototyping during various stages of different applications discussed in this thesis. The air quality optimization problem was initially studied with a simple simulation model and later with a more complex model. The design of water distribution system with redundancy considerations was initially studied for a simple system. Simulations using the simple system was used to understand the trends in tradeoffs between the objectives of interest. As mentioned above, the multiobjective optimization and MGA tools were used to incrementally learn about the seismic performance evaluation problem. The study is expected to be extended for more complex design scenarios. In conclusion, the flexible features in *Vitri* has allowed the plug-and-play of new models, design prototypes, and the use of different techniques.

Appendix A

Sample Execution of *Vitri*

The design of *Vitri* and its tools, along with the hot spots that enable the plug and play of various applications were discussed in earlier chapters. A prototype execution of *Vitri* environment and its tools are explained below.

It is assumed that a Java virtual machine environment (version 1.2 or above) is already installed. Before running *Vitri*, the `CLASSPATH` variable needs to point to the *Vitri* classes and then to JDK. The execution of *Vitri* can be done both from the command line and by using a graphical interface. The command line execution is done as follows.

The client side execution of *Vitri* is done by instantiating the appropriate program at the runtime and also by specifying if the execution is distributed or not.

```
>java Client -prgm [main program] -restart [true/false]
-distributed [true/false] -p [port]
```

The flags `-prgm`, `-restart`, `-distributed`, and `-p` refer to the main program that is being executed, if the program is restarted from a previously saved state, if the distributed environment is used, and the port number, respectively. For instance, if the SDGA is being run, the main program can be specified as `vitri.ga.SyncGeneticAlgorithm`. The execution of a server, however, is independent of the problem. A server can be instantiated as follows.

```
>java Server -client [client name] -p [port]
```

The flag `-client` refer to the client from which the server requests tasks, and `-p` refers to the port number where the connection to the client is made.

The distributed GA implementations also allow the users to specify the GA parameters at runtime. A file called `ga_info.txt` is read by the distributed GAs at run time. A sample file is shown below.

```
#Name of the organism class used (fully qualified name)
Organism class  : vitri.ga.KnapsackOrganism
#Name of the fitness class used (fully qualified  name)
Fitness class    : vitri.ga.KnapsackFitness
#Number of Generations
Number of generations  : 300
#Size of a single population.
Population size  : 100
#
Crossover percentage [0-1] : 0.95
#
Mutation rate  [0-1] : 0.005
#
Elitism level [0-2]  : 1
#
Print level[0-2] : 1
#This feature must be enabled if you want to restart from a certain stage.
Save population [true/false] : false
Save to directory  : .
Graphics [true-false] : true
Mga solutions [true/false] : false
#If mga solns are needed
Number of mga solutions : 4
Niching percentage [0-1]  : 0.85
```

The organism and fitness classes, the number of generations and population size, the crossover and mutation rates, and the level of elitism can be specified through this file. If the distributed GA needs to be restarted, the save feature must be enabled that saves the population in each generation to the specified location. The graphics flag, if enabled, allows a dynamic display of the maximum and average fitness values of each generation (Figure A.1). If the MGA solutions are required for the GA execution, the appropriate flags in the file can be enabled. The niching percentage specifies the limit for the selection of MGA solutions. i.e., the solutions within the niching percentage of the best is chosen, and the most different solutions among them are chosen as the MGA solutions.

Similar to ADGA and SDGA, the multiobjective GA tools also allow users to

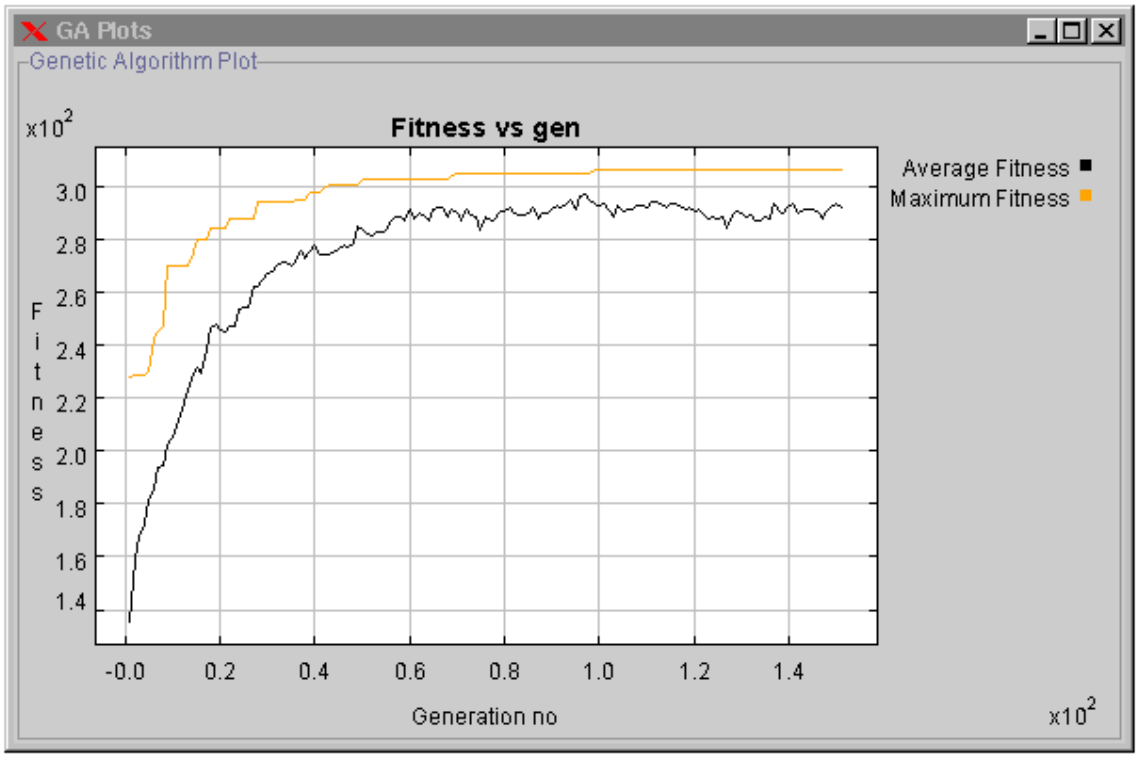


Figure A.1: Dynamic display of maximum and average fitness values for a GA execution.

specify parameters at runtime. The multiobjective GAs read the parameters from a file called `moga_info.txt`. A sample file is shown below. The file specifies the number of objectives, and the range and number of points of the objectives being constrained. The choice of the type of GA (ADGA/SDGA) for the inner loop of the multiobjective GAs can also be specified through the file.

```
Number of objectives    : 2
#specify the range and number of points of the objective (for CMEA)
z2  : 1000, 25
ga type (ADGA/SDGA)  : SDGA
```

Appendix B

Monitoring Tools

Vitri provides a graphical user interface allowing users to select simulation variables interactively, and to monitor the progress of the distributed system dynamically. The tool allows the the user in choosing the parameters required in the execution of *Vitri* as shown in Figure B.1. Snapshots of the system are shown in Figures B.2 to B.4. The interface provides a graphical representation of the pool of tasks, the status of different servers, the memory usage, load status of the client, etc.

Vitri also provides and interface (Figure B.4) that displays a dynamic Gantt chart of servers indicating how much time each server spends in communicating, idling, and executing tasks. The activity display helps in visualizing measures such as the relative communication costs of servers, the relative speeds of different servers, etc.

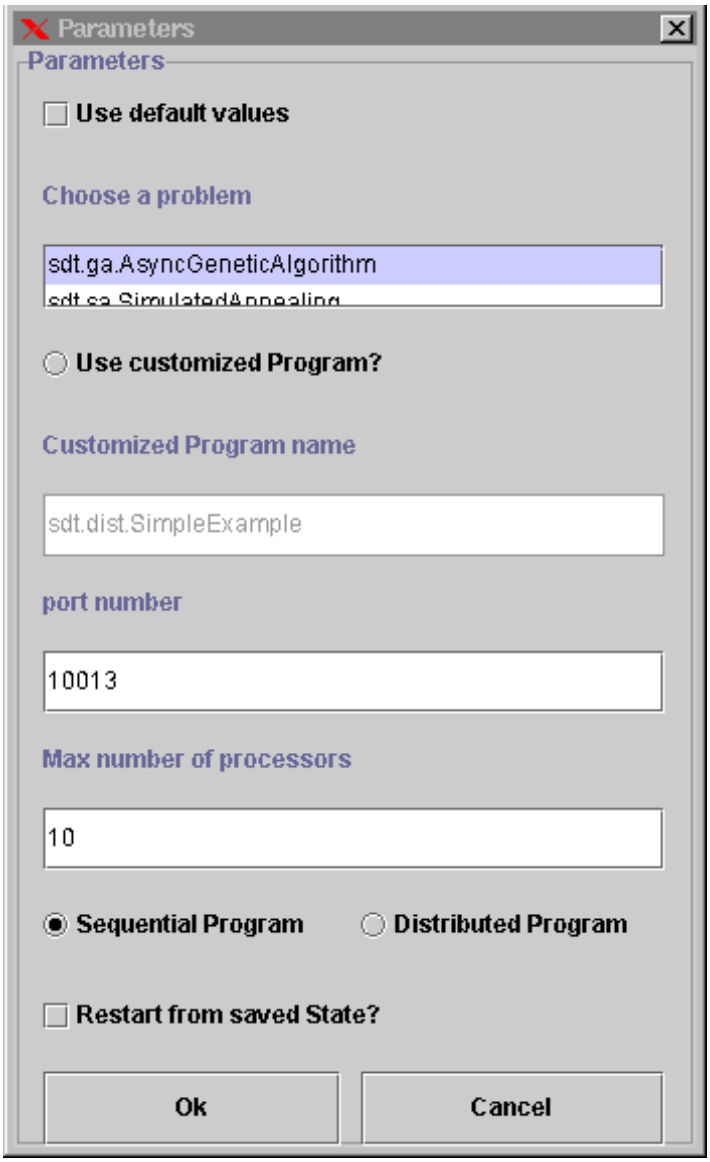


Figure B.1: Interface to choose simulation variables in *Vitri*

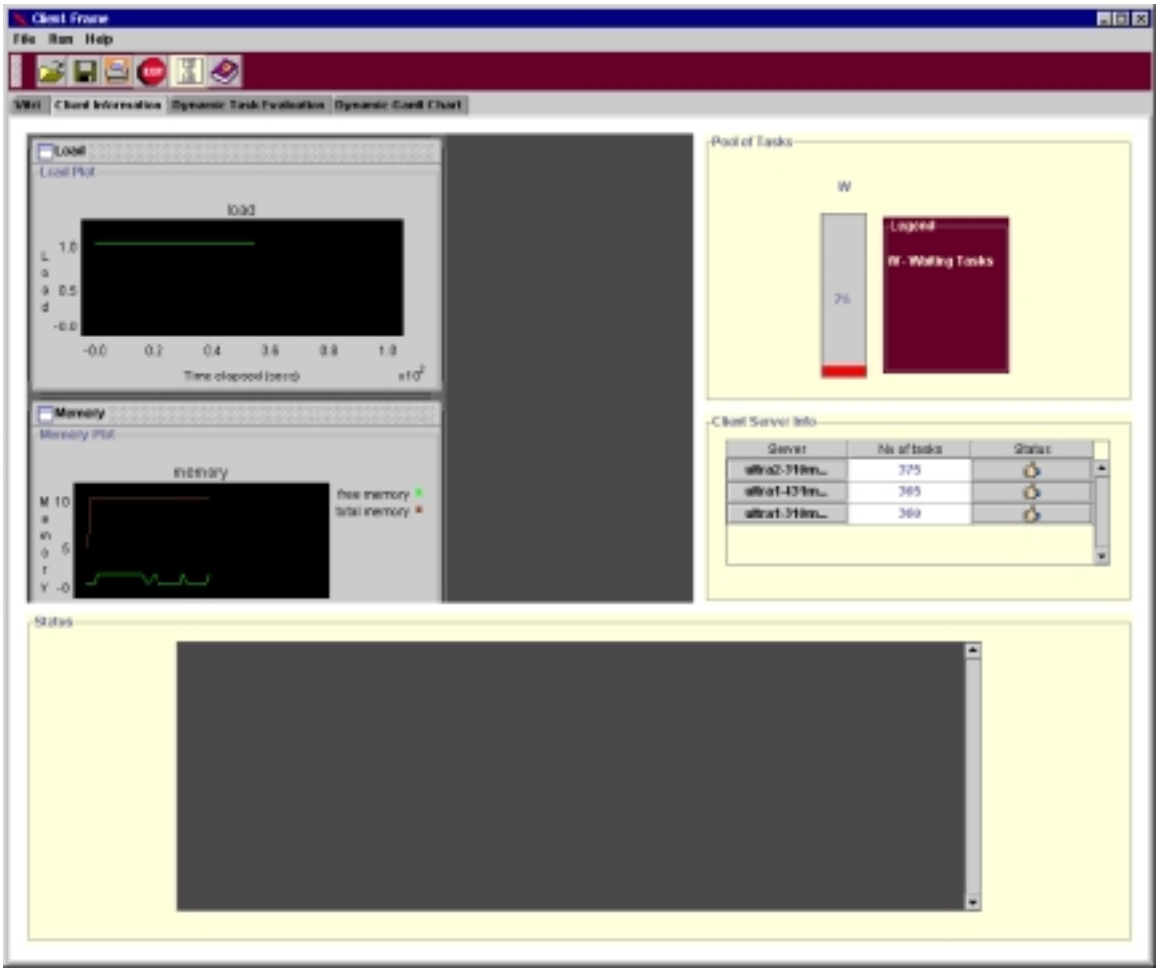


Figure B.2: Snapshot of the Vitri's interface

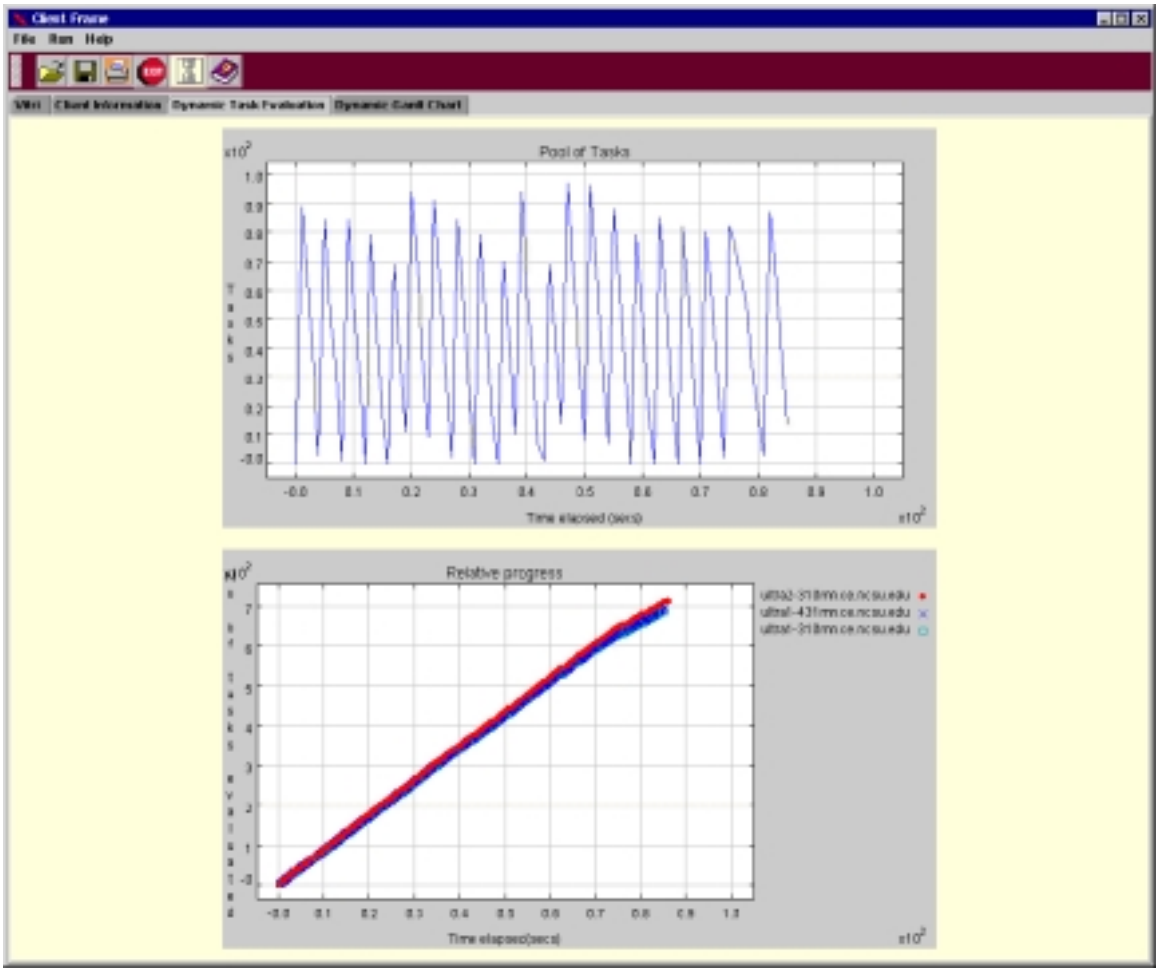


Figure B.3: Snapshot of the Vitri's interface

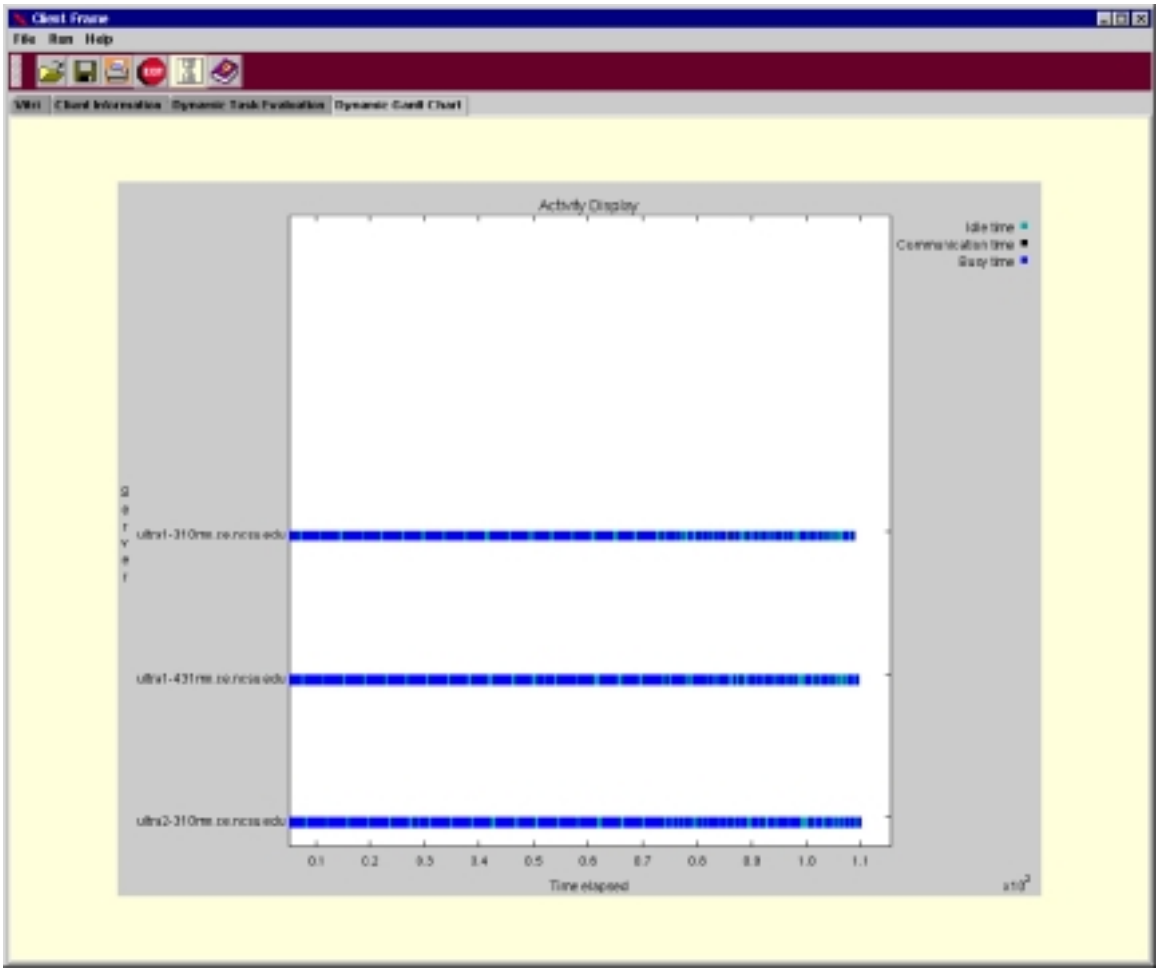


Figure B.4: Snapshot of the Vitri's interface

Appendix C

Summary of *Vitri* Applied to Various Problems

The table C.1 lists a summary of the use of *Vitri* to various problems. Parameters such as the platform, the computational intensity, problem specific parameters, the number of machines employed, the simulation model used, etc., are listed in the table.

Table C.1: Summary of different problems solved using *Vitri* (SU- Sun ULTRA10, PW - Pentium III Windows, PL - PIII Linux

Problem	Simulation Model	Runtime	Platform	GA parameters				
				GA type	Pop.size	Number of Generations	Total time	Number of Machines
Air Quality Optimization								
ELC	–	< 1 second	SU	Seq	100	300	25 minutes	1
ALC	EKMA	2 seconds	SU	SDGA	100	300	6.2 hours	3-5
ALC	UAM	20 minutes	SU	SDGA	100	90	1.735 months	3
ALC	UAM	20 minutes	SU	ADGA	100	90	1.551 months	3
Water Distribution System Design								
Toy Problem Level 0	EPANET	72 milliseconds	SU	Seq	100	200	9 minutes	3
Toy Problem Level 1	EPANET	72 milliseconds	SU	Seq	100	200	1.8 hours	3
Toy Problem Level 2	EPANET	72 milliseconds	SU	Seq	100	200	2.7 hours	3
Hanoi Problem Level 0	EPANET	80 milliseconds	PL	SDGA	100	500	27.3 minutes	3
Hanoi Problem Level 0	EPANET	80 milliseconds	PL	ADGA	100	500	25.07 minutes	3
Hanoi Problem Level 1	EPANET	80 milliseconds	PL	SDGA	100	500	14.98 hours	3
Hanoi Problem Level 1	EPANET	80 milliseconds	PL	ADGA	100	500	12.55 hours	3
Hanoi Problem Level 2	EPANET	80 milliseconds	PL	SDGA	100	500	5.91 days	3
Hanoi Problem Level 2	EPANET	80 milliseconds	PL	ADGA	100	500	5.28 days	3
Sioux Falls Problem Level 0	EPANET	150 milliseconds	SU, PL, PW	ADGA	100	500	0.21 hours	5-25
Sioux Falls Problem Level 1	EPANET	150 milliseconds	SU, PL, PW	ADGA	100	500	2.86 days	5-25
Sioux Falls Problem Level 2	EPANET	150 milliseconds	SU, PL, PW	ADGA	100	200	1.82 months	5-25
Truss Design								
Truss Design (Ten bar truss)	SAP	3.5 seconds	PW	Seq	100	200	19 hours	1
Seismic Response Analysis								
Pipe Support Design	PIPESTRESS	4.5 seconds	SU	SDGA (MGA)	100	300	3.75 hours	3-20
Knapsack Problem (simulated)								
Knapsack	–	0.5 seconds	SU	SDGA	100	100	10.42 minutes	30
Knapsack	–	0.5 seconds	SU	ADGA	100	100	8.72 minutes	30
Multiobjective GA runs								
Pipe Support Design	PIPESTRESS	4.5 seconds	SU	SDGA	100	100	37 hours	3-15
Vehicle Routing								
Routing			PW	SDGA				10