Optimized Software Component Allocation
On Clustered Application Servers

by
Hsiauh-Tsyr Clara Chang, B.B., M.S., M.S.

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

School of Computer Science and Information Systems

Pace University

March 2004

We hereby certify that this dissertation, submitted by Hsiauh-Tsyr Clara Chang, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.

____-_________________

Lixin Tao Date

Chairperson of Dissertation Committee

____-_________________

Fred Grossman Date

Dissertation Committee Member

____-_________________

Michael Gargano Date

Dissertation Committee Member

School of Computer Science and Information Systems

Pace University 2004

Abstract

**Optimized Software Component Allocation
On Clustered Application Servers**

by
Hsiauh-Tsyr Clara Chang, B.B., M.S., M.S.

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

March 2004

In the last decade, online e-commerce businesses, represented by the e-commerce portals, have grown significantly and become an important sector of world economy. This dissertation helps address the server scalability problem for supporting the sustainable growth of the online e-commerce industries.

Most of today's e-commerce portals are implemented with distributed component technologies and server clusters. Each server application comprises dozens or hundreds of distributed software components, and each of such components can run on any of a cluster of application servers connected by a high-speed fiber local area network (LAN). While multiple server machines support parallel execution of the software components, inter-server communication is a few orders slower than servers' CPU speed. This research studies the optimized allocation of software components to server machines to maximize computation load balance and minimize communication overhead.

Multi-way graph partitioning is first adopted to model the software component allocation problem. The problem is proved to be NP-hard. A novel graph transformation is introduced to combine the two conflicting objectives into a single objective function, and a transformation theorem is proved that problem instances before and after this transformation are equivalent. Based on careful observation of the properties of the solution space, a scheme for incremental objective function evaluation is designed to speed up any iterative solution heuristics to this problem by a factor proportional to the number of software components involved. Simulated annealing is adopted to solve the problem. Extensive experimental study shows that the proposed simulated annealing algorithm can outperform repeated random running in the same amount of time by 16.67% to 100%, and outperform local optimization by 1.92% to 100% with a running time about 6 to 100 times of that for the latter.

The major contributions of this research include using multi-way graph partitioning to model a challenging performance problem critical to sustainable growth of e-commerce portals, creative problem transformation for simplifying a complex problem, and incremental objective function evaluation that can benefit any iterative solution heuristics.

Acknowledgements

xyz

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In the last decade, online e-commerce businesses, represented by the e-commerce portals, have grown significantly and become an important sector of world economy. This dissertation helps address the server scalability problem for supporting the sustainable growth of the online e-commerce industries.

1.1 Distributed Software Components as a New Trend of IT Industries

Since early 1990s, the IT industries have been shifting their design and implementation technologies to software frameworks and architectures based on distributed software components [17][20] to better control the software complexity, promote specialized computing and system integration, and support middleware for object-oriented networking. A *software component* is a software unit that usually exists in binary form, exposes a public *Application Programming Interface (API)*, and is independently deployable [20][16][10][18]. Example software component technologies include Microsoft's dynamic linking libraries, Active-X controls, and COM components; and Java's JavaBeans. Software component approach completely separates the usage and implementation of a software component, and makes it sharable by multiple applications and easily replaceable. A *distributed software component* technology further supports communication abstraction, and uses a generic software framework to provide transparent communication ability to network-blind software components. Example distributed software component technologies include Microsoft's DCOM and COM+ [18], Java's Enterprise JavaBeans (EJB) [16], and Object Management Group's CORBA components [10]. Since 1995, the US Department of Defense mandated that all of its contracted software projects must be implemented with software component technologies.

A related new development of the last decade is the ubiquitous networking. Web and Internet technologies have made online e-business an important branch of world economy. A typical e-commerce portal has three tiers: the presentation tier (running on a Web server) for generating presentation documents for a client's Web browser to render, the business logic tier (running on an application server) for implementation of business logics for the portal, and the data tier (supported by databases). Both the Web servers and the application severs are based on distributed component technologies. For example, both Java's servlets running in a Java servlet container and Microsoft's ASP pages running on an IIS Web server are (converted into) distributed software components (in a more general sense), so are the EJBs running in an EJB container on an application server or the COM+ components running on a .NET Transaction Server [20].

A major challenge for today's e-commerce portals is their scalability: whether a portal can provide fast response when the number of their concurrent clients increases. To provide such scalability, a heavy-duty portal, like yahoo.com, typically uses a cluster of dozens of server machines, connected through a (relatively) fast fiber local area network (LAN), for both of its Web servers and application servers. Since the distributed software components can be independently deployed in any server containers on any of the server machines and communicate with each other transparently, they can take advantage of the hardware parallelism among the server machines to improve the portal scalability.

For a particular hosted computation, it will not be over until all of its employed components finish. Each software component may have different computation load for a particular use case. Each server machine also may have its own particular computing ability based on its resources like CPU speed and memory size. While a fiber-based LAN is faster than its copper version, sending a message through it is still a few orders slower than today's CPU speed (partially due to software overhead for buffer copying to support layered implementation). Communications between two components are much slower if they are assigned to two different server machines than to the same one. Now we have the basic form of software component allocation problem: for a particular computation, what is the optimal allocation of the participating software components to the server machines so that the computation workload of all the involved server machines are balanced, and the total load of communications between any pair of involved components is minimized. We can notice that there are here two conflicting objectives. This problem will be more complicated when we also consider the management of client session data, or when different stages of a computation may have different computation and communication patterns, as we will see in the next section.

While the World Wide Web has had big impact on our society, it only supports a limited form of client-server software architecture. The IT industries have started to work on the next wave of Internet revolution: the Application Service Provider model of computing [20], by which software applications will be maintained by domain experts on service-provider servers and accessed by clients with Web browsers through service provider's portals. This paradigm eliminates software installation on client's computers, and promotes specialized computing and service integration. The success of this new paradigm also heavily depends on whether we can run hosted applications on clustered servers efficiently. Therefore the importance of the study of efficient software component allocation problems goes beyond today's Web servers and application servers.

1.2 Software Component Allocation Problems

The potentially important software component allocation problems can be divided into two categories: *static* and *dynamic*, depending on whether the optimized allocations can be computed off-line or whether the components can migrate across server machines during execution. Multiple processors could be tightly coupled inside a single server machine; and a subset of the software components may need to maintain its unique session data to serve a particular remote client. All these make software component allocation problems different from the traditional job scheduling on distributed systems [1][21].

A server cluster typically comprises dozens (50 or more) of server machines, each of which may have different computation speed, connected by a fiber LAN. In this study we limit our attention to bus-type LAN, the dominant one in today's IT industries, for which all messages will share the same LAN bandwidth, and at any instance, there could be at most one sender but multiple receivers. The speed for a message to travel the LAN is much lower than the CPU speed of the server machines.

A hosted software application is typically made up of dozens to hundreds of software components that could be distributed in any of the component containers running on the server machines. Without loss of generality, we assume each server machine will run one component container. For each typical hosted computation, each of the involved software components has an average computation load and an average communication load with each of the other participating components. Since the hosted applications are designed for providing well-defined set of specialized services, it is reasonable to assume that these average computation load and communication load values could be obtained by profiling the applications on a single server machine (similar to Unix profiler utility *prof*).

Now we can model, simplified for essence, a software component allocation problem as a multi-way graph partitioning problem. We abstract a hosted application as an undirected graph $G = (V, E)$ in which each vertex represents a software component and each edge represents a runtime communication requirement. Let function $w_1 : V \to \Re^+$ ($\Re^+$ is the set of positive real numbers) represent the average computation load of the software components, and function $w_2 : E \to \Re^+$ represent the average communication load of the communication requirements. Assume the software components need to run on *m* server machines, and $\pi : V \to \{1, 2, \cdots, m\}$ represents one of the component assignment. For each $1 \le i \le m$, we use $P_\pi(i)$ to denote the partition of the vertices (software components) assigned by π to server machine *i*; a fixed real rate r_i to represent the relative computing ability of server machine *i* (a larger rate implies a slower execution); and $w_1(P_\pi(i)) = \sum_{v \in P_\pi(i)} w_1(v)$ to represent the total computation load of components assigned to partition *i*. Let the importance of computation time on the server machines relative to the communication time on the LAN be represented by a real ratio $0 < t < 1$. Now the question is, how to find an optimal assignment $\pi : V \to \{1,2,\cdots,m\}$ to minimize the objective function

$$f(\pi) = t \cdot W_1(\pi) + (1-t) \cdot W_2(\pi)$$

where $W_1(\pi)$ represents the degree of load balance as defined below

$$W_1(\pi) = \sum_{1 \le i < j \le m} | r_i \cdot w_1(P_\pi(i)) - r_j \cdot w_1(P_\pi(j)) |$$

and $W_2(\pi)$ represents the total communication cost as defined below

$$W_2(\pi) = \sum_{\substack{e=\{u,v\}\in E \\ \pi(u)\neq\pi(v)}} w_2(e)$$

Here the summation operator reflects our assumption that all communications share the same LAN bandwidth.

Since the basic graph bisection problem, for which the partition number is two and the vertices and edges have uniform weights, is NP-complete [1] and a special case of our simplified formulation, all the software component allocation problems described in this proposal are NP-hard.

1.3 Methodologies

The component allocation problem has many variations based on different assumptions, and this research will focus on the one where the communication cost needs to be minimized under the constraint that the computation workload is evenly distributed.

Mathematical modeling is the foundation of this research. The properties of the mathematical model will be studied to derive a problem transformation algorithm that can convert the two-objective-function optimization problem into an equivalent one with a single objective function. For efficient problem solution, solution space neighborhood will be designed to support incremental evaluation of the objective function, which can benefit any solution algorithm based on iterative solution searches. Simulated annealing is chosen as the meta-heuristic for deriving a solution heuristic. Experimental comparisons will be conducted between the proposed simulated annealing algorithm and repeated random solutions generated in the same amount of time, and between the proposed simulated annealing algorithm and local optimization for both solution quality and running time.

1.4 Major Contributions

The major contributions of this research include:

- Using multi-way graph partitioning to model an important application server performance problem critical to the sustainable growth of online e-commerce industries.

- Proving that this problem is NP-hard, so no efficient algorithms could ever be designed to produce optimal solutions to it in practical time.

- Designing a problem transformation algorithm to convert the problem with multiple objective functions into an equivalent typical combinatorial optimization problem with a single objective function.

- Designing a scheme for incremental objective function evaluation that can improve the performance of any iterative solution heuristics.

- Deriving an efficient heuristic solution based on simulated annealing, and studying the sensitivity of the heuristic to its various parameters.

- Designing experiments to study the performance of our heuristic relative to repeated random solutions and local optimization.

1.5 Dissertation Outline

The dissertation consists of six chapters described in the following manner:

Chapter 1 introduces the important scalability problem of E-commerce portal servers and the associated software component allocation problem, presents the solution methodologies and major contributions of this research.

Chapter 2 provides surveys of commonly used meta-heuristics for combinatorial optimization problems and describes the characteristics of each heuristic.

Chapter 3 describes the problem formulation for multi-way graph partition and problem transformation.

Chapter 4 provides the design of solution space neighborhood as well as the incremental evaluation of the objective function.

Chapter 5 provides the design of a simulated annealing heuristic for the proposed software component allocation problem, and conducts sensitivity analysis to its various parameters and cooling schedule.

Chapter 6 uses extensive experimental comparisons to study the performance of the simulated annealing algorithm relative to repeated random solutions and local optimization.

Chapter 7 concludes with some observations and future work.

Chapter 2

Graph Partitioning and Solution Heuristics

This chapter surveys the graph partitioning problems as well as the major meta-heuristics for combinatorial optimization.

2.1 Graph Partitioning

Graph partitioning is one of the richest fields of computing algorithms, with wide applications in parallel processing, distributed computing, VLSI design and layout, network partitioning, distributed database design, and sparse matrix factorization [4][12][1][5][22]. The most popular heuristics for graph partitioning include the Kernighan-Lin algorithm (KL) [13] for graph bisection and its enhancement variation [4]. Johnson et al. [12] performed an extensive study of the simulated annealing algorithm for the graph bisection problem and observed that simulated annealing on the average performed better than KL. Bui et al. [1] developed a genetic algorithm for multi-way graph partitioning, and conducted extensive experimental evaluations of the related algorithms to show its superior performance. Tao et al. [21], as well as many other researchers, used graph partitioning to address the problem of optimized allocation of processes/jobs to the processors in a distributed environment. Tao et al. [22] proposed *stochastic probe*, a new effective and generic meta-heuristic, and demonstrated its superior performance in multi-way graph partitioning.

The existing studies of graph partitioning usually simplify the problem constraints described in this research by dropping the weights of the vertices or edges.

2.2 Solution Heuristics

For NP-hard problems, we can only obtain optimal solutions for small problem instances. For practical problem instance sizes, heuristics must be used to find optimized solutions within reasonable time frame. Unlike algorithms, heuristics do not guarantee optimal. A *heuristic* is an algorithm that tries to find good solutions to a problem but it cannot guarantee its success. Most heuristics are not established on rigid mathematical analysis, but on human intuitions, understanding of the properties of the problem at hand, and experiments. The value of a heuristic must be based on performance comparisons among competing heuristics. The most important performance metrics are solution quality and running time.

The term *meta-heuristics*, first introduced in Glover [6], derives from the composition of two Greek words. The suffix *meta* means "beyond, in an upper level" and *heuristic* means "to find, discover". A *meta-heuristic* is a strategy that guides the search process, or an abstraction of a class of similar heuristics. Meta-heuristics are approximate and usually non-deterministic, not problem-specific. It may incorporate mechanisms to avoid getting trapped in confined areas of the search space. The basic concepts of meta-heuristic permit an abstract level description. And it may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy. More advanced meta-heuristics are used to guide the solution searches today [1]. To effectively resolve a problem based on a meta-heuristic, we need to have more understanding of the characteristics of the problem, and creatively design and implement the major components of the meta-heuristics. As a consequence, using a meta-heuristic to propose an effective heuristic to solve an NP-hard problem is an action of research.

In the following we outline the most important meta-heuristics from a conceptual point of view.

2.2.1 Local Optimization

A general heuristic search technique, *local optimization* is also called *greedy algorithm* or *hill-climbing.* It attempts to improve on the solution by a series of incremental, local changes. Each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it finds a local minimum. The high level algorithm is sketched in Algorithm 1.

Algorithm 1. Local optimization

1. Get an initial solution S.
2. While there is an untested neighbor of S do the following.
   2.1 Let S' be an untested neighbor of S.
   2.2 If $\cos t(S') < \cos t(S)$, set $S = S'$.
3. Return S.

Local optimization starts from a random initial solution, and it keeps migrating to better neighbors in the solution space. If all neighbors of the current partition are worse, then the algorithm stops. This scheme can only find local optimal solutions that are better than all of their neighbors but they may not be the global optimal solutions, as illustrated in **Figure 1**.

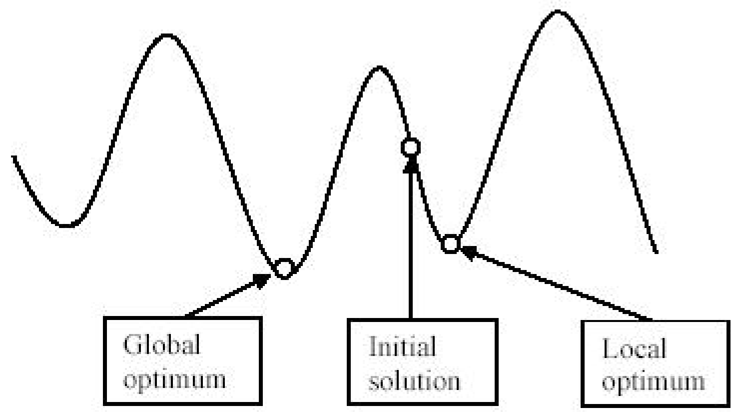


Figure 1 Local vs. global solutions

2.2.1 Genetic Algorithm

Generic algorithm is an iterative procedure maintaining population of structures that are candidate solutions to specific domain challenges. During each generation the structures in the current population are rated for their effectiveness as solutions, and on the basis of these evaluations, a new population of candidate structures is formed using specific "genetic operators" such as reproduction, crossover, and mutation. It is based on the analogy of combinatorial optimization to the mechanics of natural selection and natural genetics. Its application in combinatorial optimization area can be traced back in early 1960s [8].

A genetic algorithm starts with a set of initial solutions (chromosomes), called a population. This population then evolves into different populations for hundreds of iterations. At the end, the algorithm returns the best member of the population as the solution to the problem. For each iteration or generation, the evolution process proceeds as follows. Two members of the population are chosen based on some probability distribution. These two members are then combined through a crossover operator to produce an offspring. With a low probability, this offspring is then modified by a mutation operator to introduce unexplored search space to the population, enhancing the diversity of the population (the degree of difference among chromosomes in the population). The offspring is tested to see if it is suitable for the population. If it is, a replacement scheme is used to select a member of the population and replace it with the new offspring. Now we have a new population and the evolution process is repeated until certain condition is met, for example, after a fixed number of generations. This genetic algorithm generates only one offspring per generation. Such a genetic algorithm is called steady-state genetic algorithm [24][19], as opposed to a generational genetic algorithm that replaces the whole population or a large subset of the population per generation. A typical structure of a steady-state genetic algorithm is given in Algorithm 2 [2].

Algorithm 2. Genetic algorithm

```
1. Create initial population of fixed size.
2. Do the following
   2.1 Choose parent1 and parent2 from population.
   2.2 Offspring = crossover (parent1, parent2).
   2.3 Mutation (offspring).
   2.4 If suited (offspring), then
         Replace (population, offspring);
   Until (stopping condition).
3. Return the best answer.
```

2.2.2 Simulated Annealing

Simulated annealing is commonly said to be the oldest among the meta-heuristics and surely one of the first algorithms that has an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (Metropolis algorithm) and it was first presented as a search algorithm for combinatorial optimization problems in Kirkpatrick [14]. In 1983, Kirkpatrick and his coworkers proposed a method of using a Metropolis Monte Carlo simulation to find the lowest energy (most stable) orientation of a system. Their method is based upon the procedure used to make the strongest possible glass. This procedure heats the glass to a high temperature so that the glass is a liquid and the atoms can move relatively freely. The temperature of the glass is slowly lowered so that at each temperature the atoms can move enough to begin adopting the most stable orientation. If the glass is cooled slowly enough, the atoms are able to "relax" into the most stable orientation. If the temperature is lowered rapidly, some atoms may get stuck in foreign positions. The slow cooling process is known as *annealing*, and so their method is known as Simulated Annealing.

The simulated annealing heuristic starts by generating an initial solution (either randomly or heuristically constructed) and initializing the temperature parameter T. Then, at each iteration, a neighbor solution is randomly sampled and it is accepted as new current solution depending on the current cost, neighbor cost and temperature. If the neighbor improves the current cost, then the neighbor becomes the new current solution for the next iteration. If the neighbor worsens the current cost, it will be accepted as the new current solution with a probability. When the temperature is high, the probability is not sensitive to how worse the neighbor is. But when the temperature is low, the probability to accept a worsening neighbor will shrink with the extent of the worsening. When no improvement in solution cost happens for a period of time, the temperature will be decreased by a very small amount, and the above looping repeats. The process will stop when some termination criteria is met [23].

Simulated annealing is unique among all the other meta-heuristics for combinatorial optimization in that it has been mathematically proven to converge to the global optimum is the temperature is reduced sufficiently slowly. But this theoretical result is not too interesting to practitioners since very few real world problems will be able to afford such excessive execution time [23]. A simulated annealing heuristic is based on the following pseudo-code in Algorithm 3.

Algorithm 3. Simulated annealing

1. Get an initial solution S.
2. Get an initial temperature $T > 0$.
3. While not yet *frozen* do the following.
   3.1 Perform the following loop L times.
      3.1.1 Pick a random neighbor S' of S.
      3.1.2 Let $\Delta = \cos t(S') - \cos t(S)$.
      3.1.3 If $\Delta \le 0$ (downhill move),
         Set $S = S'$.
      3.1.4 If $\Delta > 0$ (uphill move),
         Set $S = S'$ with probability $e^{-\Delta/T}$.
   3.2 Set $T = rT$ (reduce temperature).
4. Return S.

Simulated annealing approach involves a pair of nested loops and two additional parameters, a *cooling ratio r*, which is between *zero* and *one*, and an integer *temperature length L*. In step 3 of the above algorithm, the term *frozen* refers to a state in which no further improvement in $\cos t(S)$ seems likely. The most important of this process is the loop at Step 3.1. Note that $e^{-\Delta/T}$ will be a number in the interval $(0, 1)$ when Δ and T are positive, and rightfully can be interpreted as a probability that depends on Δ and T. The probability that an uphill move of size Δ will be accepted diminishes as the temperature declines, and, for a fixed temperature T, small uphill moves have higher probabilities of acceptance than large ones [12].

While comparing local optimization and simulated annealing, we find they mainly differ in the extent to accept worsening neighbors. For simulated annealing, it starts with random walk in the solution space. When a random neighbor is better, it always takes it. But if the neighbor is worsening, its possibility of accepting it is reduced slowly. Simulated annealing becomes local optimization when the temperature is very low. Johnson made a critical evaluation for the performance of the simulated annealing approach to the graph partition problem and compared its performance with that of the Kernighan-Lin approach. In general, simulated annealing is time-consuming, but it has been very successfully applied to numerous combinatorial optimization problems.

2.2.3 Tabu Search

Tabu search is among the most cited and used meta-heuristics for combinatorial optimization problems. The basic ideas were first introduced in Glover [6][7] since 1986. It explicitly uses the history of the search, both to escape from local optima and to implement an explorative strategy. Tabu search applies a best improvement local search as basic ingredient and uses a short-term memory to escape from local optima and to avoid cycles. The short-term memory is implemented as a tabu list that keeps track of the most recently visited solutions and forbids moves toward them. The neighborhood of the current solution is thus restricted to the solutions that do not belong to the tabu list. Tabu means prohibition here.

The advocates of tabu search disagree with the analogy of optimization process to metal annealing process. They argued that when a hunter entered in an unfamiliar environment, he will not search randomly first but zero in to the area that appears most promising in finding games. This is similar to the greedy local optimization algorithm. Only when neighboring areas are all worse than the current area will the hunter be willing to search through worsening neighboring areas in hope of finding a better local optimum.

Tabu search differs from simulated annealing at two key aspects. It is more aggressive and deterministic. A tabu search heuristic starts by generating a random partition as the current solution. It then executes a loop until some stopping criteria are reached. During each iteration, the current solution is replaced with its best neighbor that is not tabued on the tabu list. The high level algorithm is sketched in Algorithm 4.

Algorithm 4. Tabu search

1. Get a random initial solution S.
2. While stop criterion not met do:
   2.1 Let S' be a neighbor of S maximizing
   $\Delta = \cos t(S') - \cos t(S)$ and not visited
   in the last t iterations.
   2.2 Set $S = S'$.
3. Return the best S visited.

Chapter 3

Problem Formulation and Transformation

In this chapter, we formulate the optimized software component allocation problem as a multi-way graph partitioning problem, prove it to be NP-hard, and simplify it with a problem transformation algorithm. The results in this chapter, as well as those in Chapter 4, are foundations of this research and can support any solution methodologies.

3.1 Problem Statement

3.1.1 Problem Assumptions

A scalable application server is implemented by a cluster of server machines connected by a high-speed fiber local area network (LAN). The LAN operates in a bus mode, like Ethernet, so all inter-machine communications share the same LAN bandwidth. The inter-machine communications are much slower than server machine CPU speed and thus should be avoided if possible. All the server machines have the same computation power, and no residual computation (all server machine resources are ready for use).

Server applications are implemented with distributed component technologies. A server application comprises dozens of software components that may need to communicate with each other at running time. The execution of an application will not be over until all the participating components finish their computation. Each software component can run transparently on any of the server machines and communicate with each other. Intercomponent communications are much slower if the senders and receivers are allocated on different server machines. Based on a profiler utility, it is known for each typical use case the average computation load of each participating software component and the average communication load between each pair of software components.

3.1.2 Problem Statement

Given the above assumptions, how to allocate the software components to the application server machines so that the communication overhead is minimized under the constraint that the computation workload is distributed evenly across the server machines.

3.2 Problem Formulation as Multi-way Graph Partitioning

A component-based server application can be modeled as a graph: each vertex representing a software component, each edge representing a communication requirement between a pair of incident software components at running time. We can use vertex weights to represent a component's computation load, and edge weights represent the potential communication load between the two incident components. The application server machines can be represented as partitions of the software components. To minimize the computation work load, we need to allocate the vertices to the partitions so that the vertex weights are evenly distributed across the partitions. To minimize the inter-machine communication overhead, we need to allocate the vertices so that the summation of the edge weights for those edges crossing the partitions (the two incident vertices of an edge belonging to two partitions) will be minimized. Here the summation operator is used to model our assumption that all inter-machine communications share the same LAN bandwidth, as is the case for most of today's enterprise-quality e-commerce portals.

Given an undirected graph $G=(V,E)$, an integer m ($1 \le m \le |V|$), and two weight functions $w_1 : V \to I$ and $w_2 : E \to I$ (I is the set of positive integers), an *m*-way partitioning π of G is a function $\pi : V \to \{1, 2, ..., m\}$ such that $V = P_\pi(1) \cup P_\pi(2) \cup ... \cup P_\pi(m)$, where $P_\pi(i) = \{v \in V | \pi(v) = i\}$ for $1 \le i \le m$. For any subset $C \subseteq V$, let $w_1(C) = \sum_{v \in C} w_1(v)$. Our objective is to derive *m*-way partitioning π that can minimize

$$W_2(\pi) = \sum_{\substack{e=\{u,v\}\in E \\ \pi(u)\neq\pi(v)}} w_2(e)$$

under the constraint that

$$W_1(\pi) = \sum_{1 \le i \le j \le m} \left| w_1(P_\pi(i)) - w_1(P_\pi(j)) \right|$$

is minimal. We call $w_1(v)$ the *vertex weight* of vertex *v*, $w_2(e)$ the *edge weight* of edge *e*. We call $W_1(\pi)$ the *balance measure* that measures the evenness of the computation load distribution, and $W_2(\pi)$ the *weighted cut size* that measures the total cost of communications across the LAN. Informally, we want to partition the graph vertices into mutually exclusive subsets so that the total weight of the edges crossing the subsets is minimized under the condition that the vertex weights are distributed evenly among the subsets.

It can be observed that this problem is unusual in the combinatorial optimization literature since it contains two objective functions, one of which is embedded in the problem constraint; and these two objective functions conflict with each other: for example, while allocating all vertices to the same partition can minimize the weighted cut size, it will be the worst case for vertex weight distribution.

As examples, **Figure 2** shows two different schemes of partitioning a set of five vertices into two or three partitions. The numbers inside the vertices are vertex weights. The numbers beside edges are edge weights. In **Figure 2** (a), the five vertices are allocated into two partitions: partition 1 has a total vertex weight of 7, partition 2 has a total vertex weight of 5, thus $W_1(\pi)$ = $|7-5|$ = 2; and the allocation has a weighted cut size $W_2(\pi)$ of 7. In **Figure 2** (b), the five vertices are allocated into three partitions: partition 1 has a total vertex weight of 4, partition 2 has a total vertex weight of 5, partition 3 has a total vertex weight of 3, thus $W_1(\pi)$ = $|4-5|+|4-3|+|5-3|$ = 1 + 1 + 2 = 4; and the allocation has a weighted cut size $W_2(\pi)$ of 8.

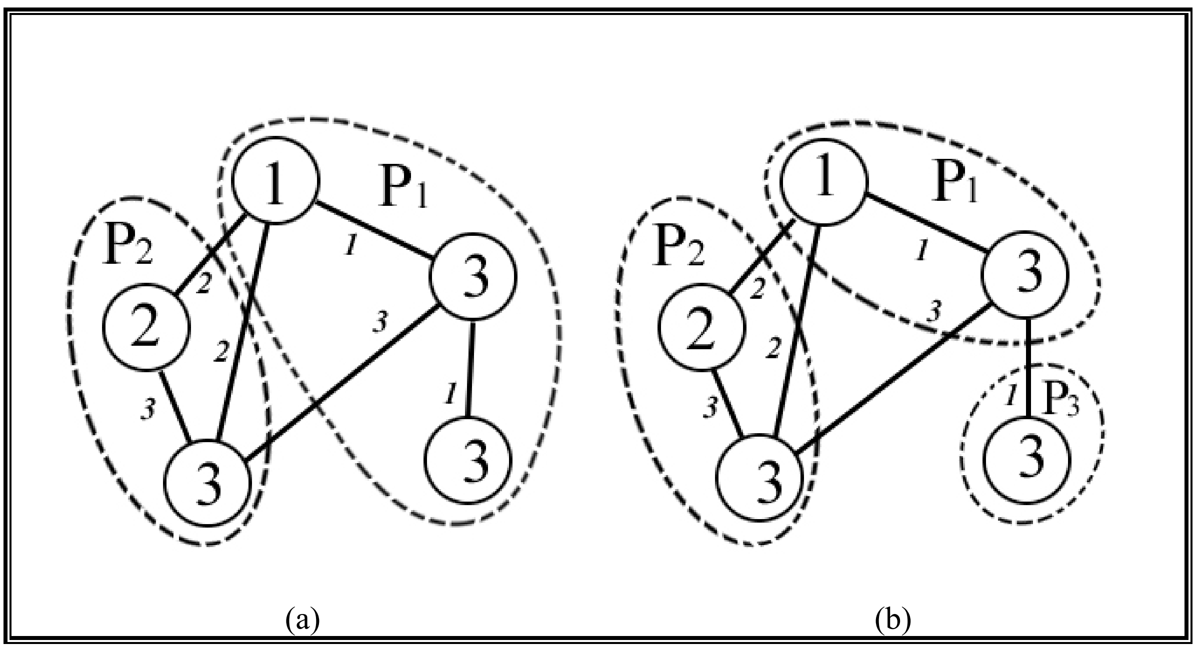


Figure 2 Multi-way graph partitioning

3.3 NP-hardness of the Problem

Now we prove that the multi-way graph partitioning problem described in this dissertation is NP-hard.

NP-Hardness Theorem: *The multi-way graph partitioning problem described in this research is NP-hard.* □

Proof: We first prove that graph bisection is a special case of our multi-way graph partitioning.

Given any graph $G=(V,E)$ where $|V|$ is even, the graph bisection problem seeks a bisection of V into the left and right two partitions P_1 and P_2 so that $|P_1|=|P_2|$ and $|\{e \in E \mid e=\{u,v\}, u \in P_1, v \in P_2\}|$, the number of edges crossing the two partitions, is minimized. Given any problem instance for graph bisection, we can construct a corresponding problem instance for the multi-way graph partitioning problem by letting $m=2$, $w_1(v)=1$ for all $v \in V$, and $w_2(e)=1$ for all $e \in E$. Suppose we get π as one of the optimal solutions for this multi-way partitioning problem instance, then $W_1(\pi)=0$ must be true since $|V|$ is even and all vertices have the same unit weight, and $W_2(\pi)$ is minimized. Since all edges have the same unit weight, $W_2(\pi)$ is exactly the same as the number of edges crossing the two partitions. Therefore we can conclude that, given any graph bisection problem instance, we can solve it as a multi-way graph partitioning problem, and the resulting optimal solution to the multi-way partitioning problem instance is also an optimal solution to the original graph bisection problem instance. Therefore graph bisection problem is a special case of our multi-way graph partitioning problem.

But it is well known that graph bisection is an NP-complete problem [3]. If our multi-way graph partitioning problem were not NP-hard, it would imply that graph bisection were not NP-complete, a contradiction. Therefore we conclude that our multi-way graph partitioning problem is NP-hard. □

Since the multi-way graph partitioning problem is NP-hard, it is impossible to have algorithms to solve its practical problem instances within realistic time frames. We have to resort to heuristic approaches to search for optimized solutions within time frames suitable for the particular application domains.

3.4 Problem Transformation

The multi-way graph partitioning problem formulated in this research differs from traditional combinatorial optimization problems in its two objective functions, one of which is embedded in the problem constraint. In this section we introduce a problem transformation algorithm to convert any instance of this problem into another problem instance of an equivalent simpler problem with only a single objective function.

Other reason for our introduction of the problem transformation is for efficient evaluation of the objective function. The time complexity of an iterative algorithm is largely determined by the efficiency by which the objective functions and the constraint conditions are evaluated. Since the move (operation) for each iteration only makes local changes to the current solution, it is desirable to have the ability to incremental update the old value of the objective function to obtain its new one after the move. While $W_2(\pi)$ allows simple incremental update after each vertex move or vertex exchange operation, $W_1(\pi)$ needs at least $O(m)$ update steps after each of such operations. The new objective function resulting from our problem transformation is easier for incremental evaluation, as shown in Chapter 4.

Graph Transformation Algorithm: Given an undirected graph $G=(V,E)$ that is needs to be divided into *m* partitions, we transform G into another complete graph $G^*=(V,E^*)$ where $E^*=\{\{u,v\}\mid u,v\in V\}$, and define a new edge weight function $w_3 : E^* \to \Re^+$ ($\Re^+$ is the set of all positive real numbers) such that

$$w_3(e)=\begin{cases} w_1(u)w_1(v)R - w_2(e) & \text{if } e=\{u,v\}\in E; \\ w_1(u)w_1(v)R & \text{if } e=\{u,v\}\in E^* - E \end{cases}$$

where R is a positive real number called the *augmenting factor*. The corresponding new *m*-way graph partitioning problem is to find an *m*-way partition π of graph G^* to maximize its objective function

$$W_3(\pi)=\sum_{\substack{e=\{u,v\}\in E^* \\ \pi(u)\neq\pi(v)}} w_3(e)$$

Problem Transformation Theorem: *Given any instance of the multi-way graph partitioning problem, if the value of R in the graph transformation is larger than the total edge weight of G, or* $\sum_{e\in E} w_3(e)$*, a solution* π *that maximizes* $W_3(\pi)$ *will also minimize* $W_2(\pi)$ *under the constraint that* $W_1(\pi)$ *is minimized.* □

But before we can prove this theorem, we need some preparations.

Definition: Given a positive integer k, a *partition of integer* k is a set of positive integers $\{k_1, k_2, ..., k_m\}$ $(m \le k)$ such that $k = \sum_{i=1}^{m} k_i$. □

Max-Prod-Min-Diff Theorem: Given positive integers m and k such that $m \le k$, any partition of k into $P = \{k_1, k_2, ..., k_m\}$ maximizing $\sum_{1 \le i \le j \le m} k_i k_j$ will minimize $\sum_{1 \le i < j \le m} |k_i - k_j|$. □

First we prove the following two lemmas.

Lemma 1: Let x and y be positive integers. If $x > y + 1$, then $x^2 + y^2 > (x-1)^2 + (y+1)^2$. □

Proof: If $x > y + 1$, then $2x - 1 > 2y + 1$. Therefore, $x^2 - x^2 + 2x - 1 > y^2 - y^2 + 2y + 1$, or $x^2 - (x-1)^2 > (y+1)^2 - y^2$. So we have the lemma. □

Lemma 2: Let m and k $(m \le k)$ be positive integers and $P = \{k_1, k_2, ..., k_m\}$ a partition of k. Assume that there exists a pair x and y in P such that $x - y > 1$. Let $x' = x - 1$, $y' = y + 1$, and $P' = P - \{x, y\} \cup \{x', y'\}$. We have

$$\sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P}} |k_i - k_j| > \sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P'}} |k_i - k_j| \tag{1}$$

and

$$\sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P}} k_i k_j < \sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P'}} k_i k_j \tag{2}$$

□

Proof: We can partition P into $P = P_1 \cup \{x\} \cup P_2 \cup \{y\} \cup P_3$, where:

P_1—the set of numbers in P that are greater than or equal to x,

P_2—the set of numbers in P that are smaller than x and great than y,

P_3—the set of numbers in P that are equal to or smaller than y.

Let N_1, N_2, N_3 be the cardinalities of P_1, P_2, and P_3 respectively. We have $N_1 + 1 + N_2 + 1 + N_3 = m$, and

$$\sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P'}} \left| k_i - k_j \right| = \sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P}} \left| k_i - k_j \right| + \left(N_1 - N_2 - 1 - N_3 \right) + \left(-N_1 - 1 - N_2 + N_3 \right)$$

$$= \sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P}} \left| k_i - k_j \right| - 2\left(1 + N_2\right).$$

So we have Inequality (1).

Because $x > y + 1$, from Lemma 1, we have $x^2 + y^2 > x'^2 + y'^2$. Since

$$k^2 = \sum_{\substack{1 \le l \le m \\ k_l \notin \{x, y\}}} k_l^{\,2} + \left(x^2 + y^2\right) + 2 \sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P}} k_i k_j = \sum_{\substack{1 \le l \le m \\ k_l \notin \{x', y'\}}} k_l^2 + \left(x'^2 + y'^2\right) + 2 \sum_{\substack{1 \le i < j \le m \\ k_i, k_j \in P'}} k_i k_j ,$$

we have Inequality (2). □

Proof of Max-Prod-Min-Diff Theorem: Since partition $P = \{k_1, k_2, ..., k_m\}$ maximizes $\sum_{1 \le i < j \le m} k_i k_j$, by Lemma 2, there can be no $x, y \in P$ such that

$x > y+1$. On the other hand, if $\max(P)$-$\min(P) \le 1$, then $\sum_{1\le i<j\le m} |k_i - k_j|$ must reach its smallest possible value $r(m-r)$, where r is the remainder of k/m. The theorem is thus proved. □

Proof of Problem Transformation Theorem: It has been proven by Lee et al. [15] that, if $R > \sum_{e\in E} w_2(e)$, any partitioning π that maximizes

$$W_3(\pi) = \sum_{\substack{e=\{u,v\}\in E^* \\ \pi(u)\ne\pi(v)}} w_3(e) = R \sum_{1\le i<j\le m} w_1(P_\pi(i))w_1(P_\pi(j)) - W_2(\pi)$$

will minimize $W_2(\pi)$ under the constraint that

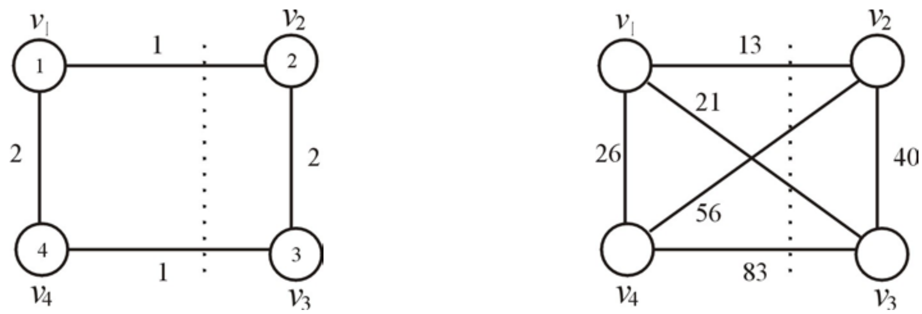
$$\sum_{1\le i<j\le m} w_1(P_\pi(i))w_1(P_\pi(j))$$

is maximized. Now we only need to prove that maximizing $\sum_{1\le i<j\le m} w_1(P_\pi(i))w_1(P_\pi(j))$ is equivalent to minimizing $W_1(\pi)$. But this follows directly from our *Max-Prod-Min-Diff Theorem* above. □

The following is an example for the graph transformation. The vertex weights are marked inside the vertices. The edge weights are marked along the edges. Figure 3 (a) shows the original graph to be bisected, and Figure 3 (b) shows its equivalent complete graph obtained from our graph transformation. The two partitions are separated by a dotted line. Since the total edge weights is 6, we set $R = 6+1 = 7$.

$W_1(\pi) = 0$, $W_2(\pi) = 2$

$W_3(\pi) = 173$

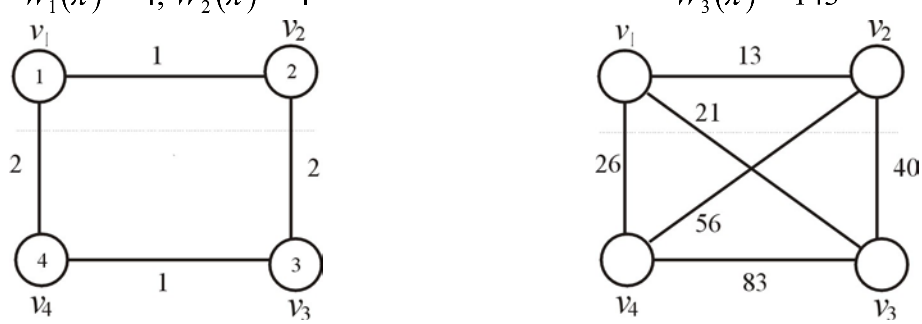


(a) (b)

Figure 3. Example of problem transformation-optimal

Figure 3 (a) shows $W_1(\pi) = |(1+4)-(2+3)| = 0$ and $W_2(\pi) = 1+1 = 2$. Let $\overline{uv}$ represent the edge weight $w_3(\{u,v\})$. Figure 3 (b) shows $\overline{v_1v_2} = 1\times 2\times 7-1 = 13$, $\overline{v_1v_3} = 1\times 3\times 7 = 21$, $\overline{v_1v_4} = 1\times 4\times 7-2 = 26$, $\overline{v_2v_3} = 2\times 3\times 7-2 = 40$, $\overline{v_2v_4} = 2\times 4\times 7 = 56$, $\overline{v_3v_4} = 3\times 4\times 7-1 = 83$. $W_3(\pi) = 13+21+56+83 = 173$. On the another hand, if we partition the graph as in **Figure 4** (c) and **Figure 4** (d), then $W_1(\pi) = |(3+4)-(1+2)| = 4$, $W_2(\pi) = 2+2 = 4$, $W_3(\pi) = 26+21+56+40 = 143$. This example confirms that a solution π with larger value for $W_3(\pi)$ will have smaller values for $W_1(\pi)$ and $W_2(\pi)$.

$W_1(\pi) = 4$, $W_2(\pi) = 4$ $W_3(\pi) = 143$



(c) (d)

Figure 4. Example of problem transformation-nonoptimal

Chapter 4

Incremental Objective Function Evaluation

Most meta-heuristics are based on iterative moves in the solution space. During each iteration, the current solution is perturbed by a *move* to obtain its neighbor. No matter which meta-heuristic is adopted, the running time of the resulting heuristics will be dominated by the evaluation time for the objective functions.

In this research we design a scheme to support incremental evaluation of the objective function $W_3(\pi)$ to reduce its evaluation time. The idea is, since each move only perturbs the current solution locally, we could avoid the evaluation of the entire objective function by modifying its most recent old value. We introduce a *gain* function for each *move* so that the new value of the objective function after a move equals to the function value immediately before the move plus the gain of the move. We use a gain table to support the incremental update of the gain value for all valid moves. This methodology is based on runtime/memory tradeoffs, often observed in the design of efficient algorithms like dynamic programming.

But first we need to design the types of moves in the solution space and the corresponding solution space neighborhood.

4.1 Solution Space Neighborhood Design

Let X be the set of all mappings $V \to \{1,2,...,m\}$. X is the solution space here. The transformed multi-way graph partitioning problem can be presented as

$$\textbf{maximize} \quad W_3(\pi): \pi \in X$$

where $W_3(\pi)$ is the new objective function.

A wide range of heuristic algorithms for solving problems capable of being written in this form can be characterized conveniently by reference to sequences of moves that lead from one trial solution (selected $\pi \in X$) to another. Let S be the set of all defined moves. We use $S(\pi)$ ($\pi \in X$) to denote the subset of moves in S applicable to π. For any $s \in S(\pi)$, $s(\pi)$, the new solution obtained by applying move s to π, is called a *neighbor* of π. We call $\{s(\pi) \mid s \in S(\pi)\}$ the *neighborhood* of solution π. If $s(\pi) \neq s'(\pi)$ for any pair of different moves $s, s' \in S(\pi)$, we can use $|S(\pi)|$ to denote the *neighborhood size* of solution π.

In order to optimize the algorithm performance, S should be defines with the following properties [23][22]:

- *Reachability*: Given any two solutions π and π' in X, it should be possible to apply a sequence of moves in S to reach π' from π. This property will greatly increase the probability for an algorithm to converge to the global optimum.

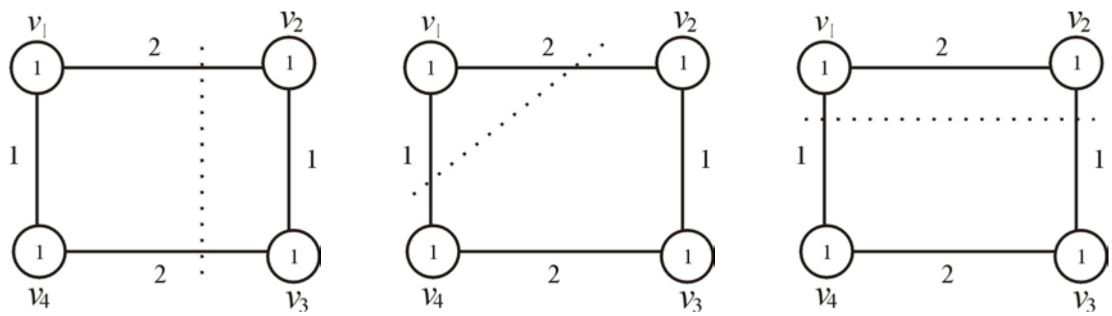
- *Efficiency*: Given any solution $\pi \in X$ and $s \in S$, the cost of $s(\pi)$ can be easily evaluated by incrementally updating the cost of π. This will allow us to avoid evaluating the cost (objective) function $W_3(\pi)$ during each iteration, an operation having time complexity $O(|V|^2)$.

- *Injectiveness*: Given two different moves s, $s' \in S$, for any $\pi \in X$, $s(\pi) \neq s'(\pi)$. This will make sure that each neighbor of the current solution will be checked only once for the current neighborhood search.

For graph partitioning, *vertex move* and *vertex exchange* are two popular categories of moves. Let S_1 be the set of all moves for moving one vertex away from its current partition, and S_2 the set of all moves for exchanging two vertices possessed by two different partitions. Both S_1 and S_2 enjoy the injectiveness property. The cost of the current solution can be incrementally updated for moves from both S_1 and S_2, as will be explained in the next section. Many graph partitioning algorithms [12][15] favor S_1 since it has smaller neighborhood size $|V|(m-1)$, while the average neighborhood size for S_2 is $O(|V|^2)$. We can make the following two further observations about the reachability of S_1 and S_2.

- S_1 also enjoys the reachability property. But if we only allow vertex moves that will not worsen the cost of the current partitioning by $R \cdot \max(w_1) - (\max(w_2) - \min(w_2))$ or more (this is the case when the simulated annealing is in its low-temperature phases, or when the tabu search always has moves with gains of smaller absolute value), then this reachability cannot always be realized. For example, for the graph bisection of G_1 in Figure 5(a), no sequence of vertex moves can transform it to the optimal bisection of G_1 in Figure 5 (c) unless we accept moves that will reduce $W_3(\pi)$ by 27. Figure 5 (b) shows an example vertex move for the bisection in Figure 5 (a). Our claim can be proved by generalizing the weights in G_1.

$W_3(\pi) = 24$ $W_3(\pi) = 18$ $W_3(\pi) = 26$



(a) (b) (c)

Figure 5. Example partitions of G_1, *R* = 7

- In general S_2 does not have the reachability property. For instance, given the graph bisection of G_2 in Figure 6 (a), vertex exchanges will never lead us to the optimal bisection of G_2 in Figure 6 (b) (while vertex moves do) because they cannot change the cardinality of each partition. However, we can easily transform the bisection in Figure 6 (a) to that in Figure 6 (c) by exchanging vertices v_2 and v_4.

$W_3(\pi) = 43$ $W_3(\pi) = 123$

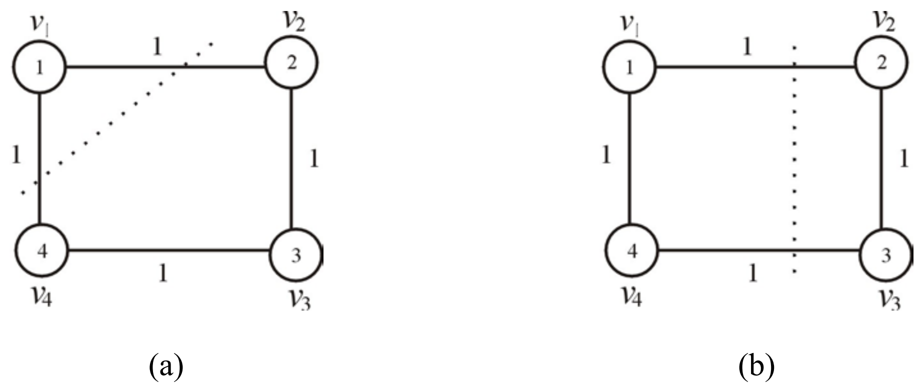


Figure 6. Example partitions of G_2, *R* = 5

For simplicity, this research only uses vertex moves in S_1 in its solution heuristics. Vertex exchanges are left for future work.

4.2 Gain Function for Moves

The strategy for our incremental objective function evaluation is implemented through defining a gain function for evaluating the profit (improvement in objective function value) of making a solution space move.

Given the current partition π and a move $s \in S(\pi)$, we call $W_3(s(\pi)) - W_3(\pi)$ the *gain* of move s.

Given a partition π of G, for any $s \in S_1$, the gain for moving any vertex $v \in P_\pi(i)$ to partition $P_\pi(j)$ $(1 \le i, j \le m)$can be defined to be

$$g_1(v, j) = \sum_{u \in P_\pi(i)} w_3(\{u, v\}) - \sum_{u \in P_\pi(j)} w_3(\{u, v\}),$$

and for any $s \in S_2$, the gain for exchanging any pair of vertices $u \in P_\pi(i)$ and $v \in P_\pi(j)$ $(i \ne j)$ can be defined to be

$$g_2(u,v) = g_1(u,j) + g_1(v,i) + 2w_3(\{u,v\})$$

because we can view the vertex exchange as consisting of two consecutive vertex moves. Since g_2 can be defined in terms of g_1, in the following we only need to consider the incremental update of g_1 after each vertex move.

Let π be the current solution with objective function value $W_3(\pi)$, $s \in S_1$ moves vertex v from its current containing partition to partition j, and π' be the new solution resulting from applying move s to solution π. It follows from the definitions that $W_3(\pi') = W_3(\pi) + g_1(v, j)$. Therefore, the problem of incremental evaluation of objective function $W_3(\pi)$ is now reduced to the problem of incremental evaluation of the gain function $g_1(v, j)$ for all $1 \le j \le m$ and all $v \in \{u \in V \mid \pi(u) \ne j\}$.

4.3 Incremental Gain Function Updating

We maintain the values of function $g_1(v, j)$ in a 2-D table. As long as we have updated values of function $g_1(v, j)$ for all combinations of $1 \le j \le m$ and $v \in \{u \in V \mid \pi(u) \ne j\}$, we can find the new objective function value after a move by just adding its gain to the old objective function value, in constant time. But based on the definition of gain function $g_1()$, it can be verified that after moving vertex $v \in P_\pi(i)$ to partition $P_\pi(j)$ ($i \ne j$), the gain function $g_1()$ can be incrementally updated as follows in $O(|V|)$:

Case 1: $g_1'(v,i) = -g_1(v, j)$

Case 2: $g_1'(v,k) = g_1(v,k) - g_1(v,j), \qquad k \notin \{i,j\}$

Case 3: $g_1'(u,j) = g_1(u,j) + 2w_3(\{u,v\}), \qquad \forall u \in P_\pi(i)$

Case 4: $g_1'(u,k) = g_1(u,k) + w_3(\{u,v\}), \qquad \forall u \in P_\pi(i), k \notin \{i,j\}$

Case 5: $g_1'(u,i) = g_1(u,i) - 2w_3(\{u,v\}), \qquad \forall u \in P_\pi(j)$

Case 6: $g_1'(u,k) = g_1(u,k) - w_3(\{u,v\}), \qquad \forall u \in P_\pi(j), k \notin \{i,j\}$

Case 7: $g_1'(u,i) = g_1(u,i) - w_3(\{u,v\}), \qquad \forall u \notin P_\pi(i) \cup P_\pi(j)$

Case 8: $g_1'(u,j) = g_1(u,j) + w_3(\{u,v\}), \qquad \forall u \notin P_\pi(i) \cup P_\pi(j)$

where $g_1'()$ marks the new value of $g_1()$. The significant speedup of $O(|V|)$, made possible by our methodology for incremental objective function evaluation, can benefit any solution heuristic for this particular problem.

In the following figures we provide one general example for each of the cases for incremental update of the gain function. They can be treated as informal proof for the correctness of this evaluation algorithm. For each of the cases, vertex v of partition i is being moved to another partition j, the left figure shows the partial partitioning just before the move, and the right figure shows the partial partitioning just after the move.

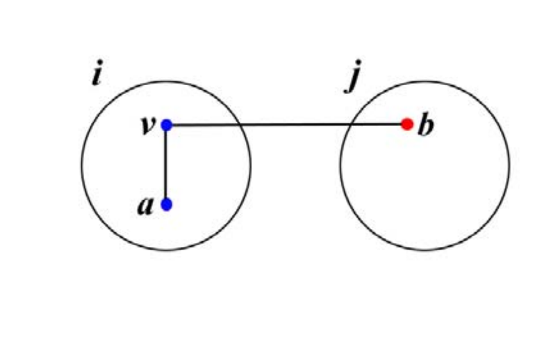
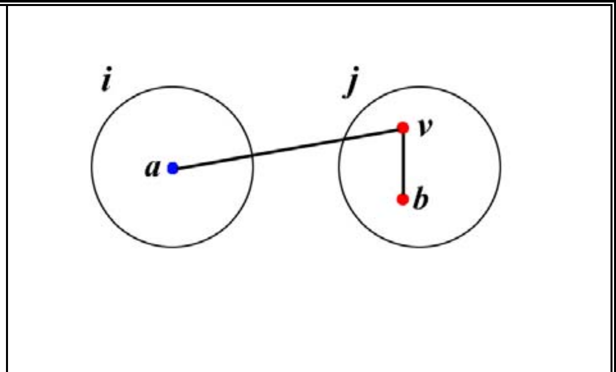
	
$g_1(v,i) = w_3(\{v,b\}) - w_3(\{v,a\})$	$g_1'(v,i) = w_3(\{v,a\}) - w_3(\{v,b\}) = -g_1(v,i)$

Figure 7 Incremental move gain update case 1

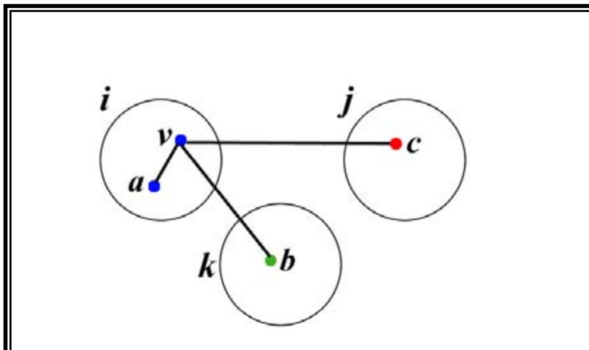
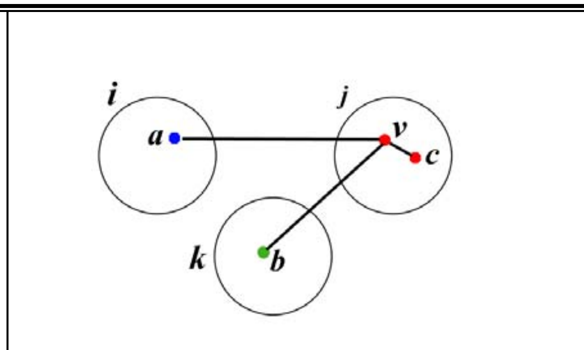
	
$g_1(v,k) = w_3(\{v,b\}) - w_3(\{v,a\})$ $g_1(v,j) = w_3(\{v,c\}) - w_3(\{v,a\})$	$g_1'(v,k) = w_3(\{v,b\}) - w_3(\{v,c\})$ $g_1'(v,k) = g_1(v,k) - g_1(v,j)$

Figure 8 Incremental move gain update case 2

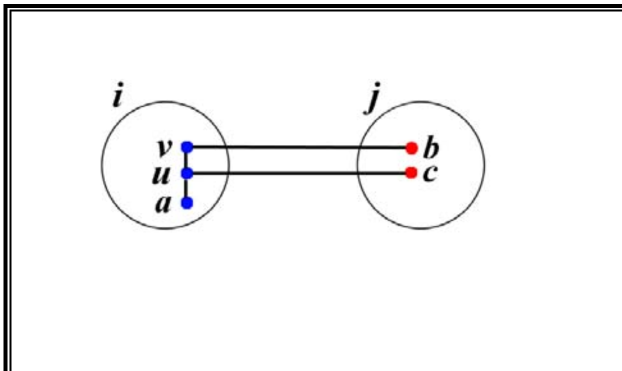
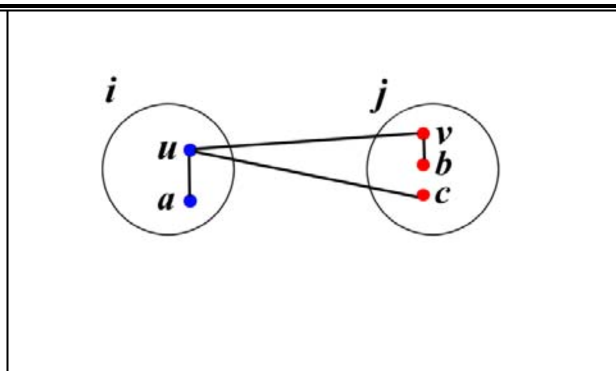
	
$g_1(u,j) = w_3(\{u,c\}) - w_3(\{u,v\}) - w_3(\{u,a\})$	$g_1'(u,j) = w_3(\{u,v\}) + w_3(\{u,c\}) - w_3(\{u,a\})$ $g_1'(u,j) = g_1(u,j) + 2w_3(\{u,v\})$

Figure 9 Incremental move gain update case 3

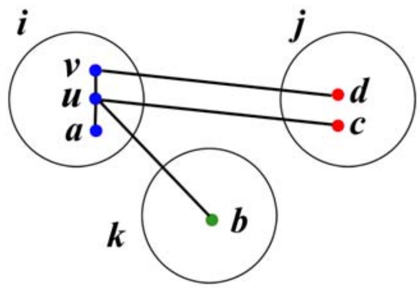
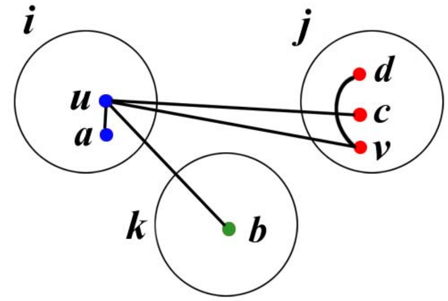
	
$g_1(u,k) = w_3(\{u,b\}) - w_3(\{u,v\}) - w_3(\{u,a\})$	$g_1'(u,k) = w_3(\{u,b\}) - w_3(\{u,a\})$ $g_1'(u,k) = g_1(u,k) + w_3(\{u,v\})$

Figure 10 Incremental move gain update case 4

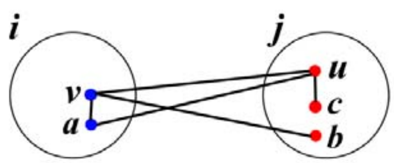
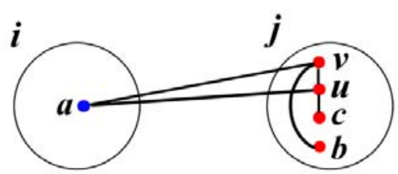
	
$g_1(u,i) = w_3(\{u,v\}) + w_3(\{u,a\}) - w_3(\{u,c\})$	$g_1'(u,i) = w_3(\{u,a\}) - w_3(\{u,v\}) - w_3(\{u,c\})$ $g_1'(u,i) = g_1(u,i) - 2w_3(\{u,v\})$

Figure 11 Incremental move gain update case 5

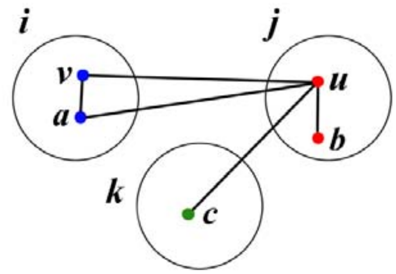
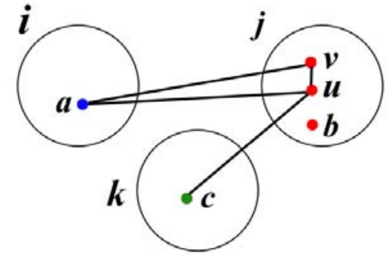
	
$g_1(u,k) = w_3(\{u,c\}) - w_3(\{u,b\})$	$g_1'(u,k) = w_3(\{u,c\}) - w_3(\{u,v\}) - w_3(\{u,b\})$ $g_1'(u,k) = g_1(u,k) - w_3(\{u,v\})$

Figure 12 Incremental move gain update case 6

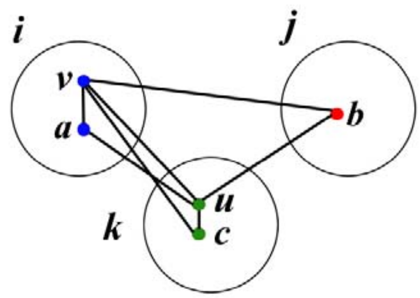
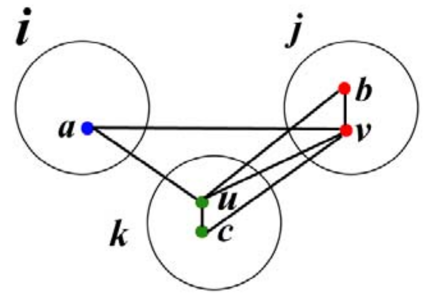
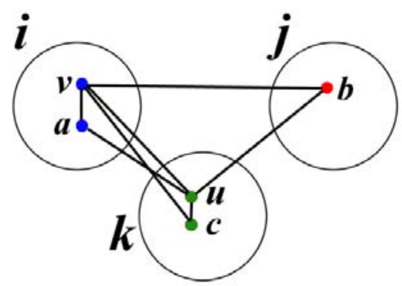
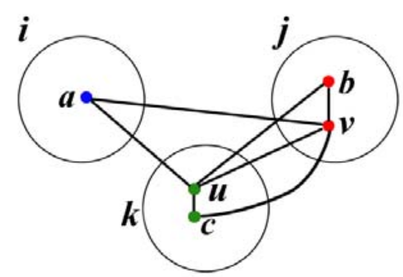
	
$g_1(u,i) = w_3(\{u,a\}) + w_3(\{u,v\}) - w_3(\{u,c\})$	$g_1'(u,i) = w_3(\{u,a\}) - w_3(\{u,c\})$ $g_1'(u,i) = g_1(u,i) - w_3(\{u,v\})$

Figure 13 Incremental move gain update case 7

$g_1(u, j) = w_3(\{u, b\}) - w_3(\{u, c\})$	$g_1'(u, j) = w_3(\{u, b\}) + w_3(\{u, v\}) - w_3(\{u, c\})$ $g_1'(u, j) = g_1(u, j) + w_3(\{u, v\})$

Figure 14 Incremental vove gain update case 8

Chapter 5

Simulated Annealing Algorithm

The design of the simulated annealing algorithm will be explored in this chapter. Sensitivity analysis will be conducted on the multiple parameters of the algorithm to find their best values.

5.1 Algorithm Design

Simulated annealing is a meta-heuristic that attempts to avoid entrapment in poor local optima by allowing occasional downhill moves. Our algorithm for the multi-way graph partitioning problem based on simulated annealing is outlined in Algorithm 5. We just call it the simulated annealing algorithm for convenience. This procedure is performed under the influence of a random number generator and a control parameter called the temperature. As typically implemented, the simulated annealing approach involves a pair of nested loops and two additional parameters, a *cooling ratio* r, which is between *zero* and *one*, and an integer *temperature length* L. The most important of this process is the loop at Step 3.1. Note that $e^{\Delta / T}$ will be a number in the interval (0, 1) when T is positive and Δ is negative, and rightfully can be interpreted as a probability that depends on Δ and T. The probability that a downhill move will be accepted diminishes as the temperature declines, and, for a fixed temperature T, small downhill moves have higher probabilities of acceptance than large ones [12][23]. This particular method of operation is motivated by a physical analogy, best described in terms of the physics of crystal growth [14]. It has been proven that the algorithm will converge to a global optimum if the temperature is lowered exponentially and the initial temperature is chosen sufficiently high [11].

Algorithm 5. Simulated annealing for graph partitioning

1. Get a random initial solution π.
2. Get an initial temperature $T > 0$.
3. While stop criterion not met do the following.
   3.1 Perform the following loop *L* times.
      3.1.1 Let π' be a random neighbor of π.
      3.1.2 Let $\Delta = W_3(\pi') - W_3(\pi)$.
      3.1.3 If $\Delta \geq 0$ (uphill move),
         Set $\pi = \pi'$.
      3.1.4 If $\Delta < 0$ (downhill move),
         Set $\pi = \pi'$ with probability $e^{\Delta/T}$.
   3.2 Set $T = r \cdot T$ (reduce temperature).
4. Return the best π visited.

5.2 Experiment Design for Parameter Tuning

There are four parameters, described below, that must find their optimized values for achieving the best performance of the simulated annealing algorithm. These parameters are inter-related and have major effect on solution quality and algorithm running time.

1. **Initial temperature--t_0:** Simulated annealing algorithms are in general time consuming in their execution. The choice t_0 has a direct effect on the annealing schedule. If t_0 is too high, the algorithm's initial random walking will be

prolonged without benefit. Conversely, if t_0 is too slow, the algorithm will be lead to entrapment in poor local optima.

2. **Temperature reduction ratio--r:** This is also a major factor to affect the execution of the algorithm. Ratio r is a real number in the interval (0, 1). If r is too large, temperature will be reduced vigorously, and the algorithm will be lead to entrapment in poor local optima. If r is too small, algorithm execution will be significantly prolonged.

3. **Number of consecutive non-improvement iterations before the temperature is reduced—l:** This is to control the number of non-improvement iterations before the temperature is reduced. If l is too large, the execution time will be spent on inefficient solution hunting. If l is too small, the solution neighborhoods will not be explored thoroughly.

4. **Number of consecutive non-improvement iterations before algorithm termination--k:** This is to control the number of non-improvement iterations before the termination of the algorithm. If k is too large, the execution time will be increased without profits. If k is too small, the alternative solution neighborhoods will not be explored thoroughly due to rushed termination.

In this research, 50 random problem instances are generated for algorithm performance evaluation. Their numbers of vertices range from 20 to 200, expected degrees of each vertex (number of incident edges) range from 8 to 30, both vertex weights and edge weights range from 1 to 5, and the number of partitions range from 2 to 8.

For parameter tuning in this chapter, we choose the following five problem instances from our 50 problem instances to conduct experiments, one instance for each vertex number. We call the five problem instances our *training set*.

Table 1 Data files for parameter tuning

Data Files	**Vertex Number**	**Expected Degree**	**Vertex Weights (Min-Max)**	**Edge Weights (Min-Max)**
g20d8c0	20	8	1-5	1-5
g40d15c0	40	15	1-5	1-5
g60d20c0	60	20	1-5	1-5
g100d30c0	100	30	1-5	1-5
g200d30c0	200	30	1-5	1-5

All experiments are conducted on a Pentium(R) 4 PC with a 2.53GHz CPU and 512 MB of RAM, running Microsoft Windows XP Professional.

After initial experimental exploration for the training set, we find the following ranges of values for the four parameters are more promising and worth further investigation.

Table 2. Simulated annealing parameter values to be explored

Name	Parameter	Values
Initial Temperature	t_0	5,10,15,20,25,30,35,40,45,50
Cooling rate	r	0.90, 0.95, 0.99, 0.995, 0.9995
Number of consecutive non-improvement Iterations before the temperature is rduced	l	100,200,300,400,500,600,700,800, 900,1000,1100,1200,1300,1400,1500, 1600,1700,1800,1900,2000
Number of consecutive non-improvement iterations before algorithm termination	k	20,40,60,80,100,120,140,160,180,200, 220,240,260,280,300,320,340,360,380,400

The search for optimal parameter values is the most difficult one since the parameters are not independent. Based on the above parameter value ranges, there are $10 \times 5 \times 20 \times 20 = 20{,}000$ different combinations. We use a driver program to systematically generate performance data for all these combinations for our training set, with partition number ranges 2, 4, 6, and 8.

5.3 Parameter Tuning Experiments

We run all the 20,000 parameter value combinations for each of the problem instances in the training set with partition numbers ranging 2, 4, 6 and 8. Table 3 shows the best parameter values for each pair of the problem instances and the partition numbers that maximize $W_3(\pi)$.

Table 3 Best parameter values for each problem instance

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	t_0	r	l	k
g20d8c0	2	514346	0	0	15	0.9	700	80
g20d8c0	4	771487	0	2	20	0.9995	300	160
g20d8c0	6	856902	8	6	10	0.995	1100	200
g20d8c0	8	899611	16	16	5	0.9995	1200	200
g40d15c0	2	6067005	0	0	5	0.99	100	20
g40d15c0	4	9099624	4	0	20	0.995	300	300
g40d15c0	6	10110476	8	7	45	0.9995	500	320
g40d15c0	8	10615837	12	20	20	0.9995	400	80
g60d20c0	2	31271405	1	0	20	0.995	800	20
g60d20c0	4	46907027	3	0	35	0.9995	1300	160
g60d20c0	6	52118889	5	2	40	0.995	1000	40
g60d20c0	8	54724823	7	16	15	0.95	700	140
g100d30c0	2	225610743	0	0	20	0.9995	600	80
g100d30c0	4	338416042	0	1	10	0.95	400	400
g100d30c0	6	376011993	8	1	35	0.99	800	200

g100d30c0	8	394818687	0	9	50	0.95	300	280
g200d30c0	2	1385000130	1	0	35	0.9995	1200	80
g200d30c0	4	2077500109	3	0	20	0.995	500	180
g200d30c0	6	2308327443	9	2	5	0.995	1600	180
g200d30c0	8	2423749881	7	3	50	0.9	1800	380

The following **Table 4** through **Table 9** present partial experiment result data for problem instances g40d15co and g60d20c0, with *m* being 4, from different presentation angles.

Table 4 Parameter values for g40d15c0 sorted by $W_3(\pi)$

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time(ms)**	t_0	r	l	k
g40d15c0	4	9099624	4	0	750	20	0.995	300	300
g40d15c0	4	9099624	4	5	1812	35	0.995	1100	200
g40d15c0	4	9099623	4	5	3782	45	0.9	1500	320
g40d15c0	4	9099623	4	5	3594	40	0.95	1600	280
g40d15c0	4	9099622	4	0	3953	35	0.9995	1800	280
g40d15c0	4	9099622	4	9	1906	10	0.9	900	260
g40d15c0	4	9099621	4	0	406	40	0.995	800	60
g40d15c0	4	9099621	4	0	875	10	0.9	1700	60
g40d15c0	4	9099621	4	10	3328	20	0.99	1800	220
g40d15c0	4	9099621	4	10	4719	5	0.9995	1900	300
g40d15c0	4	9099619	4	6	500	25	0.99	500	120
g40d15c0	4	9099619	4	10	1000	5	0.9	600	200
g40d15c0	4	9099618	4	0	344	20	0.95	100	400
g40d15c0	4	9099618	4	5	2625	35	0.995	1600	200
g40d15c0	4	9099618	4	9	250	5	0.99	100	260
g40d15c0	4	9099618	4	9	1875	30	0.99	1400	160
g40d15c0	4	9099618	4	10	2594	15	0.9	2000	160
g40d15c0	4	9099617	4	5	62	15	0.995	200	40
g40d15c0	4	9099617	4	5	3234	50	0.99	1100	360
g40d15c0	4	9099617	4	9	62	25	0.99	200	40

Table 5 **Parameter values for** g40d15c0 sorted by $W_1(\pi)$ and $W_2(\pi)$

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time(ms)**	t_0	r	l	k
g40d15c0	4	9099624	4	0	750	20	0.995	300	300
g40d15c0	4	9099622	4	0	3953	35	0.9995	1800	280

g40d15c0	4	9099621	4	0	406	40	0.995	800	60
g40d15c0	4	9099621	4	0	875	10	0.9	1700	60
g40d15c0	4	9099618	4	0	344	20	0.95	100	400
g40d15c0	4	9099615	4	0	1969	5	0.9	1200	200
g40d15c0	4	9099615	4	0	2765	50	0.995	900	380
g40d15c0	4	9099613	4	0	3812	30	0.9995	1800	260
g40d15c0	4	9099612	4	0	1297	20	0.99	2000	80
g40d15c0	4	9099612	4	0	2531	5	0.99	800	400
g40d15c0	4	9099610	4	0	4453	45	0.995	1500	380
g40d15c0	4	9099609	4	0	703	35	0.9	1000	80
g40d15c0	4	9099609	4	0	984	10	0.9995	500	240
g40d15c0	4	9099609	4	0	3063	30	0.9995	1000	380
g40d15c0	4	9099608	4	0	94	20	0.995	500	20
g40d15c0	4	9099608	4	0	3765	40	0.9	1700	280
g40d15c0	4	9099606	4	0	3312	10	0.9	1400	280
g40d15c0	4	9099605	4	0	1078	15	0.99	1100	120
g40d15c0	4	9099605	4	0	3578	10	0.9995	1800	240
g40d15c0	4	9099604	4	0	250	5	0.9995	100	300

Table 6 Parameter values for g40d15c0 sorted by running time, $W_1(\pi)$ and $W_2(\pi)$

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time(ms)**	t_0	r	l	k
g40d15c0	4	9099568	4	0	15	30	0.9995	100	20
g40d15c0	4	9099559	4	0	16	40	0.995	100	20
g40d15c0	4	9099559	4	0	16	15	0.95	100	20
g40d15c0	4	9099531	4	0	16	30	0.99	100	20
g40d15c0	4	9099594	4	0	31	45	0.95	100	20
g40d15c0	4	9099580	4	0	31	45	0.9995	200	20
g40d15c0	4	9099564	4	0	31	10	0.9995	100	20
g40d15c0	4	9099559	4	0	47	15	0.995	200	20
g40d15c0	4	9099558	4	0	47	25	0.99	200	20
g40d15c0	4	9099545	4	0	47	35	0.95	200	20
g40d15c0	4	9099540	4	0	47	30	0.9	100	60
g40d15c0	4	9099528	4	0	47	30	0.95	200	20
g40d15c0	4	9099500	4	0	47	25	0.9995	100	40
g40d15c0	4	9099587	4	0	63	5	0.99	100	60
g40d15c0	4	9099558	4	0	63	35	0.95	100	40
g40d15c0	4	9099590	4	0	78	40	0.95	100	60
g40d15c0	4	9099580	4	0	78	5	0.995	500	20
g40d15c0	4	9099567	4	0	78	5	0.99	500	20
g40d15c0	4	9099567	4	0	78	10	0.9	100	80
g40d15c0	4	9099562	4	0	78	35	0.9	200	40

Table 7 Parameter values for g60d20c0 sorted by $W_3(\pi)$

Data File	*m*	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	Time (ms)	t_0	r	l	k
g60d20c0	4	46907027	3	0	4907	35	0.9995	1300	160
g60d20c0	4	46907027	3	2	3578	30	0.95	500	360
g60d20c0	4	46907027	3	2	6422	50	0.9995	1300	200
g60d20c0	4	46907027	3	2	8266	40	0.95	1500	280
g60d20c0	4	46907027	3	7	1703	10	0.995	1600	60
g60d20c0	4	46907027	3	7	6610	50	0.99	1100	220
g60d20c0	4	46907026	3	1	10203	35	0.9995	1300	240
g60d20c0	4	46907026	3	2	1547	20	0.995	800	60
g60d20c0	4	46907026	3	2	2547	45	0.9995	1100	80
g60d20c0	4	46907026	3	7	1562	20	0.9995	700	40
g60d20c0	4	46907026	3	7	10938	10	0.99	2000	280
g60d20c0	4	46907026	3	7	21219	20	0.9995	2000	400
g60d20c0	4	46907025	3	1	985	30	0.995	200	120
g60d20c0	4	46907025	3	1	9735	45	0.9995	1300	340
g60d20c0	4	46907025	3	2	3438	50	0.95	500	300
g60d20c0	4	46907025	3	2	13234	35	0.995	1400	320
g60d20c0	4	46907025	3	7	812	25	0.99	100	280
g60d20c0	4	46907025	3	7	1297	25	0.9995	400	80
g60d20c0	4	46907025	3	7	4734	30	0.95	700	360
g60d20c0	4	46907024	3	0	891	5	0.95	300	160

Table 8 Parameter values for g60d20c0 sorted by $W_1(\pi)$ and $W_2(\pi)$

Data File	*m*	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	Time (ms)	t_0	r	l	k
g60d20c0	4	46907027	3	0	4907	35	0.9995	1300	160
g60d20c0	4	46907024	3	0	891	5	0.95	300	160
g60d20c0	4	46907022	3	0	3860	5	0.995	1600	140
g60d20c0	4	46907020	3	0	1610	35	0.95	700	80
g60d20c0	4	46907018	3	0	656	45	0.99	500	60
g60d20c0	4	46907017	3	0	1000	35	0.99	500	60
g60d20c0	4	46907015	3	0	813	5	0.995	1100	20
g60d20c0	4	46907005	3	0	3844	5	0.99	700	260
g60d20c0	4	46907003	3	0	2188	35	0.9	400	300
g60d20c0	4	46907001	3	0	14141	45	0.9995	1500	200
g60d20c0	4	46907001	3	0	1094	10	0.99	200	280
g60d20c0	4	46906999	3	0	7125	40	0.9995	500	360
g60d20c0	4	46906999	3	0	1203	35	0.995	100	180
g60d20c0	4	46906998	3	0	4297	40	0.99	600	240

g60d20c0	4	46906998	3	0	6422	10	0.9995	1400	180
g60d20c0	4	46906997	3	0	2156	5	0.95	400	320
g60d20c0	4	46906997	3	0	2515	5	0.995	900	140
g60d20c0	4	46906996	3	0	7438	10	0.95	1500	280
g60d20c0	4	46906996	3	0	735	5	0.99	500	80
g60d20c0	4	46906994	3	0	1344	35	0.99	400	100

Table 9 Parameter values for g60d20c0 sorted by Running time, $W_1(\pi)$ and $W_2(\pi)$

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time(ms)**	t_0	r	l	k
g60d20c0	4	46906943	3	0	109	45	0.95	300	20
g60d20c0	4	46906950	3	0	125	50	0.9	100	40
g60d20c0	4	46906913	3	0	125	40	0.995	100	60
g60d20c0	4	46906923	3	0	188	20	0.9995	100	100
g60d20c0	4	46906918	3	0	203	45	0.99	100	60
g60d20c0	4	46906944	3	0	219	50	0.9995	400	20
g60d20c0	4	46906930	3	0	219	50	0.9995	500	20
g60d20c0	4	46906921	3	0	250	35	0.99	200	60
g60d20c0	4	46906926	3	0	281	15	0.9995	100	80
g60d20c0	4	46906920	3	0	281	5	0.95	100	140
g60d20c0	4	46906903	3	0	328	50	0.995	600	20
g60d20c0	4	46906946	3	0	343	45	0.99	300	60
g60d20c0	4	46906993	3	0	359	20	0.95	100	180
g60d20c0	4	46906916	3	0	375	50	0.9995	100	180
g60d20c0	4	46906944	3	0	391	35	0.99	100	100
g60d20c0	4	46906971	3	0	406	15	0.9	200	100
g60d20c0	4	46906931	3	0	422	25	0.9995	900	20
g60d20c0	4	46906910	3	0	422	45	0.9995	900	20
g60d20c0	4	46906906	3	0	422	25	0.95	100	180
g60d20c0	4	46906986	3	0	500	30	0.9	100	260

Compromising solution quality and algorithm running time, we decided to use the following parameter values for our simulated annealing algorithm for all the 50 problem instances for performance evaluation. These fixed set of parameter values will be used in the next chapter to compare our simulated annealing algorithm with repeated random solutions and local optimization.

Table 10 Adopted parameter values for simulated annealing algorithm

t_0	r	l	k
20	0.9995	400	80

Chapter 6

Comparative Study

For combinatorial optimization problems like graph partitioning, comparative study of algorithms solving the same problem is fundamental to evaluating the algorithm quality. In this chapter, we design experiments to compare the solution quality and running time for simulated annealing, local optimization and repeat random algorithms.

6.1 Experiment Design

In this research, 50 random problem instances are generated for algorithm performance evaluation. Their numbers of vertices range from 20 to 200, expected degrees of each vertex (number of incident edges) range from 8 to 30, both vertex weights and edge weights range from 1 to 5, and the number of partitions range from 2 to 8.

All experiments are conducted on a Pentium(R) 4 PC with a 2.53GHz CPU and 512 MB of RAM, running Microsoft Windows XP Professional.

We run simulated annealing with selective parameter values. The running time will be generated from 50 problem instances individually. With the same time basis, the reference algorithms could adopt it for comparability.

The repeat random algorithm and the local optimization algorithm will be used as reference algorithms to solving the multi-way graph partitioning problem. For the repeated random algorithm, random solutions will be generated for as long as its competitor for each problem instance, and report the best solution found.

The parameter values for simulated annealing are selected in Chapter 5: t_0=20, r=0.9995, l=400 and k = 80.

6.2 Solution Quality of Simulated Annealing

This section reports solution quality and running time of our simulated annealing algorithm for each of the 50 benchmark problem instances, with the partition number ranging 2, 4, 6, and 8.

Table 11 Simulated annealing performance for *m*=2

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time (ms)**
g20d8c0	2	514325	0	0	94
g20d8c1	2	413319	1	0	78
g20d8c2	2	437130	0	0	78
g20d8c3	2	417943	1	0	78
g20d8c4	2	524557	0	0	78
g20d8c5	2	397501	1	0	109
g20d8c6	2	446573	1	0	78
g20d8c7	2	400651	0	0	78
g20d8c8	2	443486	1	0	78
g20d8c9	2	458351	1	0	94
g40d15c0	2	6066948	0	0	250
g40d15c1	2	8854484	0	0	265
g40d15c2	2	5338715	0	0	250
g40d15c3	2	5741993	1	0	297
g40d15c4	2	6515626	1	0	297
g40d15c5	2	6245558	0	0	281
g40d15c6	2	6132098	0	0	265
g40d15c7	2	6864784	0	0	282
g40d15c8	2	7047044	0	0	250
g40d15c9	2	7500066	1	0	265
g60d20c0	2	31271320	1	0	687
g60d20c1	2	33031198	1	0	750
g60d20c2	2	28902854	0	0	547
g60d20c3	2	28974294	0	0	671

g60d20c4	2	33351726	0	0	563
g60d20c5	2	31094151	0	5	546
g60d20c6	2	29845372	1	0	766
g60d20c7	2	29834714	0	0	562
g60d20c8	2	28687775	1	0	719
g60d20c9	2	27935910	1	0	812
g100d30c0	2	225610506	0	0	1563
g100d30c1	2	206073682	1	0	1859
g100d30c2	2	163223962	1	0	2687
g100d30c3	2	203382372	0	0	2781
g100d30c4	2	205191680	1	0	3140
g100d30c5	2	231988111	0	0	1563
g100d30c6	2	176986562	1	0	1531
g100d30c7	2	174884899	1	0	2234
g100d30c8	2	219606111	0	0	1516
g100d30c9	2	206095952	0	0	1562
g200d30c0	2	1384999918	1	0	7515
g200d30c1	2	1705949627	0	0	6344
g200d30c2	2	1751191478	0	0	6453
g200d30c3	2	1647418644	1	0	11531
g200d30c4	2	1522644568	1	0	8235
g200d30c5	2	1628774697	0	0	6312
g200d30c6	2	1735164179	0	0	6546
g200d30c7	2	1705365555	0	0	6547
g200d30c8	2	1542222944	1	0	15391
g200d30c9	2	1619136838	0	0	6468

Table 12 Simulated annealing performance for *m*=4

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time (ms)**
g20d8c0	4	771450	0	4	110
g20d8c1	4	619996	3	2	125
g20d8c2	4	655714	0	0	109
g20d8c3	4	626903	3	4	109
g20d8c4	4	786843	0	11	94
g20d8c5	4	596222	3	2	93
g20d8c6	4	669863	3	0	110
g20d8c7	4	600757	4	3	94
g20d8c8	4	665259	3	3	94
g20d8c9	4	687549	3	5	125
g40d15c0	4	9099584	4	5	250

g40d15c1	4	13281707	0	4	329
g40d15c2	4	8007178	4	5	297
g40d15c3	4	8612991	3	0	406
g40d15c4	4	9773502	3	0	344
g40d15c5	4	9368366	0	5	266
g40d15c6	4	9198164	0	3	266
g40d15c7	4	10296274	4	0	390
g40d15c8	4	10570565	0	0	328
g40d15c9	4	11250049	3	5	391
g60d20c0	4	46906932	3	1	1094
g60d20c1	4	49546762	3	0	1437
g60d20c2	4	43352454	4	1	813
g60d20c3	4	43459798	4	1	625
g60d20c4	4	50027679	0	0	844
g60d20c5	4	46639526	4	7	578
g60d20c6	4	44767946	3	3	1032
g60d20c7	4	44752097	0	0	563
g60d20c8	4	43031562	3	7	2125
g60d20c9	4	41903789	3	0	1594
g100d30c0	4	338415957	0	1	1484
g100d30c1	4	309110542	3	0	2266
g100d30c2	4	244835849	3	8	2484
g100d30c3	4	305073479	0	0	1469
g100d30c4	4	307787433	3	0	3063
g100d30c5	4	347982191	0	6	1469
g100d30c6	4	265479793	3	2	3640
g100d30c7	4	262327273	3	0	3250
g100d30c8	4	329409214	0	0	1500
g100d30c9	4	309139572	4	0	2578
g200d30c0	4	2077499709	3	1	9703
g200d30c1	4	2558924617	0	0	5875
g200d30c2	4	2626778410	4	0	13188
g200d30c3	4	2471128183	3	3	10672
g200d30c4	4	2283966643	3	0	30984
g200d30c5	4	2443152900	4	0	8812
g200d30c6	4	2602746182	0	0	5797
g200d30c7	4	2558048394	0	3	7390
g200d30c8	4	2313334399	3	0	9922
g200d30c9	4	2428696598	4	0	10203

Table 13 Simulated annealing performance for *m*=6

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time (ms)**
g20d8c0	6	856891	8	5	109
g20d8c1	6	688874	5	10	94
g20d8c2	6	728285	8	8	110
g20d8c3	6	696537	5	18	78
g20d8c4	6	874254	0	15	109
g20d8c5	6	662473	5	15	94
g20d8c6	6	744313	5	2	93
g20d8c7	6	667450	8	11	94
g20d8c8	6	739159	5	7	78
g20d8c9	6	763921	5	13	79
g40d15c0	6	10110433	8	10	547
g40d15c1	6	14756220	8	8	422
g40d15c2	6	8896624	8	18	250
g40d15c3	6	9569962	5	22	391
g40d15c4	6	10859451	5	10	281
g40d15c5	6	10409292	0	6	250
g40d15c6	6	10219025	8	3	407
g40d15c7	6	11440119	8	4	344
g40d15c8	6	11743724	8	9	562
g40d15c9	6	12499477	9	18	359
g60d20c0	6	52118856	5	6	1032
g60d20c1	6	55050712	9	6	1594
g60d20c2	6	48168986	8	11	610
g60d20c3	6	48288281	8	5	1094
g60d20c4	6	55586359	0	5	625
g60d20c5	6	51821233	8	23	1015
g60d20c6	6	49740977	9	6	829
g60d20c7	6	49722211	8	16	812
g60d20c8	6	47811720	9	21	781
g60d20c9	6	46559751	5	6	1250
g100d30c0	6	376011828	8	5	2953
g100d30c1	6	343452935	9	13	3016
g100d30c2	6	272039791	5	24	2594
g100d30c3	6	338964736	8	10	1375
g100d30c4	6	341982896	9	8	1578
g100d30c5	6	386640729	8	2	3907
g100d30c6	6	294974452	9	4	2968
g100d30c7	6	291471702	9	10	1657
g100d30c8	6	366004348	8	9	1609

g100d30c9	6	343487342	8	2	1516
g200d30c0	6	2308326999	9	6	5844
g200d30c1	6	2843249599	0	3	6531
g200d30c2	6	2918640546	8	0	12813
g200d30c3	6	2745697834	5	2	11391
g200d30c4	6	2537740634	5	8	9610
g200d30c5	6	2714624312	0	10	5343
g200d30c6	6	2891940251	0	3	5375
g200d30c7	6	2842264270	8	8	7719
g200d30c8	6	2570371315	5	5	9157
g200d30c9	6	2698549725	8	7	8922

Table 14 Simulated annealing performance for *m*=8

Data File	***m***	$W_3(\pi)$	$W_1(\pi)$	$W_2(\pi)$	**Time (ms)**
g20d8c0	8	899601	16	16	94
g20d8c1	8	723029	15	38	78
g20d8c2	8	764992	0	25	78
g20d8c3	8	731382	7	30	78
g20d8c4	8	917384	16	53	78
g20d8c5	8	695368	15	20	78
g20d8c6	8	781104	19	21	109
g20d8c7	8	700801	12	30	78
g20d8c8	8	775862	15	20	94
g20d8c9	8	801866	15	29	93
g40d15c0	8	10615837	12	20	250
g40d15c1	8	15493444	14	23	235
g40d15c2	8	9341391	12	39	406
g40d15c3	8	10048468	7	31	266
g40d15c4	8	11401395	15	19	234
g40d15c5	8	10929718	0	24	234
g40d15c6	8	10729409	16	14	360
g40d15c7	8	12012031	12	24	422
g40d15c8	8	12330287	16	35	359
g40d15c9	8	13125054	7	41	250
g60d20c0	8	54724782	7	19	921
g60d20c1	8	57802669	15	19	2703
g60d20c2	8	50577252	12	15	484
g60d20c3	8	50702496	12	9	1125
g60d20c4	8	58365606	0	4	484

g60d20c5	8	54412076	12	23	687
g60d20c6	8	52229256	7	12	1172
g60d20c7	8	52210763	0	21	485
g60d20c8	8	50203498	7	35	1297
g60d20c9	8	48887739	7	18	1078
g100d30c0	8	394818502	0	15	1250
g100d30c1	8	360628853	7	30	2235
g100d30c2	8	285637264	15	33	2890
g100d30c3	8	355919084	0	25	1328
g100d30c4	8	359085238	7	13	2093
g100d30c5	8	405970040	16	4	3078
g100d30c6	8	309726273	7	22	2562
g100d30c7	8	306048457	7	23	3078
g100d30c8	8	384301950	16	13	3250
g100d30c9	8	360661223	12	17	1344
g200d30c0	8	2423749569	7	6	9094
g200d30c1	8	2985393923	16	14	15594
g200d30c2	8	3064571656	12	2	8188
g200d30c3	8	2882982787	7	2	8000
g200d30c4	8	2664618549	15	18	9453
g200d30c5	8	2850341735	12	9	11312
g200d30c6	8	3036518744	16	11	7954
g200d30c7	8	2984389893	0	12	4921
g200d30c8	8	2698880766	15	5	6406
g200d30c9	8	2833476281	12	14	7922

6.3 Comparisons with Repeat Random Solutions

For each combination of problem instances and partition numbers, we run Repeated Random for the same amount of time as our simulated annealing algorithm and compare the resulting solution quality for Repeated Random with that for simulated annealing. The resulting data are reported in **Table 15**. In this table, RR denotes Random Repeat, SA denotes Simulated Annealing, and Diff% is the difference between the summation of $W_1(\pi)$ and $W_2(\pi)$ for Repeated Random and the summation of $W_1(\pi)$ and $W_2(\pi)$ for simulated annealing, divided by the latter summation. In the last Result column, SA=RR means that SA and RR have the same solution quality, and SA>RR means that SA outperforms RR.

Table 15 Performance comparison between RR and SA (m=2, 4, 6, 8)

Data Files	m	$W_3(\pi)$	$W_1(\pi)$ RR	$W_2(\pi)$ RR	$W_1(\pi)$ SA	$W_2(\pi)$ SA	$W_1(\pi)$ Diff.	$W_2(\pi)$ Diff.	Diff. %	Result
g20d8c0	2	514328	0	0	0	0	0	0	0.00%	SA=RR
g20d8c1	2	413352	1	0	1	0	0	0	0.00%	SA=RR
g20d8c2	2	437165	0	0	0	0	0	0	0.00%	SA=RR
g20d8c3	2	417969	1	0	1	0	0	0	0.00%	SA=RR
g20d8c4	2	524580	0	0	0	0	0	0	0.00%	SA=RR
g20d8c5	2	397510	1	0	1	0	0	0	0.00%	SA=RR
g20d8c6	2	446607	1	0	1	0	0	0	0.00%	SA=RR
g20d8c7	2	400663	0	0	0	0	0	0	0.00%	SA=RR
g20d8c8	2	443523	1	0	1	0	0	0	0.00%	SA=RR
g20d8c9	2	458382	1	1	1	0	0	1	50.00%	SA>RR
g40d15c0	2	6067000	0	0	0	0	0	0	0.00%	SA=RR
g40d15c1	2	8854525	0	0	0	0	0	0	0.00%	SA=RR
g40d15c2	2	5338709	0	5	0	0	0	5	100.00%	SA=RR
g40d15c3	2	5742002	1	0	1	0	0	0	0.00%	SA=RR
g40d15c4	2	6515679	1	0	1	0	0	0	0.00%	SA=RR
g40d15c5	2	6245604	0	0	0	0	0	0	0.00%	SA=RR
g40d15c6	2	6132178	0	0	0	0	0	0	0.00%	SA=RR
g40d15c7	2	6864822	0	0	0	0	0	0	0.00%	SA=RR
g40d15c8	2	7047096	0	0	0	0	0	0	0.00%	SA=RR
g40d15c9	2	7500090	1	0	1	0	0	0	0.00%	SA=RR
g60d20c0	2	31271366	1	1	1	0	0	1	50.00%	SA>RR
g60d20c1	2	33031255	1	0	1	0	0	0	0.00%	SA=RR
g60d20c2	2	28902854	0	0	0	0	0	0	0.00%	SA=RR
g60d20c3	2	28974425	0	0	0	0	0	0	0.00%	SA=RR
g60d20c4	2	33351852	0	0	0	0	0	0	0.00%	SA=RR
g60d20c5	2	31094276	0	5	0	5	0	0	0.00%	SA=RR
g60d20c6	2	29845363	1	3	1	0	0	3	75.00%	SA>RR
g60d20c7	2	29834768	0	0	0	0	0	0	0.00%	SA=RR
g60d20c8	2	28687760	1	0	1	0	0	0	0.00%	SA=RR
g60d20c9	2	27935895	1	0	1	0	0	0	0.00%	SA=RR
g100d30c0	2	225610683	0	0	0	0	0	0	0.00%	SA=RR
g100d30c1	2	206073734	1	0	1	0	0	0	0.00%	SA=RR
g100d30c2	2	163223939	1	3	1	0	0	3	75.00%	SA>RR
g100d30c3	2	203382396	0	0	0	0	0	0	0.00%	SA=RR
g100d30c4	2	205191662	1	0	1	0	0	0	0.00%	SA=RR

g100d30c5	2	231988261	0	0	0	0	0	0	0.00%	SA=RR
g100d30c6	2	176986509	1	0	1	0	0	0	0.00%	SA=RR
g100d30c7	2	174884885	1	0	1	0	0	0	0.00%	SA=RR
g100d30c8	2	219606207	0	0	0	0	0	0	0.00%	SA=RR
g100d30c9	2	206096042	0	0	0	0	0	0	0.00%	SA=RR
g200d30c0	2	1384999812	1	1	1	0	0	1	50.00%	SA>RR
g200d30c1	2	1705949878	0	0	0	0	0	0	0.00%	SA=RR
g200d30c2	2	1751191510	0	0	0	0	0	0	0.00%	SA=RR
g200d30c3	2	1647418744	1	0	1	0	0	0	0.00%	SA=RR
g200d30c4	2	1522644433	1	0	1	0	0	0	0.00%	SA=RR
g200d30c5	2	1628774676	0	0	0	0	0	0	0.00%	SA=RR
g200d30c6	2	1735164215	0	0	0	0	0	0	0.00%	SA=RR
g200d30c7	2	1705365752	0	0	0	0	0	0	0.00%	SA=RR
g200d30c8	2	1542222935	1	0	1	0	0	0	0.00%	SA=RR
g200d30c9	2	1619137070	0	0	0	0	0	0	0.00%	SA=RR
g20d8c0	4	771015	6	6	0	4	6	2	66.67%	SA>RR
g20d8c1	4	619982	3	3	3	2	0	1	16.67%	SA>RR
g20d8c2	4	655271	6	4	0	0	6	4	100.00%	SA=RR
g20d8c3	4	626884	3	9	3	4	0	5	41.67%	SA>RR
g20d8c4	4	786250	6	9	0	11	6	-2	26.67%	SA>RR
g20d8c5	4	596205	3	10	3	2	0	8	61.54%	SA>RR
g20d8c6	4	669461	7	2	3	0	4	2	66.67%	SA>RR
g20d8c7	4	600727	4	3	4	3	0	0	0.00%	SA=RR
g20d8c8	4	665213	3	3	3	3	0	0	0.00%	SA=RR
g20d8c9	4	687015	7	5	3	5	4	0	33.33%	SA>RR
g40d15c0	4	9096036	10	9	4	5	6	4	52.63%	SA>RR
g40d15c1	4	13275947	10	4	0	4	10	0	71.43%	SA>RR
g40d15c2	4	8005348	6	5	4	5	2	0	18.18%	SA>RR
g40d15c3	4	8611129	7	7	3	0	4	7	78.57%	SA>RR
g40d15c4	4	9771437	7	1	3	0	4	1	62.50%	SA>RR
g40d15c5	4	9366611	6	5	0	5	6	0	54.55%	SA>RR
g40d15c6	4	9196337	6	3	0	3	6	0	66.67%	SA>RR
g40d15c7	4	10292562	10	1	4	0	6	1	63.64%	SA>RR
g40d15c8	4	10570599	0	5	0	0	0	5	100.00%	SA>RR
g40d15c9	4	11246399	9	13	3	5	6	8	63.64%	SA>RR
g60d20c0	4	46906901	3	7	3	1	0	6	60.00%	SA>RR
g60d20c1	4	49539286	9	0	3	0	6	0	66.67%	SA>RR
g60d20c2	4	43352399	4	6	4	1	0	5	50.00%	SA>RR
g60d20c3	4	43452777	10	4	4	1	6	3	64.29%	SA>RR
g60d20c4	4	50024025	6	1	0	0	6	1	100.00%	SA>RR
g60d20c5	4	46631960	10	7	4	7	6	0	35.29%	SA>RR
g60d20c6	4	44757216	11	4	3	3	8	1	60.00%	SA>RR

g60d20c7	4	44748545	6	0	0	0	6	0	100.00%	SA>RR
g60d20c8	4	43027921	7	8	3	7	4	1	33.33%	SA>RR
g60d20c9	4	41892789	9	0	3	0	6	0	66.67%	SA>RR
g100d30c0	4	338319002	18	6	0	1	18	5	95.83%	SA>RR
g100d30c1	4	309091721	9	4	3	0	6	4	76.92%	SA>RR
g100d30c2	4	244826717	7	9	3	8	4	1	31.25%	SA>RR
g100d30c3	4	305064573	6	5	0	0	6	5	100.00%	SA>RR
g100d30c4	4	307759365	11	3	3	0	8	3	78.57%	SA>RR
g100d30c5	4	347954278	10	6	0	6	10	0	62.50%	SA>RR
g100d30c6	4	265470529	7	7	3	2	4	5	64.29%	SA>RR
g100d30c7	4	262309230	9	5	3	0	6	5	78.57%	SA>RR
g100d30c8	4	329382809	10	2	0	0	10	2	100.00%	SA>RR
g100d30c9	4	309139393	4	4	4	0	0	4	50.00%	SA>RR
g200d30c0	4	2077481903	7	1	3	1	4	0	50.00%	SA>RR
g200d30c1	4	2558796967	16	0	0	0	16	0	100.00%	SA>RR
g200d30c2	4	2626617110	18	0	4	0	14	0	77.78%	SA>RR
g200d30c3	4	2471056278	13	2	3	3	10	-1	60.00%	SA>RR
g200d30c4	4	2283930334	9	5	3	0	6	5	78.57%	SA>RR
g200d30c5	4	2443004976	16	0	4	0	12	0	75.00%	SA>RR
g200d30c6	4	2602653404	14	2	0	0	14	2	100.00%	SA>RR
g200d30c7	4	2557994544	10	0	0	3	10	-3	70.00%	SA>RR
g200d30c8	4	2313206084	17	2	3	0	14	2	84.21%	SA>RR
g200d30c9	4	2428660203	10	3	4	0	6	3	69.23%	SA>RR
g20d8c0	6	854652	26	14	8	5	18	9	67.50%	SA>RR
g20d8c1	6	685335	23	18	5	10	18	8	63.41%	SA>RR
g20d8c2	6	727002	22	12	8	8	14	4	52.94%	SA>RR
g20d8c3	6	693793	27	24	5	18	22	6	54.90%	SA>RR
g20d8c4	6	872499	18	23	0	15	18	8	63.41%	SA>RR
g20d8c5	6	661990	13	14	5	15	8	-1	25.93%	SA>RR
g20d8c6	6	742169	27	7	5	2	22	5	79.41%	SA>RR
g20d8c7	6	665769	24	19	8	11	16	8	55.81%	SA>RR
g20d8c8	6	738663	13	14	5	7	8	7	55.56%	SA>RR
g20d8c9	6	760241	31	23	5	13	26	10	66.67%	SA>RR
g40d15c0	6	10105170	22	15	8	10	14	5	51.35%	SA>RR
g40d15c1	6	14738938	36	8	8	8	28	0	63.64%	SA>RR
g40d15c2	6	8884282	30	13	8	18	22	-5	39.53%	SA>RR
g40d15c3	6	9553736	29	21	5	22	24	-1	46.00%	SA>RR
g40d15c4	6	10853479	21	11	5	10	16	1	53.13%	SA>RR
g40d15c5	6	10398861	24	6	0	6	24	0	80.00%	SA>RR
g40d15c6	6	10208030	30	8	8	3	22	5	71.05%	SA>RR
g40d15c7	6	11434549	22	12	8	4	14	8	64.71%	SA>RR
g40d15c8	6	11731111	28	18	8	9	20	9	63.04%	SA>RR

g40d15c9	6	12488601	29	22	9	18	20	4	47.06%	SA>RR
g60d20c0	6	52082207	35	15	5	6	30	9	78.00%	SA>RR
g60d20c1	6	54998878	43	6	9	6	34	0	69.39%	SA>RR
g60d20c2	6	48106890	46	19	8	11	38	8	70.77%	SA>RR
g60d20c3	6	48256795	36	9	8	5	28	4	71.11%	SA>RR
g60d20c4	6	55553771	34	6	0	5	34	1	87.50%	SA>RR
g60d20c5	6	51779847	38	15	8	23	30	-8	41.51%	SA>RR
g60d20c6	6	49719563	29	8	9	6	20	2	59.46%	SA>RR
g60d20c7	6	49701035	28	8	8	16	20	-8	33.33%	SA>RR
g60d20c8	6	47752978	47	20	9	21	38	-1	55.22%	SA>RR
g60d20c9	6	46523199	37	8	5	6	32	2	75.56%	SA>RR
g100d30c0	6	375923630	36	16	8	5	28	11	75.00%	SA>RR
g100d30c1	6	343312727	43	7	9	13	34	-6	56.00%	SA>RR
g100d30c2	6	271931353	37	26	5	24	32	2	53.97%	SA>RR
g100d30c3	6	338726979	60	14	8	10	52	4	75.68%	SA>RR
g100d30c4	6	341889733	37	7	9	8	28	-1	61.36%	SA>RR
g100d30c5	6	386482631	46	7	8	2	38	5	81.13%	SA>RR
g100d30c6	6	294846942	41	18	9	4	32	14	77.97%	SA>RR
g100d30c7	6	291363782	41	14	9	10	32	4	65.45%	SA>RR
g100d30c8	6	365793178	56	11	8	9	48	2	74.63%	SA>RR
g100d30c9	6	343324512	48	9	8	2	40	7	82.46%	SA>RR
g200d30c0	6	2307763475	67	7	9	6	58	1	79.73%	SA>RR
g200d30c1	6	2842775723	58	2	0	3	58	-1	95.00%	SA>RR
g200d30c2	6	2918247153	52	5	8	0	44	5	85.96%	SA>RR
g200d30c3	6	2744892799	79	2	5	2	74	0	91.36%	SA>RR
g200d30c4	6	2537415706	47	4	5	8	42	-4	74.51%	SA>RR
g200d30c5	6	2714218014	52	10	0	10	52	0	83.87%	SA>RR
g200d30c6	6	2891365601	64	3	0	3	64	0	95.52%	SA>RR
g200d30c7	6	2841509116	76	6	8	8	68	-2	80.49%	SA>RR
g200d30c8	6	2569969149	53	7	5	5	48	2	83.33%	SA>RR
g200d30c9	6	2698169272	52	5	8	7	44	-2	73.68%	SA>RR
g20d8c0	8	896484	40	25	16	16	24	9	50.77%	SA>RR
g20d8c1	8	718304	53	36	15	38	38	-2	40.45%	SA>RR
g20d8c2	8	761992	40	27	0	25	40	2	62.69%	SA>RR
g20d8c3	8	728607	39	34	7	30	32	4	49.32%	SA>RR
g20d8c4	8	912136	56	58	16	53	40	5	39.47%	SA>RR
g20d8c5	8	690328	55	29	15	20	40	9	58.33%	SA>RR
g20d8c6	8	777701	53	24	19	21	34	3	48.05%	SA>RR
g20d8c7	8	698700	38	35	12	30	26	5	42.47%	SA>RR
g20d8c8	8	773482	41	23	15	20	26	3	45.31%	SA>RR
g20d8c9	8	798156	49	42	15	29	34	13	51.65%	SA>RR
g40d15c0	8	10584441	76	25	12	20	64	5	68.32%	SA>RR

g40d15c1	8	15483859	44	19	14	23	30	-4	41.27%	SA>RR
g40d15c2	8	9320189	60	38	12	39	48	-1	47.96%	SA>RR
g40d15c3	8	10023247	63	35	7	31	56	4	61.22%	SA>RR
g40d15c4	8	11385588	53	33	15	19	38	14	60.47%	SA>RR
g40d15c5	8	10900243	74	20	0	24	74	-4	74.47%	SA>RR
g40d15c6	8	10714795	54	22	16	14	38	8	60.53%	SA>RR
g40d15c7	8	11995353	54	27	12	24	42	3	55.56%	SA>RR
g40d15c8	8	12286252	82	36	16	35	66	1	56.78%	SA>RR
g40d15c9	8	13107008	57	38	7	41	50	-3	49.47%	SA>RR
g60d20c0	8	54647925	83	32	7	19	76	13	77.39%	SA>RR
g60d20c1	8	57739726	73	25	15	19	58	6	65.31%	SA>RR
g60d20c2	8	50478646	92	31	12	15	80	16	78.05%	SA>RR
g60d20c3	8	50593990	98	13	12	9	86	4	81.08%	SA>RR
g60d20c4	8	58282416	86	7	0	4	86	3	95.70%	SA>RR
g60d20c5	8	54325669	86	26	12	23	74	3	68.75%	SA>RR
g60d20c6	8	52143650	85	14	7	12	78	2	80.81%	SA>RR
g60d20c7	8	52150807	72	18	0	21	72	-3	76.67%	SA>RR
g60d20c8	8	50045973	119	27	7	35	112	-8	71.23%	SA>RR
g60d20c9	8	48836582	67	23	7	18	60	5	72.22%	SA>RR
g100d30c0	8	394668642	72	16	0	15	72	1	82.95%	SA>RR
g100d30c1	8	360469882	75	28	7	30	68	-2	64.08%	SA>RR
g100d30c2	8	285303243	105	32	15	33	90	-1	64.96%	SA>RR
g100d30c3	8	355628475	102	22	0	25	102	-3	79.84%	SA>RR
g100d30c4	8	358647852	125	14	7	13	118	1	85.61%	SA>RR
g100d30c5	8	405700422	98	9	16	4	82	5	81.31%	SA>RR
g100d30c6	8	309280568	125	26	7	22	118	4	80.79%	SA>RR
g100d30c7	8	305805603	85	23	7	23	78	0	72.22%	SA>RR
g100d30c8	8	384046751	94	12	16	13	78	-1	72.64%	SA>RR
g100d30c9	8	360308613	108	18	12	17	96	1	76.98%	SA>RR
g200d30c0	8	2422869176	127	11	7	6	120	5	90.58%	SA>RR
g200d30c1	8	2983134602	196	12	16	14	180	-2	85.58%	SA>RR
g200d30c2	8	3063123661	156	2	12	2	144	0	91.14%	SA>RR
g200d30c3	8	2880460853	215	7	7	2	208	5	95.95%	SA>RR
g200d30c4	8	2663932791	111	17	15	18	96	-1	74.22%	SA>RR
g200d30c5	8	2849492245	114	14	12	9	102	5	83.59%	SA>RR
g200d30c6	8	3035017644	164	10	16	11	148	-1	84.48%	SA>RR
g200d30c7	8	2982717938	174	15	0	12	174	3	93.65%	SA>RR
g200d30c8	8	2698350513	95	7	15	5	80	2	80.39%	SA>RR
g200d30c9	8	2832733666	116	16	12	14	104	2	80.30%	SA>RR

From **Table 15** we can conclude that simulated annealing is better than Repeat Random when the partition number is large and the vertex number is larger. For graph bisection (partition number = 2), simulated annealing and Repeat Random generate similar results. In terms of the summation of $W_1(\pi)$ and $W_2(\pi)$, simulated annealing outperforms Repeat Random by 16.67% to 100%.

6.4 Comparisons with Local Optimization

For each combination of problem instances and partition numbers, we run Local Optimization and our simulated annealing algorithm and compare the resulting solution quality for Local Optimization with that for simulated annealing. The resulting data are reported in **Table 16**. In this table, LO denotes Local Optimization, SA denotes Simulated Annealing, and Diff% is the difference between the summation of $W_1(\pi)$ and $W_2(\pi)$ for Local Optimization and the summation of $W_1(\pi)$ and $W_2(\pi)$ for simulated annealing, divided by the latter summation. In the last Result column, SA=LO means that SA and LO have the same solution quality, and SA>LO means that SA outperforms LO.

Table 16 Performance comparison between LO and SA (*m*=2, 4, 6, 8)

Data Files	*m*	$W_3(\pi)$ LO	$W_1(\pi)$ LO	$W_2(\pi)$ LO	$W_1(\pi)$ SA	$W_2(\pi)$ SA	$W_1(\pi)$ Diff.	$W_2(\pi)$ Diff.	Diff. %	Result
g20d8c0	2	514334	0	0	0	0	0	0	0.00%	SA=LO
g20d8c1	2	413341	1	0	1	0	0	0	0.00%	SA=LO
g20d8c2	2	437151	0	0	0	0	0	0	0.00%	SA=LO
g20d8c3	2	417955	1	0	1	0	0	0	0.00%	SA=LO
g20d8c4	2	524575	0	0	0	0	0	0	0.00%	SA=LO
g20d8c5	2	397498	1	5	1	0	0	5	83.33%	SA>LO
g20d8c6	2	446601	1	0	1	0	0	0	0.00%	SA=LO
g20d8c7	2	400641	0	0	0	0	0	0	0.00%	SA=LO
g20d8c8	2	443508	1	0	1	0	0	0	0.00%	SA=LO
g20d8c9	2	458365	1	1	1	0	0	1	50.00%	SA>LO
g40d15c0	2	6066969	0	1	0	0	0	1	100.00%	SA>LO
g40d15c1	2	8854514	0	0	0	0	0	0	0.00%	SA=LO
g40d15c2	2	5338716	0	5	0	0	0	5	100.00%	SA>LO
g40d15c3	2	5742026	1	0	1	0	0	0	0.00%	SA=LO

g40d15c4	2	6515694	1	0	1	0	0	0	0.00%	SA=LO
g40d15c5	2	6245600	0	2	0	0	0	2	100.00%	SA>LO
g40d15c6	2	6132147	0	3	0	0	0	3	100.00%	SA>LO
g40d15c7	2	6864803	0	0	0	0	0	0	0.00%	SA=LO
g40d15c8	2	7047078	0	0	0	0	0	0	0.00%	SA=LO
g40d15c9	2	7500013	1	4	1	0	0	4	80.00%	SA>LO
g60d20c0	2	31271250	1	1	1	0	0	1	50.00%	SA>LO
g60d20c1	2	33031146	1	0	1	0	0	0	0.00%	SA=LO
g60d20c2	2	28902754	0	0	0	0	0	0	0.00%	SA=LO
g60d20c3	2	28974316	0	1	0	0	0	1	100.00%	SA>LO
g60d20c4	2	33351754	0	0	0	0	0	0	0.00%	SA=LO
g60d20c5	2	31094195	0	5	0	5	0	0	0.00%	SA=LO
g60d20c6	2	29845272	1	3	1	0	0	3	75.00%	SA>LO
g60d20c7	2	29834662	0	0	0	0	0	0	0.00%	SA=LO
g60d20c8	2	28687686	1	0	1	0	0	0	0.00%	SA=LO
g60d20c9	2	27935824	1	0	1	0	0	0	0.00%	SA=LO
g100d30c0	2	225610511	0	0	0	0	0	0	0.00%	SA=LO
g100d30c1	2	206073651	1	0	1	0	0	0	0.00%	SA=LO
g100d30c2	2	163223911	1	0	1	0	0	0	0.00%	SA=LO
g100d30c3	2	203382267	0	0	0	0	0	0	0.00%	SA=LO
g100d30c4	2	205191572	1	0	1	0	0	0	0.00%	SA=LO
g100d30c5	2	231987925	0	0	0	0	0	0	0.00%	SA=LO
g100d30c6	2	176986484	1	0	1	0	0	0	0.00%	SA=LO
g100d30c7	2	174884793	1	0	1	0	0	0	0.00%	SA=LO
g100d30c8	2	219606035	0	0	0	0	0	0	0.00%	SA=LO
g100d30c9	2	206095919	0	0	0	0	0	0	0.00%	SA=LO
g200d30c0	2	1384999832	1	1	1	0	0	1	50.00%	SA>LO
g200d30c1	2	1705949570	0	0	0	0	0	0	0.00%	SA=LO
g200d30c2	2	1751191201	0	0	0	0	0	0	0.00%	SA=LO
g200d30c3	2	1647418647	1	0	1	0	0	0	0.00%	SA=LO
g200d30c4	2	1522644381	1	0	1	0	0	0	0.00%	SA=LO
g200d30c5	2	1628774459	0	0	0	0	0	0	0.00%	SA=LO
g200d30c6	2	1735163940	0	0	0	0	0	0	0.00%	SA=LO
g200d30c7	2	1705365512	0	0	0	0	0	0	0.00%	SA=LO
g200d30c8	2	1542222814	1	0	1	0	0	0	0.00%	SA=LO
g200d30c9	2	1619136837	0	1	0	0	0	1	100.00%	SA>LO
g20d8c0	4	771472	0	4	0	4	0	0	0.00%	SA=LO
g20d8c1	4	620015	3	5	3	2	0	3	37.50%	SA>LO
g20d8c2	4	655726	0	4	0	0	0	4	100.00%	SA>LO
g20d8c3	4	626891	3	9	3	4	0	5	41.67%	SA>LO
g20d8c4	4	786837	0	13	0	11	0	2	15.38%	SA>LO
g20d8c5	4	596222	3	8	3	2	0	6	54.55%	SA>LO

g20d8c6	4	669882	3	2	3	0	0	2	40.00%	SA>LO
g20d8c7	4	600761	4	7	4	3	0	4	36.36%	SA>LO
g20d8c8	4	665242	3	5	3	3	0	2	25.00%	SA>LO
g20d8c9	4	687545	3	8	3	5	0	3	27.27%	SA>LO
g40d15c0	4	9099585	4	9	4	5	0	4	30.77%	SA>LO
g40d15c1	4	13281721	0	8	0	4	0	4	50.00%	SA>LO
g40d15c2	4	8007193	4	5	4	5	0	0	0.00%	SA=LO
g40d15c3	4	8613004	3	7	3	0	0	7	70.00%	SA>LO
g40d15c4	4	9773509	3	5	3	0	0	5	62.50%	SA>LO
g40d15c5	4	9368386	0	7	0	5	0	2	28.57%	SA>LO
g40d15c6	4	9198193	0	3	0	3	0	0	0.00%	SA=LO
g40d15c7	4	10296304	4	1	4	0	0	1	20.00%	SA>LO
g40d15c8	4	10570604	0	10	0	0	0	10	100.00%	SA>LO
g40d15c9	4	11250081	3	9	3	5	0	4	33.33%	SA>LO
g60d20c0	4	46906934	3	6	3	1	0	5	55.56%	SA>LO
g60d20c1	4	49546703	3	0	3	0	0	0	0.00%	SA=LO
g60d20c2	4	43352376	4	2	4	1	0	1	16.67%	SA>LO
g60d20c3	4	43459767	4	5	4	1	0	4	44.44%	SA>LO
g60d20c4	4	50027664	0	1	0	0	0	1	100.00%	SA>LO
g60d20c5	4	46639442	4	12	4	7	0	5	31.25%	SA>LO
g60d20c6	4	44767962	3	4	3	3	0	1	14.29%	SA>LO
g60d20c7	4	44752032	0	0	0	0	0	0	0.00%	SA=LO
g60d20c8	4	43031562	3	8	3	7	0	1	9.09%	SA>LO
g60d20c9	4	41903765	3	4	3	0	0	4	57.14%	SA>LO
g100d30c0	4	338415875	0	5	0	1	0	4	80.00%	SA>LO
g100d30c1	4	309110374	3	4	3	0	0	4	57.14%	SA>LO
g100d30c2	4	244835848	3	9	3	8	0	1	8.33%	SA>LO
g100d30c3	4	305073412	0	5	0	0	0	5	100.00%	SA>LO
g100d30c4	4	307787271	3	3	3	0	0	3	50.00%	SA>LO
g100d30c5	4	347982149	0	7	0	6	0	1	14.29%	SA>LO
g100d30c6	4	265479656	3	7	3	2	0	5	50.00%	SA>LO
g100d30c7	4	262327210	3	2	3	0	0	2	40.00%	SA>LO
g100d30c8	4	329409082	0	5	0	0	0	5	100.00%	SA>LO
g100d30c9	4	309139530	4	4	4	0	0	4	50.00%	SA>LO
g200d30c0	4	2077499676	3	2	3	1	0	1	20.00%	SA>LO
g200d30c1	4	2558924455	0	0	0	0	0	0	0.00%	SA=LO
g200d30c2	4	2626778230	4	0	4	0	0	0	0.00%	SA=LO
g200d30c3	4	2471128054	3	5	3	3	0	2	25.00%	SA>LO
g200d30c4	4	2283966634	3	5	3	0	0	5	62.50%	SA>LO
g200d30c5	4	2443152833	4	0	4	0	0	0	0.00%	SA=LO
g200d30c6	4	2602746069	0	2	0	0	0	2	100.00%	SA>LO
g200d30c7	4	2558048376	0	8	0	3	0	5	62.50%	SA>LO

g200d30c8	4	2313334176	3	2	3	0	0	2	40.00%	SA>LO
g200d30c9	4	2428696576	4	3	4	0	0	3	42.86%	SA>LO
g20d8c0	6	856898	8	6	8	5	0	1	7.14%	SA>LO
g20d8c1	6	688885	5	12	5	10	0	2	11.76%	SA>LO
g20d8c2	6	728302	8	12	8	8	0	4	20.00%	SA>LO
g20d8c3	6	696576	5	19	5	18	0	1	4.17%	SA>LO
g20d8c4	6	874266	0	19	0	15	0	4	21.05%	SA>LO
g20d8c5	6	662464	5	20	5	15	0	5	20.00%	SA>LO
g20d8c6	6	744309	5	6	5	2	0	4	36.36%	SA>LO
g20d8c7	6	667455	8	16	8	11	0	5	20.83%	SA>LO
g20d8c8	6	739166	5	8	5	7	0	1	7.69%	SA>LO
g20d8c9	6	763940	5	14	5	13	0	1	5.26%	SA>LO
g40d15c0	6	10110439	8	14	8	10	0	4	18.18%	SA>LO
g40d15c1	6	14756216	8	9	8	8	0	1	5.88%	SA>LO
g40d15c2	6	8896675	8	23	8	18	0	5	16.13%	SA>LO
g40d15c3	6	9570014	5	27	5	22	0	5	15.63%	SA>LO
g40d15c4	6	10859415	5	14	5	10	0	4	21.05%	SA>LO
g40d15c5	6	10409298	0	10	0	6	0	4	40.00%	SA>LO
g40d15c6	6	10219050	8	7	8	3	0	4	26.67%	SA>LO
g40d15c7	6	11440104	8	10	8	4	0	6	33.33%	SA>LO
g40d15c8	6	11743750	8	11	8	9	0	2	10.53%	SA>LO
g40d15c9	6	12499480	9	22	9	18	0	4	12.90%	SA>LO
g60d20c0	6	52118786	5	14	5	6	0	8	42.11%	SA>LO
g60d20c1	6	55050682	9	10	9	6	0	4	21.05%	SA>LO
g60d20c2	6	48168919	8	14	8	11	0	3	13.64%	SA>LO
g60d20c3	6	48288230	8	9	8	5	0	4	23.53%	SA>LO
g60d20c4	6	55586325	0	6	0	5	0	1	16.67%	SA>LO
g60d20c5	6	51821197	8	25	8	23	0	2	6.06%	SA>LO
g60d20c6	6	49740961	9	8	9	6	0	2	11.76%	SA>LO
g60d20c7	6	49722188	8	16	8	16	0	0	0.00%	SA=LO
g60d20c8	6	47811646	9	25	9	21	0	4	11.76%	SA>LO
g60d20c9	6	46559724	5	10	5	6	0	4	26.67%	SA>LO
g100d30c0	6	376011795	8	17	8	5	0	12	48.00%	SA>LO
g100d30c1	6	343453002	9	18	9	13	0	5	18.52%	SA>LO
g100d30c2	6	272039748	5	27	5	24	0	3	9.38%	SA>LO
g100d30c3	6	338964698	8	12	8	10	0	2	10.00%	SA>LO
g100d30c4	6	341982832	9	10	9	8	0	2	10.53%	SA>LO
g100d30c5	6	386640701	8	7	8	2	0	5	33.33%	SA>LO
g100d30c6	6	294974441	9	13	9	4	0	9	40.91%	SA>LO
g100d30c7	6	291471700	9	14	9	10	0	4	17.39%	SA>LO
g100d30c8	6	366004391	8	14	8	9	0	5	22.73%	SA>LO
g100d30c9	6	343487333	8	9	8	2	0	7	41.18%	SA>LO

g200d30c0	6	2308327186	9	7	9	6	0	1	6.25%	SA>LO
g200d30c1	6	2843249420	0	5	0	3	0	2	40.00%	SA>LO
g200d30c2	6	2918640509	8	5	8	0	0	5	38.46%	SA>LO
g200d30c3	6	2745697734	5	5	5	2	0	3	30.00%	SA>LO
g200d30c4	6	2537740740	5	11	5	8	0	3	18.75%	SA>LO
g200d30c5	6	2714624314	0	10	0	10	0	0	0.00%	SA=LO
g200d30c6	6	2891940118	0	3	0	3	0	0	0.00%	SA=LO
g200d30c7	6	2842264272	8	9	8	8	0	1	5.88%	SA>LO
g200d30c8	6	2570371328	5	7	5	5	0	2	16.67%	SA>LO
g200d30c9	6	2698549741	8	8	8	7	0	1	6.25%	SA>LO
g20d8c0	8	899608	16	21	16	16	0	5	13.51%	SA>LO
g20d8c1	8	723028	15	41	15	38	0	3	5.36%	SA>LO
g20d8c2	8	764583	14	24	0	25	14	-1	34.21%	SA>LO
g20d8c3	8	730837	19	27	7	30	12	-3	19.57%	SA>LO
g20d8c4	8	916814	22	54	16	53	6	1	9.21%	SA>LO
g20d8c5	8	695356	15	30	15	20	0	10	22.22%	SA>LO
g20d8c6	8	781101	19	22	19	21	0	1	2.44%	SA>LO
g20d8c7	8	700803	12	32	12	30	0	2	4.55%	SA>LO
g20d8c8	8	775871	15	23	15	20	0	3	7.89%	SA>LO
g20d8c9	8	801875	15	32	15	29	0	3	6.38%	SA>LO
g40d15c0	8	10615883	12	22	12	20	0	2	5.88%	SA>LO
g40d15c1	8	15493452	14	24	14	23	0	1	2.63%	SA>LO
g40d15c2	8	9341382	12	40	12	39	0	1	1.92%	SA>LO
g40d15c3	8	10048476	7	33	7	31	0	2	5.00%	SA>LO
g40d15c4	8	11401411	15	30	15	19	0	11	24.44%	SA>LO
g40d15c5	8	10929760	0	27	0	24	0	3	11.11%	SA>LO
g40d15c6	8	10729407	16	19	16	14	0	5	14.29%	SA>LO
g40d15c7	8	12011991	12	26	12	24	0	2	5.26%	SA>LO
g40d15c8	8	12330288	16	38	16	35	0	3	5.88%	SA>LO
g40d15c9	8	13125073	7	46	7	41	0	5	10.42%	SA>LO
g60d20c0	8	54724752	7	28	7	19	0	9	34.62%	SA>LO
g60d20c1	8	57802658	15	20	15	19	0	1	2.94%	SA>LO
g60d20c2	8	50577216	12	19	12	15	0	4	14.81%	SA>LO
g60d20c3	8	50702473	12	13	12	9	0	4	19.05%	SA>LO
g60d20c4	8	58365642	0	7	0	4	0	3	75.00%	SA>LO
g60d20c5	8	54412051	12	35	12	23	0	12	34.29%	SA>LO
g60d20c6	8	52229244	7	14	7	12	0	2	10.53%	SA>LO
g60d20c7	8	52210787	0	25	0	21	0	4	19.05%	SA>LO
g60d20c8	8	50203484	7	37	7	35	0	2	4.76%	SA>LO
g60d20c9	8	48887731	7	25	7	18	0	7	28.00%	SA>LO
g100d30c0	8	394818477	0	24	0	15	0	9	60.00%	SA>LO
g100d30c1	8	360628818	7	33	7	30	0	3	8.11%	SA>LO

g100d30c2	8	285637226	15	40	15	33	0	7	14.58%	SA>LO
g100d30c3	8	355919030	0	27	0	25	0	2	8.00%	SA>LO
g100d30c4	8	359085203	7	14	7	13	0	1	5.00%	SA>LO
g100d30c5	8	405969937	16	9	16	4	0	5	25.00%	SA>LO
g100d30c6	8	309726257	7	26	7	22	0	4	13.79%	SA>LO
g100d30c7	8	306048434	7	25	7	23	0	2	6.67%	SA>LO
g100d30c8	8	384301935	16	14	16	13	0	1	3.45%	SA>LO
g100d30c9	8	360661242	12	18	12	17	0	1	3.45%	SA>LO
g200d30c0	8	2423749484	7	13	7	6	0	7	53.85%	SA>LO
g200d30c1	8	2985393878	16	16	16	14	0	2	6.67%	SA>LO
g200d30c2	8	3064571547	12	5	12	2	0	3	21.43%	SA>LO
g200d30c3	8	2882982712	7	7	7	2	0	5	55.56%	SA>LO
g200d30c4	8	2664618735	15	22	15	18	0	4	12.12%	SA>LO
g200d30c5	8	2850341650	12	15	12	9	0	6	28.57%	SA>LO
g200d30c6	8	3036518778	16	14	16	11	0	3	11.11%	SA>LO
g200d30c7	8	2984389772	0	13	0	12	0	1	8.33%	SA>LO
g200d30c8	8	2698880793	15	9	15	5	0	4	20.00%	SA>LO
g200d30c9	8	2833476297	12	19	12	14	0	5	19.23%	SA>LO

From **Table 16** we can conclude that the performance for simulated annealing is improved in $W_1(\pi)$ and $W_2(\pi)$ when either vertex number or partition number increases. Graph bisections show the same result for local optimization and simulated annealing. When the partition number increases to 4, 6 and 8, the performance is obviously better in simulated annealing. In terms of the summation of $W_1(\pi)$ and $W_2(\pi)$, simulated annealing outperforms local optimization by 1.92 % to 100%.

But simulated annealing runs much longer than local optimization. For our benchmark problem instances, simulated annealing runs about 6 to 100 times longer than local optimization.

6.5 Summary of Solution Quality Comparisons

Table 17 compares values for both $W_1(\pi)$ and $W_2(\pi)$ among Repeated Random, Local Optimization, and Simulated Annealing.

Table 17 Comparisons of $W_1(\pi)$ and $W_2(\pi)$ for RR, LO and SA

Data Files	*m*	$W_1(\pi)$ RR	$W_2(\pi)$ RR	$W_1(\pi)$ LO	$W_2(\pi)$ LO	$W_1(\pi)$ LO	$W_2(\pi)$ SA	Result
g20d8c0	2	0	0	0	0	0	0	SA=LO=RR
g20d8c1	2	1	0	1	0	1	0	SA=LO=RR
g20d8c2	2	0	0	0	0	0	0	SA=LO=RR
g20d8c3	2	1	0	1	0	1	0	SA=LO=RR
g20d8c4	2	0	0	0	0	0	0	SA=LO=RR
g20d8c5	2	1	0	1	5	1	0	SA=RR>LO
g20d8c6	2	1	0	1	0	1	0	SA=LO=RR
g20d8c7	2	0	0	0	0	0	0	SA=LO=RR
g20d8c8	2	1	0	1	0	1	0	SA=LO=RR
g20d8c9	2	1	1	1	1	1	0	SA>LO=RR
g40d15c0	2	0	0	0	1	0	0	SA=RR>LO
g40d15c1	2	0	0	0	0	0	0	SA=LO=RR
g40d15c2	2	0	5	0	5	0	0	SA>LO=RR
g40d15c3	2	1	0	1	0	1	0	SA=LO=RR
g40d15c4	2	1	0	1	0	1	0	SA=LO=RR
g40d15c5	2	0	0	0	2	0	0	SA=RR>LO
g40d15c6	2	0	0	0	3	0	0	SA=RR>LO
g40d15c7	2	0	0	0	0	0	0	SA=LO=RR
g40d15c8	2	0	0	0	0	0	0	SA=LO=RR
g40d15c9	2	1	0	1	4	1	0	SA=RR>LO
g60d20c0	2	1	1	1	1	1	0	SA>LO=RR
g60d20c1	2	1	0	1	0	1	0	SA=LO=RR
g60d20c2	2	0	0	0	0	0	0	SA=LO=RR
g60d20c3	2	0	0	0	1	0	0	SA=RR>LO
g60d20c4	2	0	0	0	0	0	0	SA=LO=RR
g60d20c5	2	0	5	0	5	0	5	SA=LO=RR
g60d20c6	2	1	3	1	3	1	0	SA>LO=RR
g60d20c7	2	0	0	0	0	0	0	SA=LO=RR
g60d20c8	2	1	0	1	0	1	0	SA=LO=RR
g60d20c9	2	1	0	1	0	1	0	SA=LO=RR

g100d30c0	2	0	0	0	0	0	0	SA=LO=RR
g100d30c1	2	1	0	1	0	1	0	SA=LO=RR
g100d30c2	2	1	3	1	0	1	0	SA=LO>RR
g100d30c3	2	0	0	0	0	0	0	SA=LO=RR
g100d30c4	2	1	0	1	0	1	0	SA=LO=RR
g100d30c5	2	0	0	0	0	0	0	SA=LO=RR
g100d30c6	2	1	0	1	0	1	0	SA=LO=RR
g100d30c7	2	1	0	1	0	1	0	SA=LO=RR
g100d30c8	2	0	0	0	0	0	0	SA=LO=RR
g100d30c9	2	0	0	0	0	0	0	SA=LO=RR
g200d30c0	2	1	1	1	1	1	0	SA>LO=RR
g200d30c1	2	0	0	0	0	0	0	SA=LO=RR
g200d30c2	2	0	0	0	0	0	0	SA=LO=RR
g200d30c3	2	1	0	1	0	1	0	SA=LO=RR
g200d30c4	2	1	0	1	0	1	0	SA=LO=RR
g200d30c5	2	0	0	0	0	0	0	SA=LO=RR
g200d30c6	2	0	0	0	0	0	0	SA=LO=RR
g200d30c7	2	0	0	0	0	0	0	SA=LO=RR
g200d30c8	2	1	0	1	0	1	0	SA=LO=RR
g200d30c9	2	0	0	0	1	0	0	SA=RR>LO
g20d8c0	4	6	6	0	4	0	4	SA=LO>RR
g20d8c1	4	3	3	3	5	3	2	SA>RR>LO
g20d8c2	4	6	4	0	4	0	0	SA>LO>RR
g20d8c3	4	3	9	3	9	3	4	SA>LO=RR
g20d8c4	4	6	9	0	13	0	11	SA>LO>RR
g20d8c5	4	3	10	3	8	3	2	SA>LO>RR
g20d8c6	4	7	2	3	2	3	0	SA>LO>RR
g20d8c7	4	4	3	4	7	4	3	SA=RR>LO
g20d8c8	4	3	3	3	5	3	3	SA=RR>LO
g20d8c9	4	7	5	3	8	3	5	SA>LO>RR
g40d15c0	4	10	9	4	9	4	5	SA>LO>RR
g40d15c1	4	10	4	0	8	0	4	SA>LO>RR
g40d15c2	4	6	5	4	5	4	5	SA=LO>RR
g40d15c3	4	7	7	3	7	3	0	SA>LO>RR
g40d15c4	4	7	1	3	5	3	0	SA>LO>RR
g40d15c5	4	6	5	0	7	0	5	SA>LO>RR
g40d15c6	4	6	3	0	3	0	3	SA=LO>RR
g40d15c7	4	10	1	4	1	4	0	SA>LO>RR
g40d15c8	4	0	5	0	10	0	0	SA>RR>LO
g40d15c9	4	9	13	3	9	3	5	SA>LO>RR
g60d20c0	4	3	7	3	6	3	1	SA>RR>LO
g60d20c1	4	9	0	3	0	3	0	SA=LO>RR

g60d20c2	4	4	6	4	2	4	1	SA>RR>LO
g60d20c3	4	10	4	4	5	4	1	SA>RR>LO
g60d20c4	4	6	1	0	1	0	0	SA>RR>LO
g60d20c5	4	10	7	4	12	4	7	SA>RR>LO
g60d20c6	4	11	4	3	4	3	3	SA>RR>LO
g60d20c7	4	6	0	0	0	0	0	SA=LO>RR
g60d20c8	4	7	8	3	8	3	7	SA>RR>LO
g60d20c9	4	9	0	3	4	3	0	SA>RR>LO
g100d30c0	4	18	6	0	5	0	1	SA>RR>LO
g100d30c1	4	9	4	3	4	3	0	SA>RR>LO
g100d30c2	4	7	9	3	9	3	8	SA>RR>LO
g100d30c3	4	6	5	0	5	0	0	SA>RR>LO
g100d30c4	4	11	3	3	3	3	0	SA>RR>LO
g100d30c5	4	10	6	0	7	0	6	SA>RR>LO
g100d30c6	4	7	7	3	7	3	2	SA>RR>LO
g100d30c7	4	9	5	3	2	3	0	SA>RR>LO
g100d30c8	4	10	2	0	5	0	0	SA>RR>LO
g100d30c9	4	4	4	4	4	4	0	SA>LO=RR
g200d30c0	4	7	1	3	2	3	1	SA>RR>LO
g200d30c1	4	16	0	0	0	0	0	SA=LO>RR
g200d30c2	4	18	0	4	0	4	0	SA=LO>RR
g200d30c3	4	13	2	3	5	3	3	SA>RR>LO
g200d30c4	4	9	5	3	5	3	0	SA>RR>LO
g200d30c5	4	16	0	4	0	4	0	SA=LO>RR
g200d30c6	4	14	2	0	2	0	0	SA>LO>RR
g200d30c7	4	10	0	0	8	0	3	SA>LO>RR
g200d30c8	4	17	2	3	2	3	0	SA>LO>RR
g200d30c9	4	10	3	4	3	4	0	SA>LO>RR
g20d8c0	6	26	14	8	6	8	5	SA>LO>RR
g20d8c1	6	23	18	5	12	5	10	SA>LO>RR
g20d8c2	6	22	12	8	12	8	8	SA>LO>RR
g20d8c3	6	27	24	5	19	5	18	SA>LO>RR
g20d8c4	6	18	23	0	19	0	15	SA>LO>RR
g20d8c5	6	13	14	5	20	5	15	SA>LO>RR
g20d8c6	6	27	7	5	6	5	2	SA>LO>RR
g20d8c7	6	24	19	8	16	8	11	SA>LO>RR
g20d8c8	6	13	14	5	8	5	7	SA>LO>RR
g20d8c9	6	31	23	5	14	5	13	SA>LO>RR
g40d15c0	6	22	15	8	14	8	10	SA>LO>RR
g40d15c1	6	36	8	8	9	8	8	SA>LO>RR
g40d15c2	6	30	13	8	23	8	18	SA>LO>RR
g40d15c3	6	29	21	5	27	5	22	SA>LO>RR

g40d15c4	6	21	11	5	14	5	10	SA>LO>RR
g40d15c5	6	24	6	0	10	0	6	SA>LO>RR
g40d15c6	6	30	8	8	7	8	3	SA>LO>RR
g40d15c7	6	22	12	8	10	8	4	SA>LO>RR
g40d15c8	6	28	18	8	11	8	9	SA>LO>RR
g40d15c9	6	29	22	9	22	9	18	SA>LO>RR
g60d20c0	6	35	15	5	14	5	6	SA>LO>RR
g60d20c1	6	43	6	9	10	9	6	SA>LO>RR
g60d20c2	6	46	19	8	14	8	11	SA>LO>RR
g60d20c3	6	36	9	8	9	8	5	SA>LO>RR
g60d20c4	6	34	6	0	6	0	5	SA>LO>RR
g60d20c5	6	38	15	8	25	8	23	SA>LO>RR
g60d20c6	6	29	8	9	8	9	6	SA>LO>RR
g60d20c7	6	28	8	8	16	8	16	SA>LO>RR
g60d20c8	6	47	20	9	25	9	21	SA>LO>RR
g60d20c9	6	37	8	5	10	5	6	SA>LO>RR
g100d30c0	6	36	16	8	17	8	5	SA>LO>RR
g100d30c1	6	43	7	9	18	9	13	SA>LO>RR
g100d30c2	6	37	26	5	27	5	24	SA>LO>RR
g100d30c3	6	60	14	8	12	8	10	SA>LO>RR
g100d30c4	6	37	7	9	10	9	8	SA>LO>RR
g100d30c5	6	46	7	8	7	8	2	SA>LO>RR
g100d30c6	6	41	18	9	13	9	4	SA>LO>RR
g100d30c7	6	41	14	9	14	9	10	SA>LO>RR
g100d30c8	6	56	11	8	14	8	9	SA>LO>RR
g100d30c9	6	48	9	8	9	8	2	SA>LO>RR
g200d30c0	6	67	7	9	7	9	6	SA>LO>RR
g200d30c1	6	58	2	0	5	0	3	SA>LO>RR
g200d30c2	6	52	5	8	5	8	0	SA>LO>RR
g200d30c3	6	79	2	5	5	5	2	SA>LO>RR
g200d30c4	6	47	4	5	11	5	8	SA>LO>RR
g200d30c5	6	52	10	0	10	0	10	SA=LO>RR
g200d30c6	6	64	3	0	3	0	3	SA=LO>RR
g200d30c7	6	76	6	8	9	8	8	SA>LO>RR
g200d30c8	6	53	7	5	7	5	5	SA>LO>RR
g200d30c9	6	52	5	8	8	8	7	SA>LO>RR
g20d8c0	8	40	25	16	21	16	16	SA>LO>RR
g20d8c1	8	53	36	15	41	15	38	SA>LO>RR
g20d8c2	8	40	27	14	24	0	25	SA>LO>RR
g20d8c3	8	39	34	19	27	7	30	SA>LO>RR
g20d8c4	8	56	58	22	54	16	53	SA>LO>RR
g20d8c5	8	55	29	15	30	15	20	SA>LO>RR

g20d8c6	8	53	24	19	22	19	21	SA>LO>RR
g20d8c7	8	38	35	12	32	12	30	SA>LO>RR
g20d8c8	8	41	23	15	23	15	20	SA>LO>RR
g20d8c9	8	49	42	15	32	15	29	SA>LO>RR
g40d15c0	8	76	25	12	22	12	20	SA>LO>RR
g40d15c1	8	44	19	14	24	14	23	SA>LO>RR
g40d15c2	8	60	38	12	40	12	39	SA>LO>RR
g40d15c3	8	63	35	7	33	7	31	SA>LO>RR
g40d15c4	8	53	33	15	30	15	19	SA>LO>RR

Based on the above table we can observe that the performance for SA is equal to those for LO and RR when the vertex number is small or the partition number is small, and the performance for SA is better than LO, which is in turn better than RR, when the vertex number is large and the partition number is large.

Chapter 7

Conclusion

This dissertation used multi-way graph partitioning to model the distributed component allocation problem on clustered application servers, and used simulated annealing as the meta-heuristic for deriving efficient solution heuristics for optimized distributed component allocations that will maximize computation work load balance and minimize inter-machine communication overhead. The efficient solution to this problem has important implications in improving the scalability and availability of today's e-commerce portals.

The major contributions of this research include:

- Adopting multi-way graph partition as the mathematical model for addressing a practical problem critical to the performance of e-commerce portals.

- Proving that this problem is NP-hard, so no efficient algorithms could ever be designed to produce optimal solutions to it in practical time.

- Designing a problem transformation algorithm to convert the problem with multiple objective functions into an equivalent typical combinatorial optimization problem.

- Studying and designing efficient solution neighborhood structures

- Deriving incremental objective function evaluation that can improve the performance of any iterative solution heuristics.

- Deriving efficient heuristic solutions based on simulated annealing, and studying the sensitivity of its performance to its parameter values.

Potential future works include

- Adopting more recent research results in simulated annealing;

- Adopting alternative meta-heuristics like tabu search;

- Extending the mathematical model to reflect more complex properties of hosted computing based on distributed components.

References

[1] Christian Blum, “Metaheuristics in combinatorial optimization: overview and conceptual comparison,” *ACM Computing Surveys*, Vol. 35, No. 3, September 2003, pp. 268-308

[2] T. N. Bui and B. R. Moon, “Genetic Algorithm and Graph Partitioning,” *IEEE Trans. On Computers*, vol. 45, no. 7, July 1986

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979

[4] F. Glover and G. A. Kochenberger, *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2003

[5] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997

[6] F. Glover, “Future Paths for integer programming and links to artificial intelligence,” *Computers and Operations Research*, 13, 1986, 533-549

[7] F. Glover , “Heuristics for integer programming using surrogate constraints,” *Dec. Sci.*, 8, 1977, 156-166

[8] D.E. Goldberg, “Genetic Algorithm in Search, Optimization & Machine Learning,” *Addison Wesley*, 1989

[9] A. Goscinski, *Distributed Operating Systems: The Logical Design*, Addison-Wesley, Reading, Mass., 1991

[10] Object Management Group, www.omg.org

[11] B. Hajek, “Cooling schedules for optimal annealing.” *Math Operations Research*, 13, 311-329, 1988

[12] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, “Optimization by Simulated Annealing: an Experimental Evaluation; Part I, Graph Partitioning,” *Operations Research*, vol. 37, issue 6 (Nov.-Dec.), 1989, pp. 865-892.

[13] B. W. Kernighan and S. Lin, “An efficient Heuristic procedure for partitioning graphs,” *Bell System Tech. Journal*, vol. 49, Feb., 1970, pp.291-307

[14] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, 220, May 1983, pp. 671-680

[15] C. H. Lee, C.I. Park and M. Kim, “An efficient algorithm for graph-partitioning problem using a problem transformation method,” *Computer-Aided Design*, 21, 1989, pp. 611-618

[16] Sun Microsystems, "The J2EE 1.4 Tutorial," http://java.sun.com/j2ee/1.4/docs/tutorial/doc, (current February 2004)

[17] T. Mowbray and R. Zahavi, *The Essential CORBA: Systems Integration Using Distributed Objects*, John Wiley & Sons, New York, 1995

[18] D. S. Platt, *Understanding COM+*, Microsoft Press, 2000

[19] G. Svswerda. "Uniform Crossover in Genetic Algorithms," *Proc. Third Int'l Conf. Genetic Algorithms*, pp. 2-9, 1989

[20] L. Tao., "Shifting Paradigms with the Application Service Provider Model," *IEEE Computer Magazine*, Oct. 2001, pp. 32-39

[21] L. Tao, B. Narahari, and Y.C. Zhao, "Assigning Task Modules to Processors in a Distributed System," *Journal of Combinatorial Mathematics and Combinatorial Computing*, 14, 1993. pp. 97-135

[22] L. Tao and Y.C. Zhao, "Multi-Way Graph Partition by Stochastic Probe," *International Journal of Computers & Operations Research*, Vol. 20, No. 3, 1993. pp. 321-347

[23] L. Tao, "Research Incubator: Combinatorial Optimization," Technical Report #198 CSIS, Pace University, NY, http://csis.pace.edu/~lixin/dps (current February 2004)

[24] D. Whitley and J. Kauth, "Genitor: A Different Genetic Algorithm," *Proc. Rocky Mowztuin Conf. Artificial Intelligence*, pp. 118-130,1988