

We approve the thesis of Ezra Nugroho.

Date of Signature

Thang N. Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

Sukmoon Chang
Assistant Professor of Computer Science

Qin Ding
Assistant Professor of Computer Science

Pavel Naumov
Assistant Professor of Computer Science

Linda M. Null
Assistant Professor of Computer Science
Graduate Coordinator

Omid Ansary
Professor of Electrical Engineering
Director, School of Science, Engineering, and Technology

I grant The Pennsylvania State University the non-exclusive right to use this work for the University's own purposes and to make single copies of the work available to the public on a not-for-profit basis if copies are not otherwise available.

Ezra Nugroho

The Pennsylvania State University
The Graduate School

**ANT BASED OPTIMIZATION FOR
MULTIWAY GRAPH PARTITIONING**

A Thesis in
Computer Science
by
Ezra Nugroho

© 2005 Ezra Nugroho

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2005

The thesis of Ezra Nugroho was reviewed and approved* by the following:

Thang N. Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

Sukmoon Chang
Assistant Professor of Computer Science

Qin Ding
Assistant Professor of Computer Science

Pavel Naumov
Assistant Professor of Computer Science

Linda M. Null
Assistant Professor of Computer Science
Graduate Coordinator

Omid Ansary
Professor of Electrical Engineering
Director, School of Science, Engineering, and Technology

*Signatures are on file in the Graduate School.

Abstract

Given a graph $G = (V, E)$, the Multiway Graph Partitioning Problem is the problem of finding a partition of V into k disjoint subsets that minimizes the edges crossing the subsets. In this thesis, we introduce a new Ant Based algorithm for the Multiway Graph Partitioning Problem. This algorithm features the competition of animat colonies in claiming territories and the Kernighan–Lin algorithm to perform local optimization. Detailed experimental results are presented for $k = 2, 4, 8,$ and 32 , along with the benchmark data used in previous publications. This algorithm performs well in finding good k –way partition of structured graphs. Our results for random graphs are also competitive.

Table of Contents

List of Figures	vi
List of Tables	vii
Acknowledgements	viii
Chapter 1	
Introduction	1
Chapter 2	
Preliminaries	4
2.1 Existing Algorithms	4
2.1.1 Kernighan–Lin Algorithm (KL)	4
2.1.2 Cyclic k -way Partitioning Algorithm (CP)	5
2.1.3 Other Heuristics and Hybrid Algorithms	5
2.2 Ant Based Optimization (ABO)	6
Chapter 3	
Ant Based Optimization for Multiway Graph Partitioning	7
3.1 Overview of Algorithm	8
3.2 Algorithm Details	10
3.2.1 Updating Vertex Desirability	10
3.2.2 Animat Distribution	10
3.2.3 Animat Activation	11

3.2.3.1	Activation Probability	11
3.2.3.2	Pheromone Computation	12
3.2.3.3	Animat Activation Rules	12
3.2.4	Pheromone Evaporation	15
3.2.5	Jolt	15
3.2.6	Greedy Rebalancing Subroutine	15
3.2.7	Local Optimization	16
3.2.7.1	Kernighan–Lin Algorithm for bisection	16
3.2.7.2	Pairwise Kernighan–Lin Optimization	18
Chapter 4		
Results		19
4.1	Benchmark Graphs	19
4.2	Comparison	20
Chapter 5		
Conclusion		26
References		27

List of Figures

3.1	ABMGP main algorithm	9
3.2	Animat activation algorithm	14
3.3	KL algorithm for bisection	17
3.4	Pairwise Kernighan–Lin algorithm	18

List of Tables

4.1	Parameters used in experiment	21
4.2	Results of ABMGP for bisection	23
4.3	Results of ABMGP for 4-way partitioning	24
4.4	Results of ABMGP for 8-way partitioning and 32-way partitioning	25

Acknowledgements

I would like to express my immense gratitude to my advisor, Dr. Thang N. Bui. He has introduced me to many exciting areas to explore. He devotedly guided me since the conception of this research, and tirelessly helped me in making this thesis more presentable.

I would like to thank the members of the thesis committee: Dr. Sukmoon Chang, Dr. Qin Ding, Dr. Pavel Naumov, and Dr. Linda M. Null for reviewing this thesis.

I would also like to thank my colleagues, Mufit Colpan, Ishaan Joshi, and Gnanasekaran Sundarraj for their continuous supports and invaluable friendships.

Chapter 1

Introduction

A k -way partition of a graph $G = (V, E)$ is a partitioning of V into subsets S_1, \dots, S_k such that $S_i \cap S_j = \emptyset$, $\forall i \neq j$, $\bigcup_{i=1}^k S_i = V$, and $S_i \neq \emptyset$ for $i = 1, \dots, k$. We define $C = \{(u, v) \in E \mid u \in S_i, v \in S_j, i \neq j\}$. If the edges in C are removed from the original graph, the result is k disjoint subgraphs. The cardinality of C is often called the *cut size*.

The definition above does not provide many restrictions on the size of the subgraphs. Considering that heavily unbalanced partitions are usually less interesting, we define the following: an (α, k) -partition of a graph $G = (V, E)$ with $|V| = n$ is a partition of V into S_1, \dots, S_k such that $S_i \cap S_j = \emptyset$, for $i \neq j$, $\bigcup_{i=1}^k S_i = V$, $S_i \neq \emptyset \forall i = 1, \dots, k$, and $\max_{1 \leq i \leq k} |S_i| \leq \lceil \alpha n \rceil$. The parameter α determines how much the subsets can differ in size.

Furthermore, we define a *balanced k -way partition* of a graph $G = (V, E)$ as a partitioning of V into k disjoint subsets in which the size of the largest subset and the smallest subset differ by at most 1. In other words, a balanced k -way partition of a graph G is the $(1/k, k)$ -way partition of G .

In this thesis, we consider the problem of finding a balanced k -way partition that has the minimum cut size. Unless noted otherwise, hereafter, we refer to a balanced k -way partition only as a k -way partition, and we denote the multiway graph partitioning problem as the problem of finding a k -way partition with minimum cut size. The 2-way partition problem is also called the *bisection* problem.

It is known that the problem of minimizing and maximizing the cut size of a balanced k -way partition is NP-hard [15]. For the minimization problem, finding a

good approximation of the optimal cut size is NP-hard, unless if we consider special classes of graphs such as trees or planar graphs [5, 8]. The bisection problem is also NP-hard even for bipartite graphs [15].

For a graph with $|V| = n$, the size of the search space for the multiway graph partition problem is given by:

$$\frac{\prod_{i=0}^{k-1} \binom{n - i \cdot \frac{n}{k}}{\frac{n}{k}}}{n!},$$

which grows very fast as n increases. One may observe that, for a fixed n , the search space becomes smaller as k approaches n . However, we are usually only concerned about problems in which n is relatively very large compared to k . Initially, most researchers focused on the bisection problem only. To obtain k -way partitions, they typically suggested using a recursive application of the bisection algorithms. This restricted the size of k to a power of two.

Due to the extensive application of the minimization problem, numerous pervasive studies have been done. We highlight some of the most prominent applications of the partitioning problem below. We provide a brief survey of existing algorithms in the next chapter.

One application of the problem is optimization in circuit design. The underlying structure of an electronic schematic is a graph, which could be quite large, complex, and often non-planar. Choices in the placement of components on a board may significantly affect the amount of connectors needed to connect all the components. In the case of non-planar circuits, board designers must use bridges or additional boards. It is desirable to organize the components into sections such that within a section, the connectivity is maximized, while the connections among sections are minimized. A graph partition algorithm could help achieve this goal [5, 21].

Another important application of the graph partition problem is in the scheduling of processes in a parallel environment. One of the prominent challenges in parallel computing is the bottleneck caused by the communication among CPUs. When processes are not very tightly interconnected, one may reduce the communication cost by enlarging the units of tasks performed by each CPU. However for some complex problems, such simplification cannot be achieved because of dense

connectivity. A graph-partitioning algorithm may help decompose the connectivity diagram from a large collection of interrelated processes into clusters such that interconnections among clusters are rare. Assigning the clusters to different CPUs would reduce the communication cost [26].

Graph partitioning problem is also used in sparse matrix factorization, network partitioning, and circuit prototyping, among others. An exhaustive survey of applications of the graph partitioning problem and recent developments in the field is provided by Alpert and Kahng [1].

Chapter 2

Preliminaries

In this chapter we provide a quick literature survey of available algorithms to solve both the bisection problem and the general graph partition problem. We also introduce the concept of Ant Based Optimization and its close relative, Ant Colony Optimization (ACO). We describe their similarities, and we also point out how the two differ.

2.1 Existing Algorithms

2.1.1 Kernighan–Lin Algorithm (KL)

In the late 1960s, B. W. Kernighan and S. Lin [21] from Bell Laboratory developed a local optimization technique for the bisection problem. The algorithm starts with a bisection (A, B) , either created through random means or from the result of some other algorithm. The algorithm consists of a number of iterations. During each iteration, the algorithm exchanges equal size subsets between A and B . The bisection produced from one iteration is used as the initial bisection for the next iteration. The algorithm terminates when a certain number of iterations have been run, or until no more improvements can be made.

There are several ways to extend the original Kernighan–Lin algorithm to optimize k -way partitioning. If k is a power of two, one can perform the algorithm recursively, each time dividing the subset into two. This is a rather popular approach because of its simplicity. A potential problem in this approach is that the

initial partitioning stage tries to minimize the cut sizes between subsets without considering the effect on the next stage. This may cause high cut sizes in subsequent partitions [7, 19].

An alternative is to start with k subsets and run the bisection optimization between pairs [19, 21]. Since there are $\binom{k}{2}$ pairs of subsets in a k -way partition, we must run $\binom{k}{2}$ application of the original Kernighan–Lin for each pass. However, a optimization on one pair may increase the cut sizes between some other pairs. Therefore, several passes of the algorithm may be required before a k -way partition can reach a local optima. This shows that as k increases, the complexity of the problem increases greatly.

2.1.2 Cyclic k -way Partitioning Algorithm (CP)

Kang and Moon [19] suggested that the performance of KL-based optimization methods for the k -partitioning problem may be sub-optimal, noting that each KL run can only considers the “local scope”. They proposed the Cyclic Partitioning (CP) Algorithm that performs global optimization. Similar to pairwise KL, CP also consists of vertex swapping iterations between subgraphs. However, instead of predefining the ordering of pairs in which the optimizations are done, CP decides on the next pair to optimize based on the current results. One pass of CP terminates after the algorithm has performed optimization to a cyclical pair list.

2.1.3 Other Heuristics and Hybrid Algorithms

Johnson, et al. [18] has extensively studied the performance of simulated annealing algorithm for the bisection problem. They showed that simulated annealing on average performs better than the Kernighan–Linn algorithm. However, the performance of the Kernighan–Linn algorithm largely depends on the quality of the bisection that it starts with. Considering this fact, many researchers have attempted to create hybrid algorithms in which the Kernighan–Lin algorithm is used in conjunction with some earlier search mechanisms. Genetic algorithms have been used extensively with various levels of success [7, 12]. Other heuristics that have been used to solve the bisection problem include Tabu Search [2] and a hybrid ant based algorithm [10].

2.2 Ant Based Optimization (ABO)

Both *Ant Based Optimization (ABO)* and *Ant Colony Optimization (ACO)* are inspired by the collective behavior of agents and their abilities to solve problems. Initially, much of the analogy is derived from the behavior of ant colonies; nevertheless newer algorithms have also drawn analogies from other biological creatures. To express these analogies fairly, the agents are no longer addressed as ‘ants’, but instead as ‘animats.’

Animats in a colony solve problems by passing messages to each other in the form of pheromone. Animats lay more or less pheromone according to how they perceive their conditions at a given time. Other animats arriving at a similar situation consider the amount of pheromone that has been laid, and use this to make informed decisions.

Ant Based Optimization differs from Ant Colony Optimization mainly in how the agents are used. In ACO, each animat produces a solution to the problem. The initial animat solves the problem almost purely stochastically. The subsequent animats use the information provided by the previous animats to produce better solutions. In ABO, animats do not attempt to solve the problem individually. The solution to the problem is represented by the state of all animats in the colonies captured in a snapshot of time.

Ant Colony Optimization excels in solving problems that are analogous to the shortest path problem (or the longest path problem, with minor modification to the heuristic). However, for problems that do not resemble a path optimization problem, the original ACO may not perform well.

Ant Based Optimization has been shown to be effective in solving the k-tree problem [11], the clique problem [9], and the bisection problem [10], while ACO has been used to solve problems like the traveling salesman problem, the quadratic assignment problem, routing, and the knapsack problem, among others [6, 25].

Chapter 3

Ant Based Optimization for Multiway Graph Partitioning

The idea behind our algorithm is the observation that, in nature, colonies of competitive and territorial animals try to avoid each other as much as possible. If avoidance is not feasible, colonies try to reduce conflicts by minimizing the contact points (territorial boundaries), while maximizing the area of their territories.

Consider two colonies of lions sharing a savanna as their hunting ground. The prides would claim hunting territories as large as possible, proportional to their ability to defend them. They would establish clear territorial boundaries that each pride would not cross; the boundaries would tend to consist of relatively straight lines instead of wavy curves. Straight lines reduce the boundaries, although they may not be minimized. Since a larger pride may defend a larger region, it may have a larger hunting territory compared to that of a smaller pride. However, if the size of each pride is comparable, the territories should be of similar sizes.

In this algorithm, we try to utilize such behavior to solve the graph partitioning problem. We use colonies of animats to compete for vertices of a graph as the building blocks of their territories. Vertices are colonized by the animats residing there. These vertices may be contended by animats from different colonies. Consider a vertex v that has more neighbors belonging to other colonies than neighbors from its own. Such a vertex has more ways of being attacked than ways to be defended. This vertex has a higher chance to be captured by animats from other colonies than being retained. In this algorithm, the animats are efficient in finding such

vertices and assigning them to the appropriate colonies. This is our main vehicle for improvement.

We believe that k competitive and territorial animat colonies can dissect a graph into k territories that resemble a good quality k -way partition. Although the cut sizes between these k sections may not be minimal, we believe that it is good enough that a local optimization algorithm can push it to a good local optimum. We call this algorithm Ant Based Optimization for Multiway Graph Partitioning (ABMGP).

3.1 Overview of Algorithm

In this section, we present the algorithm. We begin with a brief overview, and then provide entailed descriptions on the components subsequent sections.

In our algorithm, we commission k colonies of competitive animats to create k territories. The algorithm creates an initial k -way partitioning by randomly assigning each vertex to one of the k colonies. Animats are then randomly distributed to occupy available territories (vertices). We address the state of the animats on the graph at any point in time as a *configuration*. Each different configuration represents a unique k -partitioning of the graph.

The algorithm, shown in Figure 3.1, consists of γ rounds, each containing σ iterations. Within an iteration, a number of animats are activated. Activated animats decide to perform one of three possible actions depending on their situations. These three actions include idling (not making any move), moving to an adjacent vertex owned by the colony, and attacking another animat of a different colony in an adjacent vertex.

An animat that is defending an attack is called a *defending animat*. Defending animats that are defeated are called *killed*. Animats that are killed are removed from the vertices that they occupy. When a vertex is left with no defending animat, the vertex is captured by the colony of the attacker. Such vertex captures change the configuration of the animats, hence changing the partitioning. Animats that are killed in one iteration are replaced by new animats in the next iteration. When these new animats are added, we say that these animats are *born*.

To keep partitions from becoming heavily unbalanced, the algorithm randomly

-
1. ABMGP ($G = (V, E), k$)
 2. Create initial random k -way partition
 3. Randomly assign animats to vertices
 4. for $round=1$ to γ
 5. for $iteration=1$ to σ
 6. Update vertex desirability values
 7. Distribute new animats
 8. if (activation condition met)
 9. Activate animats in parallel
 10. Evaporate ϵ percent of the pheromone from vertices
 11. else
 12. Perform jolt procedure
 13. end if - else
 - 14.
 15. Rebalance partition with ϱ
 16. Record the partition if good
 17. end iterations
 18. Perform local optimization on best partition found in this round
 19. Record the partition if good
 20. end rounds
 21. Perform further local optimization on the best overall partition
 22. Return the best partition found
 23. end of ABMGP
-

Figure 3.1: ABMGP main algorithm

uses a greedy rebalancing mechanism. Within a round, the animats may find themselves stuck at a plateau or a local optima, and unable to move away from it. In this situation, we pause the animats for several iterations and then perform random jumps in the search space. We call these random jumps *jolts*.

The best partitions found within a round and the absolute best partition throughout the algorithm are recorded. We optimize the best configuration found within the round by using a weaker derivation of the pairwise Kernighan–Lin algorithm. At the end of the algorithm, we optimize the best partitioning found so far using the original pairwise Kernighan–Lin algorithm, and return the result as our final k -way partition.

3.2 Algorithm Details

3.2.1 Updating Vertex Desirability

For each vertex, we assign attributes called the *desirability values*, which represent how desirable the vertex is to each of the colony. Desirability values are generally used to decide the distribution of animats, and the selection of destinations when animats perform actions.

An iteration is initiated by updating the desirability values. These values are largely based on the pheromone levels in vertices at the end of the previous iteration. However, a high level of pheromone in a vertex can have a feedback effect. The higher the level of pheromone in a vertex, the more it attracts animats, which in turn increases the level of pheromone even more.

To avoid over-crowding behavior, when computing desirability values, the algorithm takes into account how many animats already reside at a given vertex. The desirability value of a vertex v belonging to a colony s , denoted by $d(v, s)$, is given by:

$$d(v, s) = \begin{cases} phero(v, s)/a(v) & \text{if } v \text{ is of colony } s \\ phero(v, s) & \text{otherwise} \end{cases}$$

where $phero(v, s)$ represents the amount of pheromone from colony s deposited at vertex v , and $a(v)$ represents the number of animats in vertex v . Using this formula, overcrowded vertices will be less attractive to animats that are attempting to move, even when the pheromone level is very high.

3.2.2 Animat Distribution

Following the desirability updates, a number of new animats are born to replace the ones killed in the previous iteration. To maintain the balance of the strength of each colony, the number of animats born is set equal to the number of animats killed in the previous iteration. This encourages configurations that resemble balanced partitions.

For each animat killed, a new animat from the same colony is born. This new animat is placed on a vertex that is randomly selected from all vertices owned by that colony. The probability that a vertex is selected is proportional to the amount

of pheromone in that vertex compared to the sum of pheromone in the colonized vertices.

3.2.3 Animat Activation

The behavior of the animats is the most important aspect of this algorithm. There are several important details in the design that necessitate the organization of this section into the subsections presented below.

3.2.3.1 Activation Probability

Beyond designing our animats to be territorial and competitive, we also need to tune the level of aggressiveness of the animats. Aggressive colonies will be more prone to changes, hence promoting exploration. Conversely, colonies that are less aggressive are more selective of changes, hence promoting exploitation.

In the iterations at the beginning of a round, a high percentage of the animats are activated. Towards the end, the number of animats activated per iteration is reduced. Changes to the probability of animat activation need to be gradual, however a linear reduction to this probability may not be desirable. We define the probability of animat activation as a function of a conceptual nonlinear time.

To obtain this nonlinear time measure, the algorithm utilizes a sigmoid-like decay function. This function returns a value close to 1.0 in iterations at the beginning of a round. It begins to decay at a predefined inflexion point, and continues to decay until it reaches the predetermined minimum value at the end of the round. By adjusting the inflection point and the minimum value, we can control the rate at which the algorithm alters its strategy from exploration to exploitation.

The time function is calculated as follows:

$$\begin{aligned} \xi_{start} &= \varsigma_{start} * \sigma \\ \xi_{stop} &= \varsigma_{stop} * \sigma \\ t(i) &= \begin{cases} .95 & \text{if } i \leq \xi_{start} \\ .95 - (i - \xi_{start}) / (\xi_{stop} - \xi_{start}) & \text{otherwise} \end{cases} \end{aligned}$$

where $t(i)$ signifies the time value for iteration number i , σ is the maximum iter-

ations to be run, ξ_{start} is the inflexion point, and ξ_{stop} determines how steep the decline of time value after inflexion. The values of ζ_{start} , ζ_{stop} , and σ are given in the Table 4.1.

3.2.3.2 Pheromone Computation

To decide the amount of pheromone to drop, an animat first considers the ratio between the number of adjacent vertices owned by the colony and the vertex degree. If this ratio is high, the animat knows that the vertex it resides on is a good candidate for retention. The animat reinforces this fact by laying a larger amount of pheromone. Conversely, if this ratio is low, the animat drops a smaller amount of pheromone to avoid the retention of bad vertices in the colony.

Pheromone accumulation that grows too fast encourages premature exploitation behavior. To avoid this problem, animats drop only a small amount of pheromone at the beginning of a round and more towards the end. This allows animats' decisions to be more stochastic early in the round, when pheromone levels are low. As pheromone levels build up with time, animats' actions are more directed by the pheromone.

The size of a *unit pheromone* deposited by an animat residing in vertex v at iteration i , denoted by $p(v, i)$, is given by:

$$p(v, i) = \frac{v_{col}}{v_{total}} \cdot \frac{i}{\sigma}$$

where v_{col} is the number of vertices adjacent to v that belong to the same colony, v_{total} is the total number of vertices adjacent to v , and σ is the maximum number of iterations.

3.2.3.3 Animat Activation Rules

In ABMGP, each vertex is colonized at all times. This is guaranteed by having at least one animat from the colony reside in each colonized vertex. Therefore, if there is only one animat in a vertex, that animat must not make any action. This is called idling.

An activated animat first decides whether to make a defensive move or to make an offensive one; this decision is based on the colonization of the neighbors. An

animat residing in vertex v chooses to make a defensive move with a probability v_{col}/v_{total} , where v_{col} is the number of vertices adjacent to v that belong to other colonies, and v_{total} is the degree of vertex v . Animats surrounded by many enemies will tend to be more aggressive, while ones surrounded by more friends tend to be defensive. This rule further promotes exploration in the early iterations and exploitation towards the end of a round.

If an animat chooses to make a defensive action, it stochastically decides whether to reallocate to a friendly neighbor or to stay. If it chooses to move, it randomly selects the destination proportional to the desirability values of vertices in the neighborhood list. Because desirability values of colonized vertices take into account the number of current residing animats, animats prefer not to overcrowd good vertices. Regardless of the animat's decision, it deposits one unit of pheromone on the vertex of origin.

The rules for an animat's offensive actions are slightly more complicated compared. An attacking animat randomly selects an attack destination from the neighborhood list based on the proportion of the desirability values. If the attacker wins, an animat in the defending vertex is killed, i.e., the animat is removed from the vertex. The attacking animat deposits two units of its pheromone on the target vertex to entice its friends to attack the same target. If the attack is lost, the attacking animat simply uses up its turn. It remains on the vertex to be activated in the next iteration.

Although the attack on a vertex may be won, the hostile vertex may or may not be captured by the attacking colony. One victory means the reduction of the number of animats in the target vertex. Only when the attacking animat defeats the last animat in a target is the vertex captured. Upon capturing the target, the winning animat moves to the target and drops one more unit of pheromone to the newly colonized vertex.

The result of an attack is decided randomly, much like flipping a coin. However the probability of winning is not equal to the probability of losing. The probability

```

1. Activate ( $v, i$ )
2.    $p \leftarrow p(v, i)$  //the unit pheromone
3.   for each animat  $a$  in  $v$ 
4.     activate  $a$  with probability  $\pi * t(i)$ 
5.     if  $a$  activated
6.        $a$  drops  $p$  pheromone in  $v$ 
7.        $a$  chooses defensive action with probability  $v_{col}/v_{total}$ 
8.       if  $a$  defensive
9.         if  $a$  chooses to move
10.           $n \leftarrow$  proportionally chosen friendly neighbor
11.           $a$  moves to  $n$ 
12.        end if move
13.      else //  $a$  offensive
14.         $n \leftarrow$  proportionally chosen hostile neighbor
15.         $a$  attacks an animat  $b$  in  $n$  and wins with probability  $\pi_{win}$ 
16.        if  $a$  wins
17.           $a$  kills  $b$ 
18.           $a$  drops  $2p$  pheromone in  $n$ 
19.        end if wins
20.        if  $n$  captured
21.           $a$  moves to  $n$ 
22.           $a$  drops  $p$  pheromone in  $n$ 
23.        end if captured
24.      end if-else defensive
25.    end if activated
26.  end for
27. end Activate

```

Figure 3.2: Animat activation algorithm

of winning an attack (π_{win}) is computed as follows:

$$\begin{aligned}
\phi_{desirability} &= (\rho_{self} - \rho_{foe}) / (\rho_{self} + \rho_{foe}) \\
\phi_{size} &= 2 * (\vartheta_{foe} - \vartheta_{self}) / (\vartheta_{self} + \vartheta_{foe}) \\
\pi_{win} &= 0.5 + \phi_{desirability} + \phi_{size}
\end{aligned}$$

where ρ_{self} and ρ_{foe} are the desirability of the originating attack and the target respectively, ϑ_{self} and ϑ_{foe} are the sizes of the territory of the attacking and defending colony respectively. The value $\phi_{desirability}$ allows the animat from the vertex with a larger desirability value to have a better chance of winning. The value ϕ_{size} grants a small advantage to the animat from a smaller colony, thus promote a balanced partition. The animat activation algorithm is shown in Figure 3.2.

3.2.4 Pheromone Evaporation

While exploring the search space, it is important to avoid premature exploitation of good regions that are found early. There may exist better regions to exploit that could be overlooked because pheromone levels do not allow the animats to find them. To reduce the risk of the colonies from getting stuck in a search space, a certain amount of pheromone is evaporated at the end of each iteration.

Pheromone evaporation is done simply by reducing the amount of pheromone from each colony by a factor of ϵ . However, to avoid disproportionate levels between the largest pheromone value and the lowest one, we do not allow pheromone levels to go below ρ_{min} .

3.2.5 Jolt

In the event that animats fail to produce any new discoveries for several iterations, the animats are not activated. Instead, random jumps in the search space are performed to help the animats get out of a plateau or a local optimum. These jumps are called *jolts* operations. A jolt consists of randomly choosing pairs of vertices owned by different colonies and swapping their owners. Each vertex is chosen to be swapped with probability ν .

3.2.6 Greedy Rebalancing Subroutine

Although the animats are enticed to produce balanced partitions, equal-sized partitions can never be guaranteed. It is plausible to design an algorithm that forces the animats to maintain balanced partitioning at all time, however, such a strict rule may prohibit the animats from exploring the search space. Instead, we allow the animats to produce unbalanced partitions, but we also provide a greedy subroutine to rebalance partitions. This subroutine is randomly called throughout iterations in a round.

The probability that this subroutine is called in an iteration is

$$r(i) = (1 - t(i)) * \varrho,$$

where $t(i)$ is the time function defined in an earlier subsection, and the value of ϱ

is defined in Table 4.1. From the equation above, it is clear that the frequency of invocations of this subroutine increases as the algorithm approaches the end of a set. This strategy helps push the animats to exploit the search space towards the end of a round.

3.2.7 Local Optimization

At the end of a round, the algorithm has identified the best partitioning found within the round. The partition found by the animats is usually near a local optimum. However, because the animats cannot perform deterministic hill-climbing, we perform a local optimization procedure to push the partition to a local optimum. Since a k -way partition has already been obtained, the pairwise Kernighan–Lin algorithm appear to be a laudable choice. The algorithm that we use to optimize the partitioning found within a round is actually a weaker derivation of the original pairwise Kernighan–Lin. This derivation reduces the running time while producing partitions with comparable quality. Similar reduction is also used by Bui and Moon [7]. Before terminating, the best partitioning found throughout the rounds is optimized once more. However, this optimization is performed using the full pairwise Kernighan–Lin algorithm.

3.2.7.1 Kernighan–Lin Algorithm for bisection

To describe our weaker derivation of the algorithm and the extension for optimizing k -way partitions, we must first describe the original Kernighan–Lin algorithm for bisection.

Let (A, B) be a bisection of $G = (V, E)$, such that $A \cup B = V$ and $A \cap B = \emptyset$. Let v be a vertex, and let g_v denotes the reduction in cut size if v is moved to the other side. For vertices $a \in A$ and $b \in B$, let $g(a, b)$ denotes the reduction to the cut size if a and b are swapped. Clearly:

$$g(a, b) = g_a + g_b - 2\delta(a, b)$$

where:

$$\delta(a, b) = \begin{cases} 1 & \text{if } (a, b) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

-
1. Optimize(A, B)
 2. Compute g_a, g_b for each $a \in A$ and $b \in B$
 3. $Q_A \leftarrow \emptyset; Q_B = \emptyset$
 4. for i from 1 to $n/KL_RATIO - 1$
 5. Choose $a_i \in A - Q_A$ and $b_i \in B - Q_B$
 such that $g(a, b)$ is maximized
 6. $Q_A \leftarrow Q_A \cup a_i; Q_B = Q_B \cup b_i$
 7. update g_a for each $a \in A - Q_A$
 8. update g_b for each $b \in B - Q_B$
 9. end for
 10. Choose $k \in \{1, \dots, n/KL_RATIO - 1\}$ such that
 $\sum_{i=1}^k g(a_i, b_i)$ is maximized
 11. Swap the subsets $\{a_1, \dots, a_k\}$
 and $\{b_1, \dots, b_k\}$
 12. end of Optimize

 1. KL(A, B)
 2. Do {
 3. Optimize(A, B)
 4. until (there is no improvement)
 5. end of KL
-

Figure 3.3: KL algorithm for bisection

The Kernighan–Lin algorithm consists of several iterations of the *Optimize* algorithm, as seen in Figure 3.3. Optimization is performed until no improvement can be found. The Optimize algorithm is initiated by identifying pairs of vertices (a, b) with maximum $g(a, b)$. Once a vertex is chosen, it is not be considered again until the next iteration. After these pairs are identified, the algorithm continues by selecting a sequence of pairs $(a_1, b_1), \dots, (a_j, b_j)$ such that $\sum_{i=1}^j g(a_i, b_i)$ is maximized. The iteration ends with the swapping of subsets $\{a_1, \dots, a_j\}$ and $\{b_1, \dots, b_j\}$.

In lines 4 and 10 in the algorithm of Figure 3.3, the original KL requires for KL_RATIO to be k . This means that all possible pairings of (a, b) are considered. In our implementation, we use $KL_RATIO = 3k$ for the weaker optimization, and $KL_RATIO = k$ for the final optimization.

```

1. Pairwise KL( $S_1, \dots, S_k$ )
2.   Do
3.      $P \leftarrow \emptyset$ 
4.     for  $i$  from 1 to  $k$ 
5.       for  $j$  from 1 to  $k$ 
6.         if ( $i \neq j$ )
7.            $u \leftarrow \text{cut\_size}(S_i, S_j)$ 
8.            $P \leftarrow P \cup \{(i, j, u)\}$ 
9.
10.    Sort  $P$  into descending order based on the  $u$  values
11.    for each  $(i, j, u) \in P$  in the sorted ordering
12.      Optimize( $S_i, S_j$ )
13.  until (there is no improvement)

```

Figure 3.4: Pairwise Kernighan–Lin algorithm

3.2.7.2 Pairwise Kernighan–Lin Optimization

To optimize partitions with $k > 2$, we begin by listing all pairings of subgraphs and computing the cut sizes between the subgraphs in the pairs. With each pair, we call the Optimize subroutine shown in Figure 3.3. The ordering of pairs to optimize is determined by sorting the list of pairs according to the size of the cut in descending order. We conjecture that optimizing highly connected pairs first may be more fruitful than optimizing ones with low cut sizes. In fact, we cannot improve the cut size of pairs that are already disjoint. These steps are repeated until no improvements can be made. This algorithm is shown in Figure 3.4

Chapter 4

Results

We tested our algorithm on six classes of widely-used graphs. These graphs differ significantly in their sizes, structures, and connectivities. They provide an ample test bed to measure the performance of graph partition algorithms. The sizes of these graphs range from 100 to 5252, with average degree from 2 to 36.

ABMGP was implemented in C++ and was run on a Pentium Xeon 3.06 GHz processor with 2 GB of RAM. With parameters cited in Table 4.1, we run our algorithm for 100 trials on each graph. Results are shown in Tables 4.2, 4.3, and 4.4.

4.1 Benchmark Graphs

The classes of graphs used to test our algorithm are as follows:

Gn.p: A random graph on n vertices, where an edge is placed between any two vertices with probability p independent of all other edges. The expected vertex degree is $p(n - 1)$.

Un.d: A random geometric graph on n vertices that lie in the unit square and whose coordinates are chosen uniformly from the unit interval. An edge is placed between two vertices if their Euclidean distance is t or less, where the expected vertex degree, d , is given by $d = n\pi t^2$.

Breg.n.b: A random regular graph on n vertices each of which has degree 3, and the optimal bisection size is b with probability $1 - o(1)$.

Cat.n: A caterpillar graph with n vertices. It is created by starting with a straight line, called the spine. To each of the vertices, except the outermost vertices, six new vertices called legs are connected. If the number of vertices in the spine minus the two outermost vertices is divisible by k , the optimum cut size for the k -way partition is $k - 1$.

Rcat.n: A caterpillar graph with n vertices, where each vertex on the spine has \sqrt{n} legs. Similar to *Cat.n* graphs, the optimum cut size for the k -way partition is $k - 1$ if the number of vertices in the spine is divisible by k .

Grid.n.b: A grid graph with n vertices. The optimal cut size for bisection is b .

W - grid.n.b: A grid graph where the two sides are wrapped around. The optimal cut size for bisection is b .

The *Gn.p* and the *Un.d* graphs were originally used by Johnson, et al. [18] to test the performance of simulated annealing algorithm for the bisection problem. The other graphs were initially used by Bui and Moon [7] to test the performance of their hybrid genetic algorithm.

4.2 Comparison

The results of ABMGP for the bisection problem are compared to two readily available benchmark data. Bui and Moon provide results of BFS-GBA in [7], while Bui and Strite provide results of ASGB in [10]. Bui and Moon [7] also provide results for 4-way partition from two algorithms, GMPA and BFS-GMPA. The results described in Table 4.3 are obtained by taking the better results from the two algorithms. The 8-way partition and the 32-way partition are less studied. Comparison data for these problems are only available from Kang and Moon [19], in which CP and GCMA are studied.

For the bisection problem, ABMGP found the best known results for 32 graphs out of the possible 40. This is an improvement over the 27 found by ASBG. ABMGP missed only two optimal results for the two largest graphs from the Caterpillar class, which are difficult for several known algorithms. ABMGP missed another 6 best known values for graphs from the *G* class, which has the least defined

Table 4.1: Parameters used in experiment

PARAMETER	VALUE	DESCRIPTION
γ	20	The number of rounds to be performed
σ	500	The number of iterations per round
ζ_{start}	0.65	* $\sigma \rightarrow$ The iteration number for the inflexion point of time function
ζ_{stop}	1.15	* $\sigma \rightarrow$ The projected iteration number when time function reaches 0
ι	1.8	* $ V \rightarrow$ Number of animats used
ρ_{max}	500	The maximum pheromone a vertex may hold per colony
ρ_{min}	1	The minimum amount of pheromone in a vertex per colony
π	0.9	Probability that an animat is activated
ϵ	0.08	Pheromone reduction rate at the end of an iteration
ϱ	0.2	How often we force the subgraphs to a balanced partition
φ_{count}	20	Number of iterations in a plateau before starting to jolt
φ_{width}	0.025	* $ E \rightarrow$ The size of the plateau window
ν	0.05	The jolt rate

internal structures compared to the other classes. The results found by BFS-GBA are believed, although not proved, to be optimal, as no other algorithm has found any better results since 1996.

ABMGP found 5 new best-known results for the 4-way partition. It also matched 15 other best known results. Four of the five new best values were found for all of the largest *Breg* graphs, while the other one was obtained for a large *Un.d.* Other than the *G* graphs, our results are actually very close to our competitor.

For the 8-way partition, the results of ABMGP were better than GCMA for 14 graphs, while the two tied only on one graph. Eight of these fourteen values were found in the Caterpillar class. The remaining values were found in the *Grid* and the *U* classes. In this comparison, ABMGP generally performed worse than GCMA on all of the *G* and the *Breg* graphs. This finding was rather interesting, considering ABMGP's excellent performance on the *Breg* class for the 4-way partition problem. Generally, the differences of the results between both algorithms were not very significant. These two algorithms consistently perform better than the multistart-CP.

ABMGP once more showed its ability to partition Caterpillar graphs by obtaining 4 new best known results. ABMGP matched the results of GCMA for 7 other graphs and performed worse for the other ones. Other than for the *G* class, the results of ABMGP were reasonably close to the ones found by GCMA. Interestingly, multistart-CP outperformed ABMGP on several graphs. Such finding supports the conjecture that the performance of CP as a global optimization al-

gorithm excels against the pairwise Kernighan–Lin algorithm as k increases. The utilization of CP as a local optimization for our ant system algorithm is obviously an interesting consideration for the future.

Generally, ABMGP excelled in partitioning graphs when the internal structures of the graphs were well defined. This is not a surprising outcome considering that we designed our animats to be territorial. ABMGP’s performance was not laudable for graphs generated in a purely stochastic manner, such as graphs in the G class.

Table 4.2: Results of ABMGP for bisection

Graph	BFS-GBA	ASGB	ABMGP	Avg cut/ sd	Avg time
Breg500.0	0	0	0	0 / 0	2.47
Breg500.12	12	12	12	12 / 0	3.05
Breg500.16	16	16	16	16 / 0	3.02
Breg500.20	20	20	20	20 / 0	3.5
Breg5000.0	0	0	0	0 / 0	3000
Breg5000.16	16	16	16	16 / 0	3000
Breg5000.4	4	4	4	4 / 0	3000
Breg5000.8	8	8	8	8 / 0	3000
G500.005	49	51	51	54.96 / 1.78	4.58
G500.01	218	218	220	226.57 / 2.40	6.1
G500.02	626	626	626	636.26 / 3.05	7.59
G500.04	1744	1744	1744	1762.89 / 6.12	8.92
G1000.0025	95	97	99	105.17 / 3.22	31.93
G1000.005	447	450	451	462.01 / 4.61	39.16
G1000.01	1362	1367	1373	1388.78 / 4.84	47.57
G1000.02	3382	3385	3393	3427.53 / 13.90	58.28
U500.05	2	2	2	5.86 / 2.33	3.91
U500.10	26	26	26	28.93 / 4.37	4.16
U500.20	178	178	178	182.90 / 8.74	5
U500.40	412	412	412	438.78 / 44.48	5.45
U1000.05	1	3	1	10.76 / 4.60	26.29
U1000.10	39	39	39	53.38 / 10.58	28.24
U1000.20	222	222	222	247.33 / 21.17	34
U1000.40	737	737	737	800.47 / 72.30	35.87
Grid1000.20	20	20	20	20 / 0	28.34
Grid100.10	10	10	10	10 / 0	0.39
Grid5000.50	50	50	50	50.31 / 0.87	3000
Grid500.21	21	21	21	22.03 / 1.76	4.16
W-grid1000.40	40	40	40	40 / 0	28.62
W-grid100.20	20	20	20	20 / 0	0.46
W-grid500.42	42	42	42	43.73 / 3.32	4.49
W-grid5000.100	100	100	100	100.5 / 1.36	3000
Cat.352	1	3	1	1.94 / 1.00	1.51
Cat.702	1	3	1	3.02 / 1.17	6.13
Cat.1052	1	7	1	3.92 / 1.37	20.93
Cat.5252	1	14	16	15.17 / 2.55	3000
RCat.134	1	1	1	1 / 0	0.52
RCat.554	1	3	1	1.86 / 0.99	3.43
RCat.994	1	5	1	2.78 / .96	14.32
RCat.5114	1	7	8	6.375 / 2.02	3000

Table 4.3: Results of ABMGP for 4-way partitioning

Graph	BFS-GBA	ABMGP	Avg cut/ sd	Avg Time
Breg500.0	68	68	70.40 / 1.35	4.88
Breg500.12	72	73	75 / 1.2	5.22
Breg500.16	75	75	78.57 / 1.6	5.4
Breg500.20	81	81	85.46 / 1.68	5.4
Breg5000.0	652	628	638 / 6.03	5393
Breg5000.4	658	633	643.25 / 5.67	5754
Breg5000.8	670	638	650.83 / 7.09	5783
Breg5000.16	648	638	648.25 / 6.74	5934
G500.005	86	93	99.10 / 2.05	5.96
G500.01	364	374	384.31 / 4.31	8.08
G500.02	1012	1027	1042.04 / 5.07	9.96
G500.04	2747	2785	2802.07 / 7.20	11.39
G1000.0025	169	179	189.76 / 3.77	42.43
G1000.005	737	756	773.64 / 7.20	55.72
G1000.01	2193	2239	2255.37 / 8.50	69.12
G1000.02	5354	5426	5470.64 / 12.91	81.04
U500.05	9	10	14.60 / 2.38	5.36
U500.10	64	64	69.83 / 8.23	5.48
U500.20	332	332	353.57 / 16.65	6.66
U500.40	1020	1022	1032.04 / 5.68	7.63
U1000.05	7	10	21.27 / 5.51	36.35
U1000.10	84	82	105.63 / 15.65	38.32
U1000.20	467	470	513.12 / 18.76	41.24
U1000.40	1493	1495	1564.80 / 41.56	45.37
Grid100.10	20	20	20 / 0	0.58
Grid500.21	47	47	47.33 / 1.16	5.19
Grid1000.20	62	62	69.22 / 5.85	40.61
Grid5000.50	150	154	171.17 / 11.56	8205
W-grid100.20	40	40	40 / 0	0.71
W-grid500.42	86	92	95.12 / 1.21	5.78
W-grid1000.40	84	84	86.96 / 1.72	37.5
W-grid5000.100	200	208	238.33 / 32.60	8265
Cat.352	9	9	9.28 / 0.48	2.02
Cat.702	3	4	5.95 / 1.55	9.01
Cat.1052	10	10	13.16 / 1.36	29.61
Cat.5252	10	27	33.45 / 3.18	5375
RCat.134	3	3	3.0 / 0	0.88
RCat.554	3	3	3.43 / .59	4.65
RCat.994	3	3	4.22 / .98	21.8
RCat.5114	4	8	11.65 / 1.94	4568.13

Table 4.4: Results of ABMGP for 8-way partitioning and 32-way partitioning

Graph	8-way partition				32-way partition			
	M-CP	GCMA	ABMGP	Avg / sd	M-CP	GCMA	ABMGP	Avg / sd
Breg500.0	136	116	121	127.98 / 2.24	230	222	242	246.27 / 2.43
Breg500.12	144	118	127	132.27 / 2.75	228	222	236	241.16 / 2.26
Breg500.16	147	122	128	132.27 / 3.14	231	222	239	244.06 / 2.30
Breg500.20	151	128	135	142.71 / 3.98	236	231	249	253.68 / 1.85
Breg5000.0	1690	1096	1140	1158.65 / 10.05	2031	1720	1947	1983.86 / 12.36
Breg5000.4	1735	1093	1138	1161.03 / 9.05	2032	1725	1949	1988.42 / 12.54
Breg5000.8	1675	1098	1152	1169.73 / 8.4	2027	1737	1969	1996.47 / 12.77
Breg5000.16	1721	1077	1142	1157.84 / 8.10	2025	1693	1953	1980.96 / 12.04
G500.005	130	116	128	133.04 / 2.41	195	181	195	199.194 / 2.06
G500.01	489	468	488	496.22 / 3.62	652	631	681	689.32 / 3.56
G500.02	1283	1257	1290	1299.84 / 3.96	1608	1587	1652	1663.21 / 5.21
G500.04	3384	3354	3391	3417.15 / 8.51	4076	4043	4125	4152.552 / 8.94
G1000.0025	262	220	242	255.10 / 4.50	371	326	350	361.79 / 4.09
G1000.005	997	942	983	996.36 / 5.99	1288	1222	1323	1347.27 / 6.70
G1000.01	2801	2729	2789	2809 / 7.90	3454	3370	3496	3526.23 / 12.67
G1000.02	6638	6531	6609	6650.41 / 14.40	7949	7839	8009	8043.77 / 12.39
U500.05	53	22	17	22.19 / 3.32	130	115	116	122.73 / 2.95
U500.10	163	143	144	150.04 / 5.82	550	532	537	550.209 / 4.96
U500.20	613	611	620	654.62 / 12.47	1831	1825	1829	1851.95 / 7.36
U500.40	1867	1865	1857	1867.71 / 4.17	5337	5328	5356	5381.34 / 10.34
U1000.05	140	57	27	37.80 / 5.55	211	130	137	150.30 / 5.95
U1000.10	349	187	176	199.82 / 11.40	653	577	606	631.90 / 12.17
U1000.20	855	812	823	849.76 / 20.57	2441	2367	2445	2479.806 / 20.91
U1000.40	2564	2562	2570	2584.69 / 5.21	7370	7329	7370	7404.209 / 23.71
Grid100.10	40	40	38	38.24 / 0.43	108	108	108	108.58 / 0.62
Grid500.21	87	86	88	93.45 / 1.66	225	220	220	231.29 / 3.71
Grid1000.20	114	114	114	119.31 / 2.25	327	314	326	331.85 / 2.95
Grid5000.50	311	250	272	297.53 / 16.08	918	659	732	767.04 / 10.20
W-grid100.20	60	60	58	59.66 / .82	128	128	128	128.75 / 0.5
W-grid500.42	132	131	138	141.74 / 1.20	268	266	270	278.60 / 3.42
W-grid1000.40	180	176	191	197.16 / 2.81	392	384	396	406.22 / 3.43
W-grid5000.100	415	400	425	441.07 / 6.24	1069	820	883	914.12 / 8.04
Cat.352	29	19	17	17.0 / 0	90	85	85	85 / 0
Cat.702	58	31	17	17.85 / .80	90	60	31	35.44 / 7.23
Cat.1052	83	50	15	16.76 / 1.07	148	101	72	75.76 / 1.73
Cat.5252	429	204	41	50.46 / 3.94	563	377	110	137.88 / 10.79
RCat.134	27	27	25	25.16 / .37	91	91	91	91 / 0
RCat.554	14	9	7	7.0 / 0	159	159	159	159 / 0
RCat.994	26	15	7	7.03 / 0.19	31	31	31	31 / 0
RCat.5114	124	45	14	16.80 / 1.68	495	489	455	456.30 / 4.5

Chapter 5

Conclusion

A new algorithm for the Multiway Graph Partitioning has been developed adhering to ideas of Ant Based optimization. Agents, called animats, from k colonies are used to divide the graph into k territories, which resembles a k -way partitioning. Our algorithm utilizes the pairwise Kernighan–Lin algorithm to optimize the partitioning created by the animats. We compared our algorithms with several algorithms for $k = 2, 4, 8,$ and 32 . Our results are generally quite competitive, although we do not excel in every case. Nonetheless our algorithm obtained several results that are better than the previous best known.

Further works may include the utilization of different local optimization methods, including CP. This may improve the performance of the algorithm for problems with higher k values. One may also consider investigating different animat activation rules to improve the result of this algorithm in general.

References

- [1] C.J. Alpert and A.B. Kahng, “Recent Directions in Netlist Partitioning: A Survey,” *Integration: The VLSI Journal*, 19(1-2), 1995, pp. 1–81.
- [2] R. Battiti and A. Bertossi, “Greedy, Prohibition, and Reactive Heuristics for Graph Partitioning,” *IEEE Transactions on Computers*, 48(4), April 1999, pp. 361–385.
- [3] E. Bonabeau, M. Dorigo and G. Theraulaz, “Inspiration for Optimization from Social Insect Behavior,” *Nature*, Vol. 406, July 6, 2000, pp. 39–42.
- [4] T.N. Bui, S. Chaudhuri, F.T. Leighton, and M. Sipser, “Graph Bisection Algorithms With Good Average Case Behavior,” *Combinatorica*, 7(2), 1987, pp. 171–191.
- [5] T.N. Bui and C. Jones, “Finding Good Approximate Vertex and Edge Partitions is NP-Hard,” *Information Processing Letters* 42 (1992), pp. 153–159.
- [6] T.N. Bui and C. Jones, “A Heuristic for Reducing Fill-In in Sparse Matrix Factorization,” *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, 1993, pp. 445–452.
- [7] T.N. Bui and B.R. Moon, “Genetic Algorithm and Graph Partitioning,” *IEEE Transactions on Computers*, 45(7), 1996, pp. 841–855.
- [8] T.N. Bui and A. Peck, “Partitioning Planar Graphs,” *SIAM Journal of Computing*, 21(2), April 1992, pp. 203–215.
- [9] T.N. Bui and J.R. Rizzo, “Finding Maximum Cliques with Distributed Ants,” *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004, pp. 24–35.

- [10] T.N. Bui and L.C. Strite, “An Ant System Algorithm for Graph Bisection,” Proceedings of Genetic and Evolutionary Computation Conference, 2002, pp. 43–51.
- [11] T.N. Bui and G. Sundarraj, “Ant System for the k -Cardinality Tree Problem,” Proceedings of the Genetic and Evolutionary Computation Conference, 2004, pp. 36–47.
- [12] J.P. Cohoon, W.N. Martin, and D.S. Richards, “A Multi-Population Genetic Algorithm for Solving the k -Partition Problem on Hypercubes,” Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 244–248.
- [13] A. Coloni, M. Dorigo, and V. Maniezzo, “Distributed Optimization by Ant Colonies,” in Proceedings of the First European Conference on Artificial Life, New York: Elsevier, 1991, pp. 134–142.
- [14] M. Dorigo and L. Gambardella, “Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem,” IEEE Transactions on Evolutionary Computation, 1(1), April 1997, pp. 53–66.
- [15] M. Garey and D.S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness,” San Francisco: Freeman, 1979.
- [16] B. Hendrickson and R. Leland. “A Multilevel Algorithm For Partitioning Graphs,” Technical Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM 87185-1110, 1993.
- [17] H. Inayoshi and B. Manderick, “The Weighted Graph Bi-Partitioning Problem: A Look at GA Performance,” Parallel Problem Solving in Nature, 1992, pp. 617–625.
- [18] D.S. Johnson, C. Aragon, L. McGeoch, and C. Schevon, “Optimization by Simulated Annealing: An Experimental Evaluation, Part 1, Graph Partitioning,” Operations Research, Vol. 37, 1989, pp. 865–892.

- [19] S.J. Kang and B.R. Moon, “A Hybrid Genetic Algorithm for Multiway Graph Partitioning,” *Proceedings of Genetic and Evolutionary Computation Conference*, 2000, pp. 159–166
- [20] G. Karypis and V. Kumar, “Analysis of Multilevel Graph Partitioning,” *Proceedings of the 7th Supercomputing Conference*, San Diego, 1995.
- [21] B. Kernighan and S. Lin, “An Efficient Heuristic Procedure for Partitioning Graph,” *The Bell System Tech J.*, 49(2), 1970, pp. 291–307.
- [22] P. Kuntz and D. Snyers, “Emergent Colonization and Graph Partitioning,” *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, 1994, pp. 494–500.
- [23] P. Kuntz, P. Layzell and D. Snyers, “A Colony of Ant-like Agents for Partitioning in VLSI Technology,” *Proceedings of the Fourth European Conference on Artificial Life*, 1997, pp. 417–424.
- [24] T. Leighton and S. Rao, “An Approximate Max-Flow Min-Cut Theorem for Multicommodity Flow Problems with Applications to Approximation Algorithms,” *Proceedings of the Symposium on Foundation of Computer Science*, 1988, pp. 422–431.
- [25] V. Maniezzo and A. Carbonaro, “Ant Colony Optimization: An Overview”, in *Essays and Surveys in Metaheuristics*, C. Ribeiro (ed.), Kluwer Academic Publishers, 2001, pp. 21–44.
- [26] A. Pothen, “Graph Partitioning Algorithms with Applications to Scientific Computing,” in *Parallel Numerical Algorithms*, D. E. Keyes, A. H. Sameh and V. Venkatakrisnan (eds.), Kluwer Academic Press, 1997, pp. 323–368.