

Ant Colony Multi-Optimization Algorithm for Circuit Bi-partitioning

BY

Emran A. Ba-Abbad

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

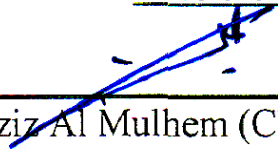
June 2005

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA


DEANSHIP OF GRADUATE STUDIES

This thesis, written by Emran A. Ba-Abbad
under the direction of his Thesis Advisor and approved by his Thesis
Committee, has been presented to and accepted by the Dean of Graduate
Studies, in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN COMPUTER ENGINEERING

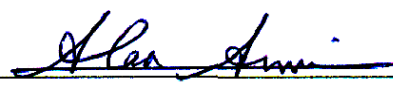
Thesis Committee




Dr. Abdulaziz Al Mulhem (Chairman)



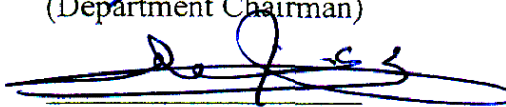
Dr. Sadiq M. Sait (Member)



Dr. Alaaeldin Amin (Member)



Dr. Abdulaziz Al Mulhem
(Department Chairman)



Dr. Mohammad Al Ohali
(Dean of Graduate Studies)

Date

August 2, 2005



Heartily dedicated to my family and especially to
my dear Mother, Father, and wife

ACKNOWLEDGEMENT

All praise to Allah, the most Merciful, who enabled me to complete my thesis work. I make a humble effort to thank Allah for his endless blessings on me, as His infinite blessings cannot be thanked for. Then, I pray Allah to bestow peace on his last prophet Muhammad (Sal-allah-'Alaihe-Wa-Sallam) and on all his righteous followers till the Day of Judgment.

I pay a hearty tribute to all my family members, especially to my parents, who guided me during all my life endeavors. Their love and support motivated me to continue my education and achieve higher academic goals. Without their moral support and sincere prayers, I would have been unable to accomplish this task. Next, I am grateful to my thesis advisor Dr. Abdulaziz Al Mulhem and my thesis committee member Dr. Sadiq M. Sait and Dr. Alaaeldin Amin.

I acknowledge the academic and computing facilities provided by the Computer Engineering Department of King Fahd University of Petroleum & Minerals (KFUPM).

I also appreciate the friendly support from all my friends and colleagues at Saudi Aramco and KFUPM. In particular, I want to thank Husam Ben Siddeeq, Mansour Al Ansari, and Nasif Dawd.

THESIS ABSTRACT

Name: Emran A. Ba-Abbad
Title: Ant Colony Multi-Optimization Algorithm for Circuit Bi-Partitioning.
Major Field: Computer Engineering
Date of Degree: June, 2005

The acceleration of the product to market cycle of VLSI based technology products dictates continuously refining design and implementation methodologies. Circuit partitioning is a physical design problem in which a given circuit is divided into segments meeting some constraints and objectives. The circuit partitioning problem is NP hard; that means for this class of problems no algorithm of polynomial complexity could be found. In this thesis work, a biologically inspired heuristic (ant colony) is used to solve such problem. The ant colony implemented is closely rooted at the biological and behavioral model of the real social insects. It is a non-deterministic heuristic and could be used as both constructive and iterative. The solution uses many ants of simple nature and limited memory requirements. The intelligence of this heuristic is not portrayed by individual ants, but rather is expressed by the colony as a whole. Careful presentation of the problem to the ant colony model facilitates the close biological solution derivation. The solutions obtained by this heuristic produced good result compared to those of other established heuristics.

MASTER OF SCIENCE DEGREE
King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia
June, 2005

خلاصة الرسالة

الإسم: عمران عبدالله سالم باعبداد

العنوان: خوارزمية مستعمرة النمل لتقسيم الدوائر الإلكترونية

الدرجة: الماجستير في العلوم

التخصص الرئيسي: هندسة الحاسوب

تاريخ التخرج: حزيران "يونيو" 2005

يتطلب تحسين التصميم وطرق التطبيق بشكل (VLSI) تعجيل تصنيع وإنتاج الأنظمة المتكاملة مستمر. إن تقسيم الدوائر الإلكترونية يمثل معضلة في مرحلة التصميم الفيزيائي ويستوجب تحقيق أهداف والخضوع لشروط معينة. تصنف معضلة تقسيم الدوائر تحت بند المعضلات الصعبة التي لم يمكن حلها ضمن متعددة حدود زمنية. في هذه الرسالة تستعمل خوارزمية مستعمرة النمل المستلهمة من نظام الحشرات الحية لحل معضلة التقسيم المذكورة. خوارزمية مستعمرة النمل المذكورة تحاكي بدقة طريقة تعامل الحشرات الاجتماعية الحية وتصنف هذه الخوارزمية على أنها من النوع المبهم ويمكن استعمالها بطريقة بنائية أو تكرارية. هذه الخوارزمية تستعمل العديد من النمل المبسط الذي يستهلك ذاكرة محدودة. إن ذكاء هذه الخوارزمية لا يتمثل في أفراد النمل، ولكنه يتشكل من المستعمرة بشكل إجمالي. إن دقة المؤلف بين المعضلة و طراز مستعمرة النمل يساعد على الإلتزام بطريقة حل محاكية لتلك الموجودة في أنظمة الحشرات الحية. مقارنة الحلول الناتجة عن هذه الخوارزمية بمثيلاتها المشهورة عالمياً أثبتت نتائج جيدة.

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
حزيران "يونيو" 2005

Contents

ACKNOWLEDGEMENT.....	iv
THESIS ABSTRACT.....	v
List of Figures.....	xi
List of Tables.....	xiii
Chapter 1 Introduction.....	1
Chapter 2 Literature Review	4
2.1 Introduction to Circuit Bi-partitioning	4
2.2 Approaches to Partitioning.....	6
2.2.1 Move-based Approaches:.....	6
2.2.2 Geometric Representation Approaches:.....	9
2.2.3 Combinatorial Formulations:.....	9
2.2.4 Clustering Approaches:	11
2.3 Performance-Driven Circuit Partitioning in Physical Level	15
2.4 The Conventional Ant Colony Optimization Algorithm.....	18
Chapter 3 Problem Formulation and Solution Methodology	25
Chapter 4 The Ant Colony Heuristic.....	31
4.1 Habitat	32
4.2 The colony.....	34
4.3 Ant.....	37

4.3.1 Scout data structure	39
4.3.2 Vision data structure.....	39
4.3.3 Track data structure	39
4.3.4 Steps Counter variable.....	40
4.3.5 Minimum Carried Weight	40
4.3.6 Navigate data structure.....	40
4.4 Ant’s lifecycle	43
4.4.1 Foraging.....	43
4.4.2 Storing	46
4.4.3 Update habitat-cell pheromone “forage”	53
4.4.4 Update ant’s scout information	55
4.4.5 Update ant’s vision information.....	56
4.4.6 Pheromone evaporation.....	56
4.4.7 Decide on next habitat cell to move to “navigate”	60
4.4.8 Move ant.....	61
4.4.9 Increment ant’s steps counter	62
4.4.10 Re-Orient ant	62
4.4.11 Select lightest weight gate.....	63
4.4.12 Set habitat cell help flag.....	64
4.4.13 Pick the gate up	64
4.4.14 Carry gate in the ant’s bag.....	67

4.4.15 Update ant's min carried weight.....	67
4.4.16 Increment ant's steps counter	68
4.4.17 Flip ant's direction.....	68
4.4.18 Update habitat cell pheromone "storage"	69
4.4.19 Update habitat cell breadcrumbs info.....	71
4.4.20 Move out of the colonies	72
4.4.21 Get a colony vacant bag	73
4.4.22 Store the carried gates	73
4.4.23 Put down the carried gates.....	74
4.4.24 Social Ants.....	74
4.5 Heuristic Related Issues	76
4.5.1 Controlling Greediness.....	76
4.5.2 Pheromone and Cost Optimization.....	79
4.5.3 Implementing Different Cost Measures	80
4.5.4 Hill Climbing.....	81
Chapter 5 Cost functions	84
5.1 Power cost function.....	85
5.2 Delay cost function.....	87
5.3 Cut-set cost function.....	87
5.4 Imbalance constraint.....	87
5.5 Fuzzy goodness evaluation.....	89

Chapter 6 Ant Colony Deployment.....	91
Chapter 7 Experiments and Results.....	99
7.1 Circuit details.....	100
7.2 Results and comparisons	106
7.3 Trend analysis.....	114
7.4 Heuristic parameters	123
Chapter 8 Conclusion	126
Appendix A	131
References	139

List of Figures

Figure 1: An 8-module example, (a) an agglomerative and (b) a hierarchical construction.	13
Figure 2: Double bridge experiment. (a) Ants start exploring the double bridge. (b) Eventually most of the ants choose the shortest path [38].	20
Figure 3: Ant Colony Algorithm.	22
Figure 4: A habitat grid cell structure and methods.	33
Figure 5: The ant's colony.....	35
Figure 6: the ant's data structure	38
Figure 7: Flow chart of the ant's "forage" lifecycle.....	51
Figure 8: Flow chart of the ant's "storage" lifecycle.	52
Figure 9: Nonlinear pheromone increase during foraging.	54
Figure 10: Nonlinear pheromone evaporation.....	58
Figure 11: Pick up probability.....	66
Figure 12: Storage pheromone increment.	70
Figure 13 Power dissipation calculation function.	86
Figure 14: Maximum path delay calculation function.	88
Figure 15: One gate assignment configuration.....	101
Figure 16: Another gate assignment configuration.	102
Figure 17: the minimum cutset nets of circuit s298.	105

Figure 18: Cutset, delay, power, fitness, and imbalance of s298.	115
Figure 19: Cutset, delay, power, fitness, and imbalance of s386.	116
Figure 20: Cutset, delay, power, fitness, and imbalance of s641.	117
Figure 21: Cutset, delay, power, fitness, and imbalance of s832.	119
Figure 22: Cutset, delay, power, fitness, and imbalance of s953.	120
Figure 23: Cutset, delay, power, fitness, and imbalance of s2081	121
Figure 24: Using simulated evolution to calculate the parameter values.	125
Figure 25: Random search pattern.	127
Figure 26: Definite pattern of nondeterministic hill climbing search heuristic.	128
Figure 27: Cutset, delay, power, fitness, and imbalance of s298	132
Figure 28: Cutset, delay, power, fitness, and imbalance of s386	133
Figure 29: Cutset, delay, power, fitness, and imbalance of s641	134
Figure 30: Cutset, delay, power, fitness, and imbalance of s832	135
Figure 31: Cutset, delay, power, fitness, and imbalance of s953	136
Figure 32: Cutset, delay, power, fitness, and imbalance of s2081	137

List of Tables

Table 1: Circuits characteristics.	103
Table 2: Gates characteristics.	104
Table 3: Ant Colony results.	107
Table 4: Original values of the circuits.	109
Table 5: A Comparison between the quality of the best solutions obtained from GA by performing SOP for cut-only and MOP [9].	110
Table 6: A Comparison between the qualities of the best solutions obtained from TS by performing SOP for cut-only and MOP [9].	111
Table 7: Best solutions obtained from SimE by performing MOP [9].	112
Table 8: The heuristic parameters for small and medium circuits.	124
Table 9: Results summary for perfect partition balance (0%) for SOP and MOP.	138

Chapter 1

Introduction

Since the beginning of VLSI production acceleration, many conflicting design decisions have to be made. Optimizing one design aspect may lead to performance degradation in others. Attempts are continuously being made to come up with a method to reach a compromising design decision such that the complete product performance is acceptable in the ever increasing marketing competition. In addition, the time to market is of utmost importance. The marketing window has become very narrow such that delivering the product to market in the right time is a company survival issue.

Circuits are constituted by the interconnection of logic gates and sequential elements. As the functional complexity performed by the circuit increases, the circuit components and interconnects increase. Performance driven large circuits have to be divided in such a way to minimize critical path delay, power, and number of partition interconnects. The partitioning process is NP-Hard [7].

No theoretically proven method was found to solve this class of problems in polynomial time. Instead, “heuristics” are used to produce solutions with acceptable quality.

Heuristics are “smart” methods that are known, rather than proven, to find a solution that most of the time is not optimum. Measurable criteria should be devised to evaluate the produced solution and an acceptance level classifies the obtained results.

Heuristics could be grouped in two classes, constructive and iterative [7]. Constructive heuristics build the solution from scratch, whereas iterative ones “iterate” many times in improving a previously obtained one. Furthermore, heuristics can be identified either as deterministic or non-deterministic. Deterministic heuristics will attempt to solve a given problem in organized, but repeatable steps. Several attempts to solve the problem will definitely lead to an identical solution. On the other hand, non-deterministic heuristics use probability in defining the search path; this guarantees a fresh search path each time the heuristic is run. Non-deterministic heuristics although more complex, lead to a better quality solution and in a shorter time; than that produced by the deterministic counterpart. This is attributed to their nature of accepting bad solutions, whenever is advisable, hoping that the investigated path may lead to a much better solution at the end [8].

The solution is evaluated against conflicting objectives while enforcing important requirement constraints. The idea is to find a unified acceptance measure that integrates the objectives and the constraints while preserving the relative importance of each. The use of fuzzy rules devised to give each design aspect an importance weight such that the complete design can be evaluated. Each conflicting optimization criterion is assigned a

value reflecting its deviation from the acceptable level, the fitness membership value. This methodology is widely applied and was known to produce the sought after acceptance measure [9].

The following chapter summarizes the reviewed previous work related to the solution methodology implemented in this thesis.

Chapter 2

Literature Review

2.1 Introduction to Circuit Bi-partitioning

The essence of netlist partitioning is to divide a system into clusters such that the number of inter-cluster connections is minimized. The partitioning task is ubiquitous to many subfields of VLSI CAD. Most top-down hierarchical (i.e., divide and conquer) approach in system design must rely on some underlying partitioning technique. There are several reasons why partitioning has recently emerged as a critical step in many phases of VLSI system synthesis, and why the past several years have seen so much research activities on this subject [10, 14].

Partitioning heuristics are used to address the increasing complexity of VLSI design; systems with several million transistors are now common, presenting instance complexities that are unmanageable for existing logic level and physical level design

tools. Partitioning divides a system into smaller, more manageable components; the number of signals which pass between the components corresponds to the interactions between the design sub-problems. In a top-down hierarchical design methodology, decisions made early in the system synthesis process (e.g., at the system and chip levels) will constrain succeeding decisions. Thus, the feasibility not to mention the quality of automatic placement, global routing, and detailed routing will somewhat depend on the quality of the partitioning solution.

A bottom-up clustering approach may also be applied to reduce design complexity, typically in cell-level or gate-level layout. The current emphasis on a quick turn-around design cycle reinforces the need for reliable and effective algorithms. Partitioning heuristics also have a great impact on system performance as designs become dominated by interconnects.

Finally, partitioning heuristics affect the layout area; wires between clusters at higher levels of the hierarchy will tend to be longer than wires between clusters at lower levels, and total wire length is directly proportional to layout area due to minimum wire spacing design rules. The traditional minimum-cut objective is natural for this application, if the layout area is divided into a dense uniform grid. Total wire length can be expressed in grid units or equivalently as the sum over all gridlines of the number of wires crossing each gridline. This view can also improve auto-routability since it suggests reducing the

wire congestion in any given layout region. All of these considerations motivate the development of netlist partitioning algorithms that identify interconnection and communication structure in a given system design. In the following section, we discuss different approaches to partitioning which consider only cutset as an objective.

2.2 Approaches to Partitioning

As the partitioning problem is NP-complete [15, 11], an exact (globally optimal) solution cannot be found in a feasible amount of time. Therefore, heuristics must be used to reach a good solution within reasonable time limits. Major research directions in netlist partitioning can be categorized into four types of approaches:

- Move-based Approaches [16, 17, 18, 19].
- Geometric Representation Approaches [20, 21, 22].
- Combinatorial Approaches [23].
- Cluster-based Approaches [24].

2.2.1 Move-based Approaches: This category explores the solution space by moving from one solution to another. Greedy and iterative exchange [12] approaches are most common. These always try to make the best move, but can easily be trapped in local minima. To avoid this behavior, many other strategies have been proposed including Stochastic Hill-

Climbing (Simulated Annealing), Evolutionary Algorithms, and the Multi-start strategy [25]. A partitioning approach is move-based if it iteratively constructs a new candidate solution based on two considerations:

- A neighborhood structure is defined over the set of feasible solutions.
- The previous history of optimization is maintained.

The first consideration requires the notion of a local perturbation of the current solution; this is the heart of the move-based paradigm. The type of perturbation used determines the topology over the solution space, known as the neighborhood structure. For the objective function to be smooth over the neighborhood structure, the perturbation (also known as a neighborhood operator) should be small and local.

Typical neighborhood operators for partitioning include swapping a pair of modules or shifting a single module across a cluster boundary. For example, two partitioning solutions are neighbors under the pair-swap neighborhood structure if one solution can be derived from the other by swapping two modules between clusters. In general, the solution space is explored by repeatedly moving from the current solution to a neighboring solution. With respect to previous history, some approaches are memoryless, e.g., a simple greedy method might rely only on the current solution to generate the next solution. On the other hand, methods such as Kernighan-Lin [12] or Fiduccia-Mattheyses [13] implicitly remember the entire history of the pass. Hybrid genetic-local search or

Tabu Search approaches must also remember the lists of previously seen solutions. Move-based approaches dominate in both the literature and industry practices for several reasons. First, they are generally very intuitive; the logical way of improving a given solution is to repeatedly make it better via small changes, such as moving individual modules. Second, iterative algorithms are simple to describe and implement. For this reason, the bi-partitioning method of Fiduccia-Mattheyses [13] and the Multi-way partitioning method of Sanchis [26] are standards against which nearly all other heuristics are measured. Third, the move-based approach encompasses more sophisticated strategies for exploring the solution space e.g., Simulated Annealing, Tabu Search, and Genetic Algorithms which yield performance improvements over greedy iterative methods while retaining the intuitiveness associated with local search.

Finally, the move-based approach is independent of the nature of the objective function that is used to measure the solution quality. While other approaches might require the objective to be of a particular form, or a relatively simple function of solution parameters, the move-based approach can flexibly incorporate arbitrary constraints (e.g., on critical path delays or I/O utilization). Thus, the move-based approach has been applied successfully to virtually every known partitioning formulation. The main algorithms, included in this approach are:

- Fiduccia-Mattheyses Algorithm [13].

- Kernighan-Lin Algorithm [12].
- Sanchis' Multi-Way Partitioning Algorithm [26].
- Simulated Annealing Algorithm [12].
- Tabu Search [28].
- Genetic Algorithms [29].

2.2.2 Geometric Representation Approaches: A geometric representation of the circuit netlist can provide a useful basis for a partitioning heuristic. These approaches discuss finding a geometric representation of a graph or hypergraph and applying geometric algorithms to find a partitioning solution. This means that the circuit netlist is embedded in some type of geometry, e.g., a 1-dimensional linear ordering or a multi-dimensional vector space; the embeddings are commonly constructed using Spectral methods [21]. Spectral methods are of primary importance in constructing geometric representations.

2.2.3 Combinatorial Formulations: An approach is classified under this category if the partitioning problem can be transformed into some other “classic” type of optimization problem (e.g., maximum flow, mathematical programming, graph labeling etc.). These approaches are promising since complex formulations that include timing, module pre-assignment, replication, and other hard constraints can often readily be expressed in terms of a mathematical program or flow networks. In addition, the constantly changing user

requirements for solution quality and runtime, and the improved computing platforms, have made such approaches more practical.

It is possible, that the next frontier of optimization strategies for CAD applications will involve large-scale mathematical programming instances, including mixed integer-linear programs that require branch-and-bound search. Following are some of the methods employed under this category:

- Min-Delay Clustering by Graph Labeling, first considered by Lawler et al. [30], assumes that the module and intra-cluster delays (i.e., delays between modules in the same cluster) are negligible compared to inter-cluster delay that results from placing clusters onto different chips.
- Mathematical Programming optimizes an objective function subject to inequality constraints on the variables (an equality constraint can be captured by two inequality constraints). A linear program (LP) requires every equation to be linear in terms of each variable. An LP can be solved in an average case polynomial time using the simplex method. An integer linear program (ILP) is an LP with the additional constraint that the variables must take on integer values; solving general ILP instances is NP-Hard. A quadratic program (QP) [23] is an LP with an

objective that is quadratic in the variables, and a Quadratic Boolean program (QBP) additionally restricts the variables to 0-1 values.

- Fuzzy partitioning or the Fuzzy k-means (FKM) [31] algorithm is a well-known optimization technique for clustering problems that arise in such fields as geological shape analysis, medical diagnosis, etc. The problem formulation generally involves clustering data points in multi-dimensional space. A fuzzy partitioning [32] can partially assign a module to several clusters. FKM begins with an initial fuzzy partitioning X , and then iteratively modifies X to optimize the objective function.

2.2.4 Clustering Approaches: A clustering solution is typically used to induce a smaller and more tractable problem instance. Many clustering algorithms utilize a bottom-up approach where each module initially belongs to its own cluster. Clusters are gradually merged or grown until the desired decomposition is found. Bottom-up approaches are agglomerative, if new clusters are formed one at a time and hierarchical if several new clusters may be formed simultaneously.

The agglomerative approach [33] begins with n -way clustering (where each module is a cluster) and iteratively constructs the new clusters by choosing a pair of clusters, and merging them into a new cluster. The criterion for choosing the two clusters is what

distinguishes among agglomerative variants, e.g., [33] merges the two clusters that minimize the diameter of the newly formed cluster. This approach is applied to hypergraphs by picking a random net (perhaps with size-dependent probability) and contracting two random incident clusters (or all incident clusters). An alternative greedy approach would be to simply merge the two clusters with high connectivity.

Generally, agglomerative methods will not be very efficient. Finding the best pair of clusters to merge require $O(k^2)$ time, unless a list of cluster merging costs is stored and updated (which will likely require $O(n^2)$ space). An alternative strategy is to find many good clusters to merge, and then perform all merges simultaneously; this is called hierarchical strategy.

The difference between agglomerative and hierarchical strategies is illustrated for the 8-module example in Figure 1. In (a), the diagram reveals the order in which clusters are merged; each dotted horizontal line is a level in the hierarchy, and an agglomerative algorithm will have $n-1$ levels.

Fig. 1(b) shows a hierarchical algorithm that simultaneously merges as many cluster pairs as possible, yielding a hierarchy with $\lceil \log n \rceil$ levels.

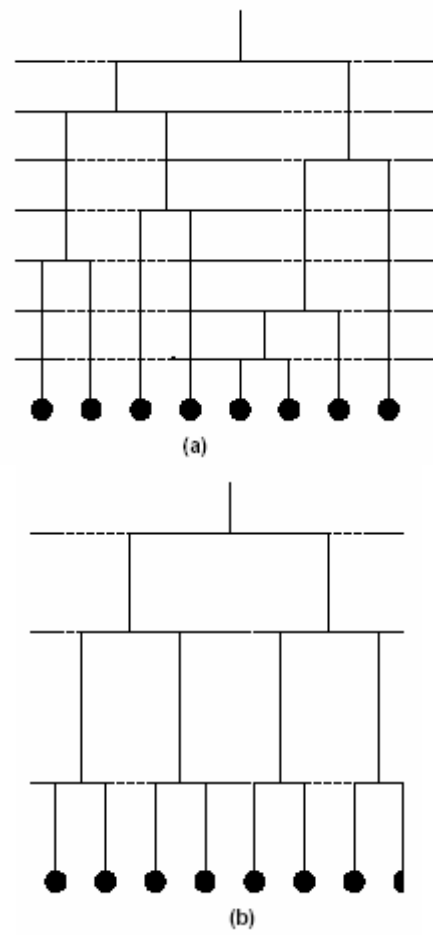


Figure 1: An 8-module example, (a) an agglomerative and (b) a hierarchical construction.

Other intuitive approaches involve random walks, iterative peeling of clusters, vertex orderings, and simulated annealing. Another set of approaches are specific to (acyclic) combinational Boolean networks [34, 35]. However, move-based approaches, and iterative improvement in particular, are the most common partitioning algorithms in current CAD tools. Clustering techniques are motivated by the fact that a common weakness of move-based approaches is that the solution quality is not stable, i.e., unpredictable. It is highly dependent on the starting solution and the choices taken during the optimization process.

Hagen et al. [16] used a random multi-start approach to FM, where the algorithm is executed many times from random starting points and returning the best solution found. However, it may need hundreds of runs to achieve stable performance.

The hierarchical clustering algorithm [33] groups a set of objects according to some measure of closeness. Two closest objects are clustered first and considered to be a single object for future clustering. Clustering continues by grouping two individual objects, or an object or cluster with another cluster on each iteration.

The process stops when a single cluster is generated and a hierarchical cluster tree is formed. A cut-line through the tree indicates a set of segments in a partition. Clustering can be integrated into other move-based algorithms. The simplest way to incorporate a

clustering solution into a bi-partitioning heuristic is via the two-phase approach, i.e., to run FM on the contracted netlist, and then use the result as the starting solution of a second run on the flattened netlist [36]. However, more sophisticated techniques may be preferable.

2.3 Performance-Driven Circuit Partitioning in Physical Level

This section reviews some recent approaches for performance driven partitioning (power, delay) in CMOS VLSI circuits. Different techniques are applicable and have been reported at different steps of the VLSI design process [10]. In standard CMOS VLSI circuits, switching activity of circuit nodes is responsible for most of the power dissipation. It is reported in [37] that this switching activity contributes up to 90% of the total power dissipation in the circuit. Therefore, most of the reported techniques focus on this aspect [38].

A reasonable number of techniques aiming at low power objective are proposed for all phases in physical design including partitioning of circuit, floor-planning, placement and routing [7].

For the partitioning phase, two low-power oriented techniques based on Simulated Annealing (SA) algorithm have recently been presented in [27]. One of the algorithms

uses the Shannon expansion-based scheme and the other uses the Kernel-based scheme. These algorithms partition the circuit into a number of sub-circuits and a single sub-circuit needs to be active at a particular time. In this way, the unnecessary signal transitions are prevented. Circuit partitioning is performed by using an adaptive SA algorithm. The cost function is modeled for low-power consumption under given area constraint. A partitioning solution is obtained by recursive bi-partitioning of the circuit and the solution space is represented as a binary tree. The stopping criterion used is non-improvement in the solution for a constant number of moves. The performance of the algorithm is evaluated by its application to MCNC benchmark circuits and its comparison with the results of Synopsis design analyzer show an 8.7% power reduction over the latter without allowing any increase in the layout area.

An optimal delay partitioning algorithm targeting low power is proposed in [14] which provides a formal mechanism to implicitly enumerate the alternate partitions and selects a partition that has the same delay but less power dissipation. One disadvantage of this algorithm is that the runtime is one to two orders of magnitude higher than that of Lawler's clustering algorithm [30]. Another disadvantage of this enumeration technique is that as the size of the circuits grows, the algorithm runtime will increase sharply hence this technique is not suitable for industries seeking a faster time to design and market the chips.

A circuit partitioning algorithm under path delay constraint is proposed in [30]. The proposed algorithm consists of the clustering and iterative improvement phases. In the first phase, the problem size is reduced using a new clustering algorithm to obtain a partition in a short computation time. The first phase consists of the following steps:

1. Clustering considering timing constraints
2. Clustering considering timing and area constraints

In step 1, the path which violates the timing constraint (i.e., if the path is cut) is clustered. This means assigning all nodes in the path to the same cluster.

In step 2, clustering is performed again considering the timing and area constraints so as to obtain a better partition in reasonable computation time. This is done by clustering nodes based on a cost function in which the timing and area constraints are considered.

Phase 2 is an iterative improvement phase with an extended FM method in which a term to handle the timing constraints was introduced into the gain of the original FM. Phase 2 consists of the following three steps:

1. Initial partitioning
2. Iterative improvement with the extended FM method
3. Removal of timing violations.

In the following sub-section the conventional Ant Colony Optimization Algorithm is introduced to be followed by a description of the biologically inspired version. The biologically inspired Ant Colony heuristic can be differentiated from the conventional algorithm by the fact that it closely represents how real and biological ant live and behave to solve the food collecting “foraging” problem.

2.4 The Conventional Ant Colony Optimization Algorithm

The Ant Colony Optimization (ACO) algorithm is a meta-heuristic that has a combination of distributed computation, *autocatalysis* (positive feedback), and constructive greediness to find an optimal solution for combinatorial optimization problems. This algorithm tries to mimic the ant’s behavior in the real world. Since its introduction, the ACO algorithm has received much attention and has been incorporated in many optimization problems, namely the network routing, traveling salesman, quadratic assignment, and resource allocation problems [39].

The ACO algorithm has been inspired by the experiments run by Goss et al. [40] using a colony of real ants. They observed that real ants were able to select the shortest path between their nest and food resource, in the existence of alternate paths between the two. The search is made possible by an indirect communication known as *stigmergy* amongst

the ants. While traveling their way, ants deposit a chemical substance, called *pheromone*, on the ground. When they arrive at a decision point, they make a probabilistic choice, biased by the intensity of pheromone they smell. This behavior has an autocatalytic effect because of the very fact that an ant choosing a path will increase the probability that the corresponding path will be chosen again by other ants in the future. When they return back, the probability of choosing the same path is higher (due to the increase of pheromone). New pheromone will be released on the chosen path, which makes it more attractive for future ants. Shortly, all ants will select the shortest path.

Figure 2 shows the behavior of ants in a double bridge experiment [38]. In this case, because of the same pheromone laying mechanism, the shortest branch is most often selected. The first ants to arrive at the food source are those that took the two shortest branches. When these ants start their return trip, more pheromone is present on the short branch than the one on the long branch. This will stimulate successive ants to choose the short branch. Although a single ant is in principle capable of building a solution (i.e., of finding a path between nest and food resource), it is only the colony of ants that presents the “shortest path finding” behavior. In a sense, this behavior is an emergent property of the ant colony.

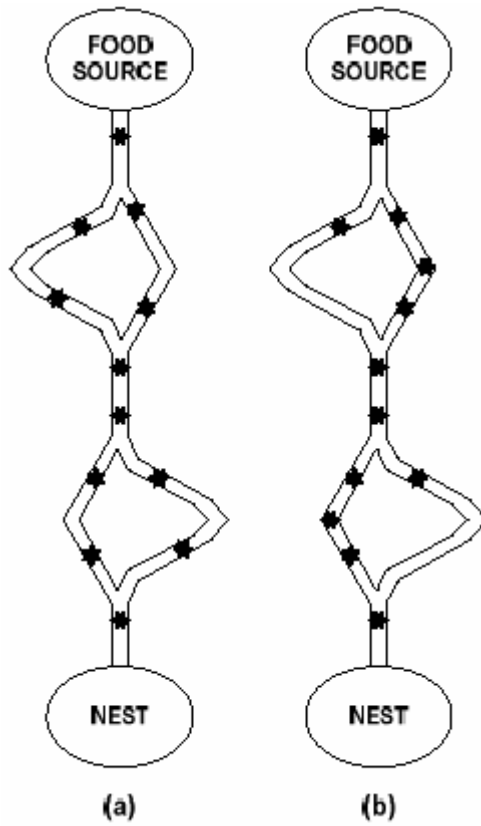


Figure 2: Double bridge experiment. (a) Ants start exploring the double bridge. (b) Eventually most of the ants choose the shortest path [38].

This behavior was formulated as Ant System (AS) by Dorigo et al. [39]. Based on the AS algorithm, the Ant Colony Optimization (ACO) algorithm was proposed [41]. In ACO algorithm, the optimization problem is formulated as a graph $G = (C; L)$, where C is the set of components of the problem, and L is the set of possible connections or transitions among the elements of C . The solution is expressed in terms of feasible paths on the graph G , with respect to a set of given constraints. The population of agents (ants) collectively solves the problem under consideration using the graph representation. Though each ant is capable of finding a (probably poor) solution, good quality solutions can emerge as a result of collective interaction amongst ants. Pheromone trails encode a long-term memory about the whole ant search process. Its value depends on the problem representation and the optimization objective.

A general outline of the ACO algorithm is presented in Figure 3 [41]. Informally, the behavior of ants in ACO algorithm can be summarized as follows. A colony of ants concurrently and asynchronously moves through adjacent states of the problem by moving through neighbor nodes of G . They move by applying a stochastic local decision policy which makes use of the information contained in the local node and ant's routing Table. By moving, ants incrementally build solutions to the optimization problem. When the solution is being built, every ant evaluates the solution and puts the information about its goodness on the pheromone trails of the connection used. This pheromone information will direct the search of future ants, until a feasible solution is found.

```
Algorithm ACO meta heuristic();  
    while (termination criterion not satisfied)  
        ant generation and activity();  
        pheromone evaporation();  
        daemon actions(); “optional”  
    end while  
end Algorithm
```

Figure 3: Ant Colony Algorithm.

The ants in ACO algorithm have the following properties [39]:

1. Each ant searches for a minimum cost feasible partial solution.
2. An ant k has a memory M^k that it can use to store information on the path it followed so far. The stored information can be used to build feasible solutions, evaluate solutions and retrace the path backward.
3. An ant k can be assigned a start state s_s^k and more than one termination conditions e^k .
4. Ants start from a start state and move to feasible neighbor states, building the solution in an incremental way. The procedure stops when at least one termination condition e^k for ant k is satisfied.

5. An ant k located in node i can move to node j chosen in a feasible neighborhood N_i^k through probabilistic decision rules. This can be formulated as follows:

An ant k in state $s_r = \langle s_{r-1}, i \rangle$ can move to any node j in its feasible neighborhood N_i^k , defined as $N_i^k = \{j \mid (j \in N_i) \wedge (\langle s_r, j \rangle \in \mathcal{C} S)\}$ $s_r \in S$, with S is a set of all states.

6. A probabilistic rule is a function of the following.

- a) The values stored in a node local data structure $A_i = [a_{ij}]$ called *ant routing table* obtained from pheromone trails and heuristic values,
- b) The ant's own memory from previous iteration, and
- c) The problem constraints.

7. When moving from node i to neighbor node j , the ant can update the pheromone trails τ_{ij} on the edge (i, j) .

8. Once it has built a solution, an ant can retrace the same path backward, update the pheromone trails and die.

It has been shown that ACO algorithm produced better quality results compared to those obtained by other heuristics when it is applied to combinatorial optimization problems such as TSP and QAP [42]. Unfortunately, only few published works found in literature that uses ACO algorithm for evolutionary logic design (Coello et al. [37]). Therefore, there is a need for investigating further the use ACO for evolutionary design of digital circuits.

In the following chapter, the problem to be solved is defined concisely and it will be followed by the description of the biologically inspired Ant Colony heuristic designed to solve the circuit bi-partitioning problem.

Chapter 3

Problem Formulation and Solution Methodology

The circuit bi-partitioning optimization is focused on finding an acceptable solution based on the delay, power, and cut-set cost [9]. The cut-set cost is the number of inter-partition connects, which if not selected carefully, will immensely degrade the overall solution quality. Fuzzy logic rules are used to balance the different optimization criteria [9].

Ant colony optimization algorithm is used for partitioning the circuit. This algorithm mimics biological ants in finding their food and marking their own territory in a real habitat [4]. In this work the biological ant's model is followed as closely as possible. Other approaches using the same heuristic rely on "smart" ants; which are "aware" of their environment and calculate distances to targets. These heuristics use small number of ants to solve their problems. In this work, many "simple" ants are used to accomplish the same task. Simple ants are oblivious to their environment and require very small memory which makes this model match closely that of the biological model. There is no central organization; simple rules applied by all ants will collectively form the new solution.

The principal idea this approach is based on is self-organization in a “super-organism.” Societies of social insects composed of thousands of individuals, which have “cognitive abilities” that by far transcend the abilities of each of the individual members. This happens, as if the society is ruled by the invisible hand of a central organizer [1]. Each individual in the community works according to simple instinctive rules. They are totally unaware of the direction their entire society is heading. In fact, there is no direct connection between the individual behavior and the society direction as a whole. The integration of all individuals in the society produces an overall flow that has a definite purpose and direction. By carefully devising the individual instinctive rules, we can steer the entire society into exploring the combinatorial optimization solution space and finding our acceptable solution.

Facilitating the simple individual mission requires carefully adapting the problem to the solution concept. The circuit is laid out as the ant habitat. The habitat is presented to ants in such a way that matches the ant’s instinctive behavior. The circuit to be partitioned is mapped into a grid. The grid cells contain the circuit components and are considered the “food” for ants to seek and store at their nest (the pertinent partition.) The following summarizes the ant’s instinctive behavioral rules.

- Each ant look for food “forage”, once found, it heads back to colony “storage.”

- Ants prefer the paths with high pheromone level.
- Trail information is communicated among ants by reading the pheromone value.
- Although unaware of the distances, ants deposit more pheromones on shorter paths and would prefer them.

Every ant follows its instinct as explained above. Different ants form trails in their mission of exploring the habitat. The trails are identified by high concentration of pheromone values on the habitat cells. Ants in-turn sense the pheromone and react in relation to its concentration.

Circuit components are distributed in clusters over the entire habitat. The clusters are formed by selecting a seed cell and position the cells connected to it nearby. A given gate would join a certain cluster if it is connected to it. The larger the cluster the more important its relative position to the colonies would become. If a cluster is positioned in the middle between colonies, ants of different colonies would pick its gates up, and thus assign them to different partitions. In this case the number of nets cut would increase.

Gates forming a long delay path should be positioned closer to a colony to facilitate their pickup by a single colony and thus decreasing the unnecessary delay introduced by partitioning.

Gates with high power dissipation should also be positioned carefully. High power gates of a critical input output path should be part of the same cluster otherwise they most probably end up in the different partitions. Partitioning will add up to the power loss due to long communication lines. However, if the power dissipation of the circuit is to be distributed over the partitions (to eliminate areas of intense heat which may render the circuit partitioning solution not economical to package) the high power gates should be positioned in different partitions to facilitate their assignment to different partitions.

The gates clustering depending on the conditions above and the ants' trails form the basic dynamics of the system and emphasize its self organization. Pheromone trails are dynamically created by the ants to signal habitat area of gates abundance.

Soon after the ants signal themselves for areas rich in gates, the area will be depleted and the pheromone evaporation will steer the ants away. The evaporation function is chosen such that it extremely penalizes high pheromone levels.

When an ant finds a gate it calculates its weight according to a simple criterion. The weighing function is in relation to the individual ant's colony's assigned partition. Ants of different colonies will weigh the same gate differently, and will base their decision of carrying the gate accordingly.

A very attractive gate of light weight would be picked up by an ant from a specific colony. The same gate would be weighed too heavy for another ant from a different colony.

The weighing function was assumed to be the cutset in this thesis. Gates connecting to an equal potential wires form a net. Assigning any of the gates of the net to different partition will introduced an undesirable cut, the number of which should be minimized.

In addition to cutset, the other cost measures the solution is evaluated against are the power dissipation and delay.

As mentioned earlier in this section the power is affected by the partitioning operation. Assigning gates to different partitions will require the electrical signal to travel across additional distances that were newly introduced. These signals travel from an input towards the outputs and traverse gate paths. If the signal feeding a gate came from another partition, the power is multiplied by a factor indicating the undesirable introduction of power dissipation.

Power is calculated as follows:

Circuit power dissipation = (power factor)(circuit switching factor)(net cut factor).

The power factor was calculated as 12.5×10^{-15} as in previous work[9].

Circuit switching factor was calculated as 1×10^{16} as in previous work[9].

Net cut factor calculated as 100 as in previous work[9]

Delay is also affected by partitioning. Signals traveling to the other partition will suffer from additional delay due to lengthy wires. Delay traditionally is calculated for specific input output paths constituting the circuit long delay paths. These paths should be dealt with extra cautiously in order not to impact the delay measure immensely.

The basis of delay calculations is the capacitance and the resistance of the circuit components and the connecting wires. The inherent gate delay depends on the gate type and the technology used in its fabrication. The gate delay is obtained from the manufacturing specifications. The resistance of the wire path and its capacitance are added in a special way to account for the gate load factor.

$$\text{Delay} = \text{inherent gate delay} + [(\text{gate load factor "Ohms"}) (\text{sum of the capacitance} + \text{net factor})]$$

The net factor is the parameter used to penalize cutting a long delay path and was calculated as 100.

The above cost calculations are taken directly from previous work [9] and used in this thesis to establish common grounds for heuristic comparisons.

Chapter 4

The Ant Colony Heuristic

The following are important definitions:

- Habitat: the place where ants live. Implemented as a grid of 2-D doubly linked list.
- Ants: artificial intelligence agents. Implemented as a simple data structure.
- Colony: a part of the habitat where member ants deposit collected items.
- Nest locus: the colony location on the grid.

The circuit graph is mapped onto a grid such that the number of vertices in each cell is limited by a parameter (habitat cell bag capacity.) The grid is a two dimensional doubly linked list which makes the ants totally oblivious to their location and orientation and, thus, react only to their instinctive behavior. Ants are generated and start foraging “collecting circuit components” from their nest locus and the number of colonies will define the number of desired circuit partitions. The number of ants is selected in relation to the circuit size.

The Ant Colony Optimization heuristic is based on three components habitat, colony, and ants. The careful design of these components makes the ant society navigate into the solution space in a smart way.

4.1 Habitat

The habitat is designed as a two dimensional doubly linked list of cells forming a grid. Each grid cell has a link to the north, south, east, and west neighbors. Figure 7 below shows a representation of a grid cell.

Each grid cell contains a “Bag” of nodes. The Bag is a data structure consists of a linked list and a set of controlling methods. A pointer to the bag is maintained in the cell structure. The cell bag has a maximum capacity set to an integer value at the iteration beginning. In addition, the cell contains a floating point variable for the pheromone value, a Boolean variable to signify an ant needing help (to be explained later in the ant sub-section).

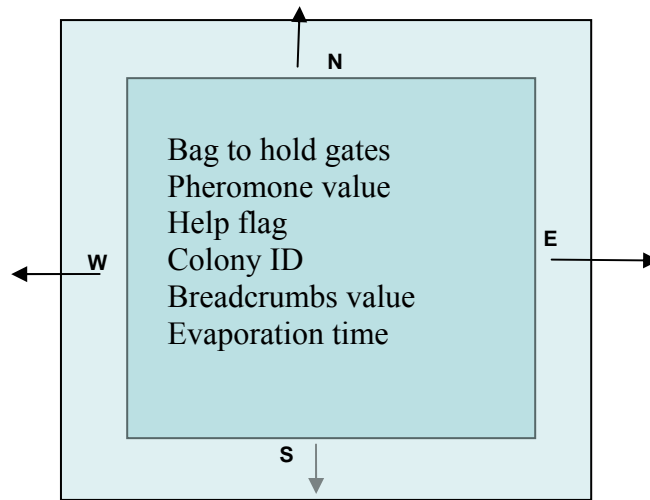


Figure 4: A habitat grid cell structure and methods.

In addition, the grid cell contains colony ID variable to signify that the cell has become part of a colony bag of cells. “Breadcrumbs” is a value used to help ants get out of loops. Ants which may be traveling in loops can leave a trace on the habitat cells to help them in breaking off of the loops (this will be explained later in the ant sub-section.) Furthermore, the cells contain another variable, the evaporation time, which holds the time of latest evaporation.

Furthermore, the grid cell structure contains methods to restore the pheromone value to the default one if the pheromone was faded away by ants passing by that cell and not finding any nearby food “gates” (as will be explained in the ant sub-section.) It is worth mentioning that pheromone, help, and colony ID have setters and getters methods, so they can be accessed by ants and colony entities.

4.2 The colony

The number of colonies is the same as the number of partitions the solution requires. The colony is basically a data structure containing a pointer to a Bag of habitat cells. In addition, it contains methods to calculate the number of gates stored and others to check against the dictated partition balance criterion. Of course, colonies contain ants. Figure 8 depicts the colony.

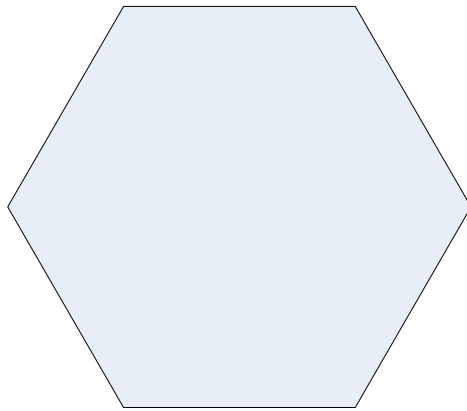


Figure 5: The ant's colony.

When a grid cell is added to the colony, the new cell is not extracted from habitat; rather a copy of the pointer to the grid cell structure is added to the colony bag. This is quite useful for the ants since they can enter their colony (or other colonies) the same way they do with other habitat cells. At the end of each iteration, the colony stores all the gates pertinent to the partition the colony is representing.

The colony inserts the gates into the habitat cell's gates bag. Ants hand in the gate to the colony and the colony checks its cells bag for a cell with a gates bag vacancy. The ant triggers the insert function, and the colony performs it. If the colony runs out of vacant habitat cells, it prompts the ants to look for a suitable cell nearby. Then the colony adds the habitat cell to its cells bag and labels the cell as a colony cell.

The colony keeps track of the number of gates deposited since the balance criterion controls the acceptance of any solution. The bag structure has an inherent property of keeping the number of stored gates handy at all times.

The colony has naturally very high concentration of pheromone that is never evaporated. This will help the ants to sense the colony from far away cells. In addition, ants from other colonies will sense the high pheromone too. Ants from other colonies will be attracted and lured to raid a colony that is not their own. In the raiding mode ants pick up

heavily connected gates and assign them to a different partition, unintentionally helping to improve the quality of the solution. In addition, raiding is a good technique to get out of solution local minima.

The colony is a dynamic structure that grows and shrinks during run time accommodating the number of gates picked up at that instance. When too many gates are picked up, more habitat cells are added to the colony. Conversely, when the colony is raided by other colony ants, the colony reduces the number of habitat cells marked as colony cells.

The colony generates the ants and it triggers the beginning of their lifecycle. Ant treads are first generated in suspended mode, and then they are all activated to provide better parallelism.

4.3 Ant

The ant agent was carefully designed to be both simple and flexible. Each ant structure contains a bag of predetermined capacity for gates, three sets of simple arrays containing vital information on the immediate environment. Figure 9 below depicts the ant structure.

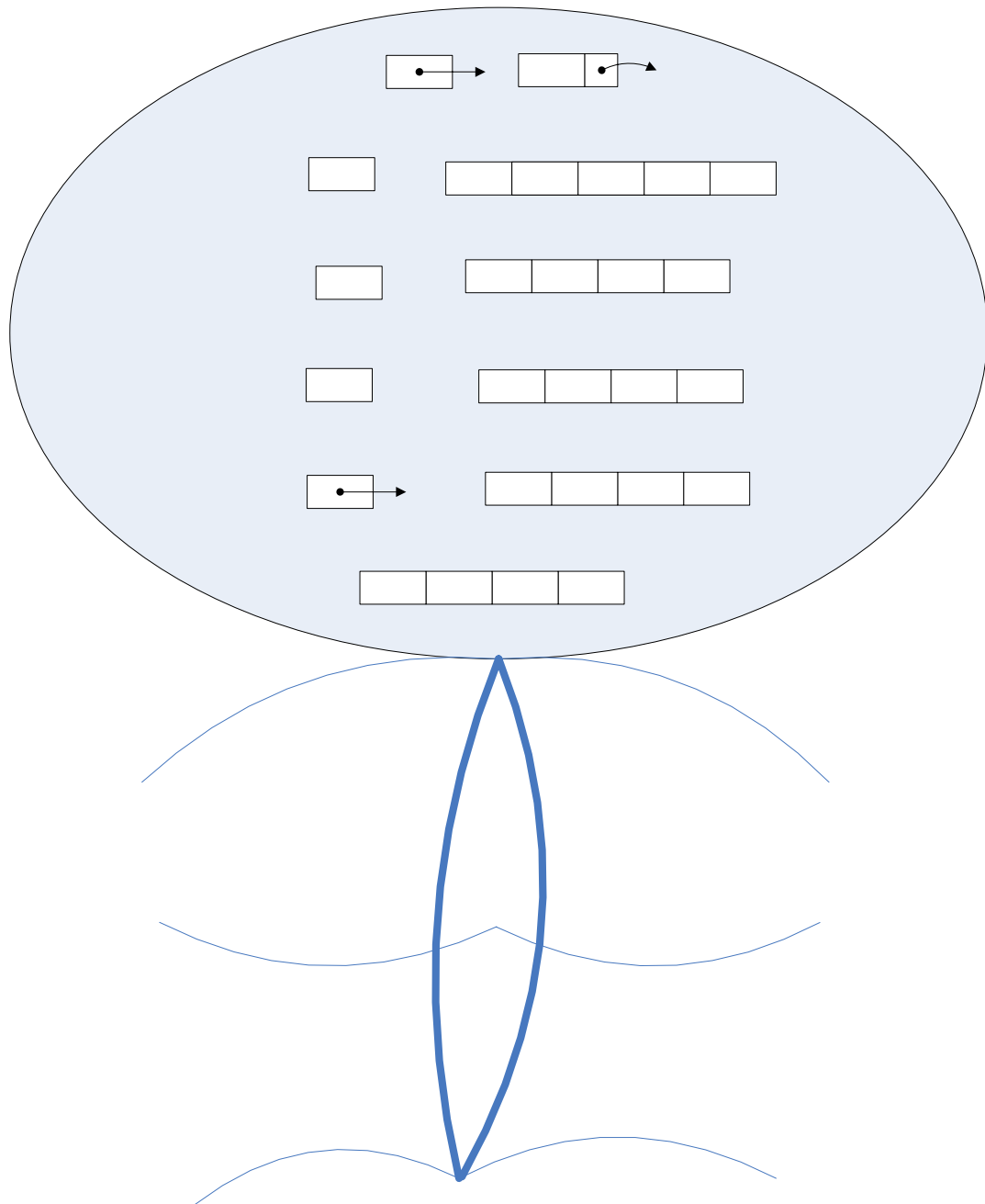


Figure 6: the ant's data structure

4.3.1 Scout data structure

Scout is an array containing a pointer to immediately surrounding grid cells and the number of gates contained in that cell's Bag. This information is saved in each element for the cell on the left, ahead, and right of the ant.

4.3.2 Vision data structure

Similarly, the vision array keeps the calculated sum of the pheromone and help value. Both of these values are calculated for the n cells ahead of each cell in the direction pointed to by each of the Scout matrix elements. The number of cells ahead for each direction is another user set parameter to be set for optimum performance. Furthermore, the scout direction symbol stores a representation of the four compass directions (N, S, E, and W) the order of these symbols is compared with the array index. The array index is as follows (Left, Front, Right, and Back.) As the ants turn the scout direction symbol array is transformed to reflect the new orientation. This process is explained in more details in the re-orient ant paragraph.

4.3.3 Track data structure

Finally, Track keeps record of the grid cell pointers as well as assisting the calculation of the ant coordination north, south, east, and west pointers to the surrounding cells in the grid habitat. Each ant keeps a track record of the current cell and the one immediately before "here and previous." When the ant turns, simple re-ordering of the coordination

pointers will guarantee the directions ahead, left, and right are reoriented with respect to the absolute ones in the grid cell habitat.

The vision arrays hold the pheromone values and the help values for the gates of the four directions. The number of gates examined in each direction is controlled by a user set parameter.

4.3.4 Steps Counter variable

Ant Steps variable keeps track of the number of steps the ant makes in each round trip. If the ant exceeds the number of steps it made in the foraging during its storing period, a certain mechanism will be triggered to help the ant break the loops in its path. This mechanism will be explained in detail in the “Update habitat cell breadcrumbs” function description.

4.3.5 Minimum Carried Weight

Minimum carried weight keeps track of the minimum weight carried by the ant. The ant may carry more than one gate in its complete cycle of forage and storage. The condition is that the gate will be picked up only if its weight is lower than that of the minimum gate weight carried by the ant in the current trip.

4.3.6 Navigate data structure

The navigation information helps the ant to take the decision of the next move to another habitat cell. This is done by calculating mainly the pheromone and the number of gates in each direction. Other factors like help and site colony are also included.

Help is a flag set by the ant on the habitat cells to signify the presence of a gate having many connections that is not attractive to be assigned to the partition the ant subscribes to. Site colony is another factor used by ants when they are lost. The ant out of desperation will look for its colony while having the radius of its sensation progressively increasing. One experiment shows this was rarely needed.

The ant explores the habitat freely using its internal arrays and variables to keep track of its immediate steps only. The ant is supposed to be on the move always in a continuous cycle of foraging and storing. Aided by its internal variables, the ant recalculates its orientation when it turns around. The ant takes a decision on its next move, and then moves to the navigated position on the habitat.

The ant does not respond directly to the habitat cell topology, rather it responds to the clues in it. This fact made the ant although oblivious to its environment, explores the habitat freely. By carefully hiding the absolute habitat compass directions and the cells

layout, the ant is forced to calculate the necessary orientation and navigation information based on the clues we allow.

Habitat clues include the pheromone and number of gates within a given habitat cell. Each ant has its own interpretation of its surroundings due to the localized calculation nature. The ant's interpretation of the habitat clues is reflected in the pheromone trace it leaves on the cells it explores. Therefore, the pheromone acts as the medium to store the ant's history based on its experience cultivated for a specific region of the habitat.

All ants read the pheromone information and base their decisions accordingly. The ant's decisions are influenced by the history encoded into the pheromone. Hence, the coordination of the ants is established through the depositing and sensing of the pheromone. Ants build up their indirect awareness through the pheromone.

The synergy of the ant's interaction yields pheromone traces starting from the colonies and ending at areas of high gate concentration. As the gates deplete from a region the pheromone trail will evaporate and lose its definition. A new path could emerge nearby as a result of re-exploration, but the old path traversing the same habitat cells may never be reestablished.

Dynamic path generation depending on the gates concentration facilitates the pick up and storage of gates. The gates are clustered according to their connectivity, delay, and power, and hence the gates picked up will most probably be of low cost when assigned to the same colony.

The explanation provided above serves as the reasoning and the motivation behind the success of this heuristic.

4.4 Ant's lifecycle

Ants go in a continuous cycle of forage and storage, until all cells are collected where new program iteration starts. Figure 10 and Figure 11 are flow chart diagrams of these parts of the ant's life.

4.4.1 Foraging

The ant's lifecycle starts with foraging. This is the phase where the ant starts to collect the gates and ends when a gate is picked up from a habitat cell.

The ant starts by updating the pheromone value of the habitat cell. The incremental value is none linear as discussed in the "Update habitat cell pheromone" in the forage cycle.

Next, the ant updates its own scout matrix consisting of the number of gates and the direction symbol. The scout operation is focused primarily on the surrounding cells whose bag cells are examined. In addition, the scout matrix saves the current orientation reference information, the scout cell direction symbol.

The ant updates its vision matrix. In this step, the ant examines the pheromone and the help flag of the surrounding habitat cells. The number of cells examined in this step is controlled by a user defined parameter.

The ant then navigates, i.e. decides on the next habitat cell to move to. The navigation primarily selects the direction with the highest weighted combined value consisting of the number gates, pheromone, and help. Each of these components will have its own importance weights which are another group of user defined parameters. The navigation produces a pointer to the next habitat cell to move to.

Next, the ant moves to the navigated direction. In this step, the ant update the internal Track matrix starting by the “here” element of the array which saves a pointer to the center cell where the ant suppose to be located.

The ant increments its steps counter. This will be used to help the ant in guessing it is traveling in loops.

The above steps are repeated until a gate is found in the bag of the habitat cell pointed to by the “here” element of the “Track” data structure of the ant.

Once a habitat cell having gates is found, most frequently the cells bag will contain more than one gate. In this case the ant will evaluate the weight based on the cutset criterion, and a decision is made. If the gate is too costly to be assigned to the partition represented by the colony, the gate is not picked up; the above foraging steps are repeated. But, if the gate is heavier than a specific user defined parameter value, the gate is not picked up, and a help flag is set on that habitat cell and the ant goes through the same foraging steps above. Finally, if the gate’s weight is lower than that of the user parameter, the ant will attempt to pick the gate up.

Ants pick up gates with a nonlinear function producing a probability inversely proportional to the gate’s weight. If the gate is not picked, the ant repeats the same foraging loop above. If the gate is picked, the ant will carry it.

The ant has a sack “bag” which is used to carry the gates. The ant first extracts the gate from the habitat cell bag, and then it inserts it into its bag. When this is done, the ant is the only keeper of that node, and nothing else has access to it until it is released.

The ant keeps track of the minimum weight of all the gates carried, and only carry additional gates that are less in cutset weight than the ones it is currently carrying in the current cycle.

The ant prepares itself for the storing part of the cycle by starting to decrement its trail steps counter. By simple calculation, the ant should make it back to its nest locus in the same or less number of steps.

The foraging part of the cycle is concluded by the ant flipping its direction via performing a special transformation on the “Scout” and “Track” matrices.

4.4.2 Storing

The storing part of the ant’s lifecycle is explained briefly and the details of the functions will follow.

First the ant checks if it is in another colony, if this is the case, it gets out of the colony by first trying to go straight through it. If this effort failed (that could be because the other

colony ended on the edge of the habitat and the ant has no easy way to return), the ant will travel across the other colony in a spiral out fashion. If the ant's gate bag has not reached its capacity, the ant raids the other colony. Raiding meaning the ant will pick up a gate from another colony if it is weighed light when assigned to that ant's colony partition.

Then the ant modifies the cell pheromone value by a special nonlinear function used by the storage process. The same pheromone value of the cell is modified by both the forage and the storage processes; each with its function. Ants investigating the habitat cell afterwards will see the combined effect of both. The lack of distinction of these types of pheromones adds to the chaos of the algorithm which is believed to play an integral part in its ability to climb up of local minima.

The ant proceeds by updating the scout matrix and the vision matrix, and then it calculates the navigation to get the address of the next cell to move to. These steps were mentioned in the forage section above.

At this point, the ant needs to check if it already traveled a longer path in storing phase than it did for the foraging one of the same food collecting cycle. If the number of the extra steps the ant has taken is larger than a user defined parameter, the ant uses breadcrumbs.

Breadcrumbs method is an idea inspired from an old fairy tale where a kidnapped child left a trail of breadcrumbs to guide his rescuers to his place. Off course, the birds ate the breadcrumbs and the child used something else! In this heuristic, all ants store their steps counter number in a variable in each habitat cell. They only do that out of desperation when they are lost. As in the fairy tail, the heuristic adapts a very rough mechanism to guide the ants out of the loops. A more robust method would have been to label the habitat cells by the pointer address of the ant as an ant ID. Throughout the heuristic care was taken not to introduce intelligent ant decisions and to capitalize on the chaos.

The ant moves in the navigated direction. In its way, the ant checks its gate bag if it can carry more gates. The ant weighs the gate's potential cutset if it were to be assigned to the partition signified by its colony. The gate of selection is considered for pick up only if its weight is lighter than the lightest gate carried by the ant during the current forage and storage round trip. If the gate is picked up, it will be inserted into the ant's gate bag "carry" and the ant's minimum carried gate weight will be updated. If the gate is not picked, or its weight is not attractive at this moment, the ant bypasses the unnecessary steps and proceeds to the next step.

The ant decrements its steps counter to safeguard it against being lost and re-orientes by transforming the direction matrices of the "Track" and "Scout". The ant checks if it

entered another colony. In that case, it tries to get out as explained earlier and also it raids the other colony if its bag has vacancies.

This part of storage is repeated until the ant reaches its colony. In which case, the ant resets its steps counter and proceeds to communicate with its colony as follows.

If the colony was found full and can not take more gates, for if it did the balance criterion will be jeopardized, the ant will flip its direction (turning backwards) and re-orient its internal directions. The ant puts the gate down in the nearest non-colony vacant cell.

If the colony is not full, the ant prompts its colony to hand in a pointer to a vacant habitat colony cell. If there is none, the ant is requested back to look on the colonies behalf for a non-colony vacant habitat cell, which the colony would add to its bag of cells.

Next, the ant stores the gate in the colony cell which the colony had handed its pointer to it.

Finally, the ant flips its direction and turns backwards in preparation for another foraging cycle.

This concludes the forage/storage life cycle which continues until one of the ants pick up the last gate positioned outside the colonies. When that happens, a new iteration is started after changing the relative position of the colonies.

Many functions are shared between the forage and storage phases of the ant's lifecycle. Sometimes the function is slightly different just to eliminate unnecessary calculations, yet it maintains the same logic. The ants' actions and lifecycles are not synchronized at all and the only information sharing is through the pheromone deposited on the habitat cells.

The storage part is considerably more complex than that of the foraging because it is more difficult to find the ant's colony with this limited amount of information. The ability to collect gates when the ant heads home improved the quality of the solution since the lengthy part of the trip is being used to pick up more suitable gates having less number of nets cut.

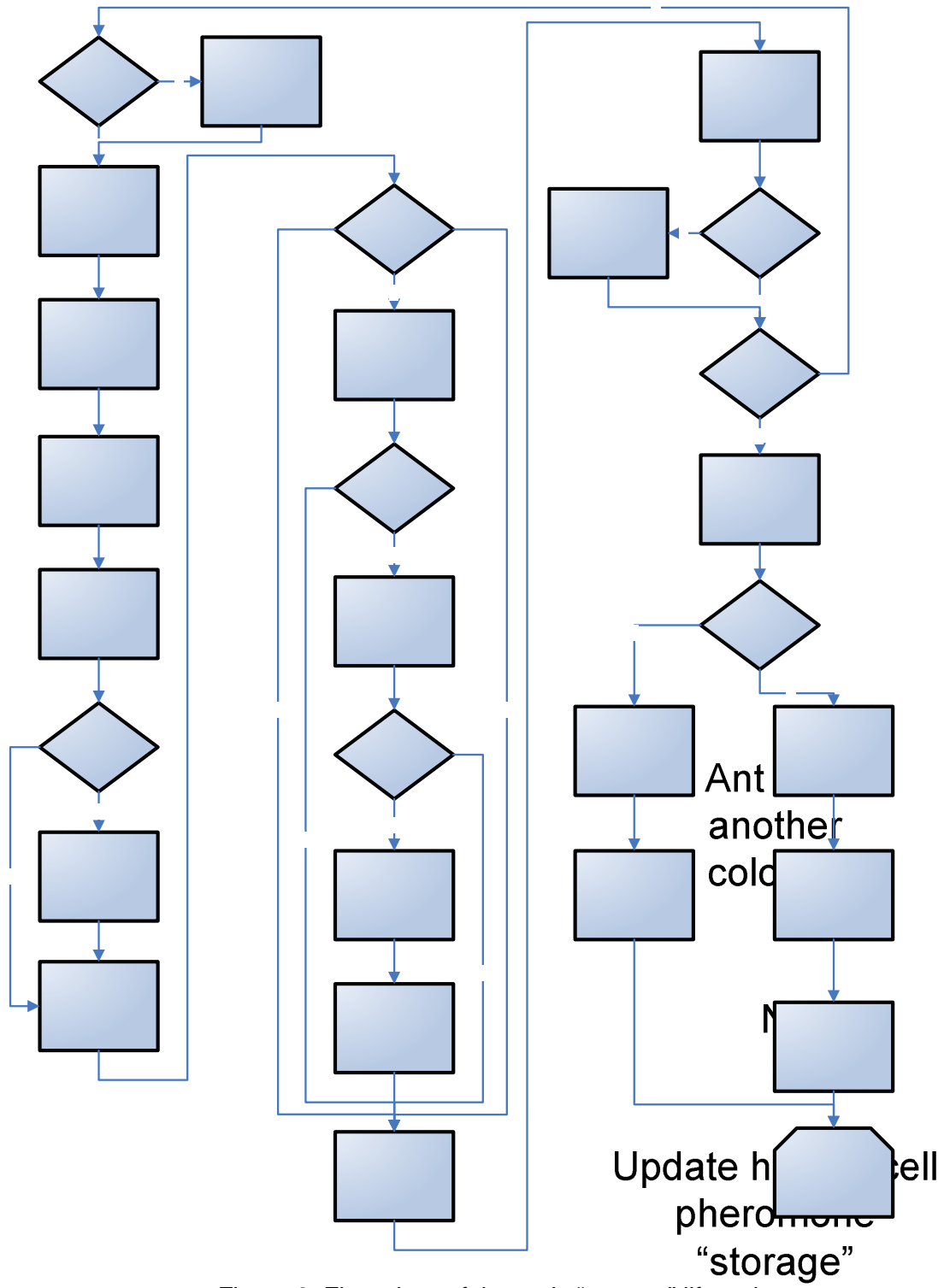


Figure 8: Flow chart of the ant's "storage" lifecycle.

The following is a description of the functions used by the ant. Figure 10 and Figure 11 provide the way these functions are linked and how finally the ant community would be organized to solve the partitioning problem.

4.4.3 Update habitat-cell pheromone “forage”

The ant deposits more pheromone on habitat cells having more gates. The pheromone is calculated by the following formula:

$$\begin{aligned}
 \text{pheromone} = & \text{previous_pheromone_value} + \\
 & (\text{foragePheromoneYScale})((\text{pheromone})(\text{foragePheromoneXScale}))^{\text{KForage}} * \\
 & e^{-(\text{pheromone})(\text{foragePheromoneXScale})} \quad (2)
 \end{aligned}$$

Where forage Pheromone X-Scale, forage Pheromone Y-Scale, and K-Forage are user defined parameters to be set at the beginning of the program.

During the ant’s foraging period, the habitat cell’s pheromone is increased by the above nonlinear function. The function is plotted in Figure 12 below by setting the parameters as follows:

Forage Pheromone X-Scale = .4.0

Forage Pheromone Y-Scale = 1.5

K-Forage = 3.0

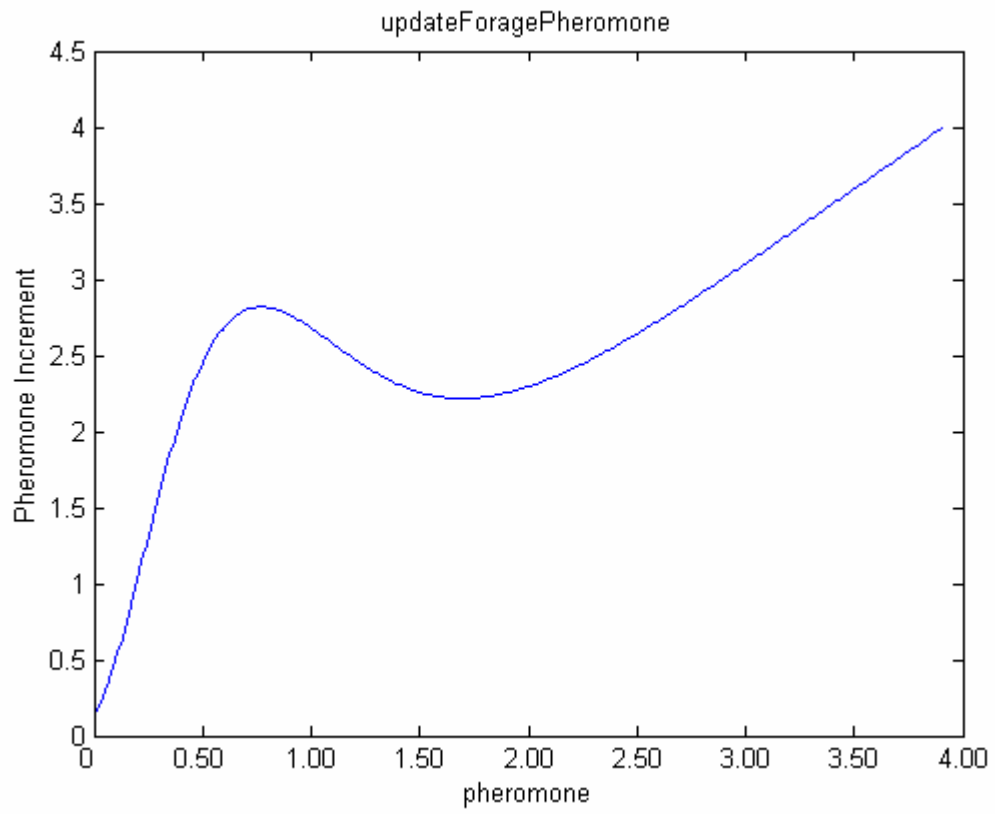


Figure 9: Nonlinear pheromone increase during foraging.

The pheromone increases rapidly in the first section where the habitat cell is visited for the first times, and then it gradually levels off. As the number of visits increases again, the pheromone level climbs up uniformly. This behavior was selected to increase the pheromone of habitat cells visited few times to encourage the surrounding territory exploration. Once too many ants are in the area, no need for pheromone boost, and the increase levels off. The actual shape of the graph is controlled by the parameters mentioned above and the curve shape in Figure 12 was used for small circuits as stated earlier.

4.4.4 Update ant's scout information

In each step, ants scout three directions: left, forward, and right. Scouting is done by examining the number of gates in each of the three directions. Ants examine a parameterized number of cells "ScoutLimit" in each direction. ScoutLimit represents a percentage of the habitat cells that the ant examines in a particular direction. The description of this function is as follows:

The ant tries to match the pointer values on the habitat cells to those stored internally in the track array. When a match is found the ant will identify the compass direction (N, S, E, and W) signified by the scout cell direction symbol. This direction comparison is important since the ant is totally oblivious to the habitat as a whole. The ant does not

directly know the compass directions of the habitat and this issue is encountered each time the ant turns.

Furthermore, the ant adds the number of gates sensed in the ScoutLimit habitat cells ahead in each direction. The process of scouting is repeated for each of the directions.

4.4.5 Update ant's vision information

Vision information is basically the pheromone sensing capability of the ant. As in the scout update function, the ant needs to match the scout cell direction symbols to the compass directions embedded in the habitat.

Similar to the scout function, the pheromone values of the habitat cells are measured and added for each of the three directions. The number of habitat cells examined is controlled by the VisionLimit program parameter to be set by the user or another heuristic.

For performance purposes, the vision operation triggers the habitat cell pheromone evaporation function.

4.4.6 Pheromone evaporation

Each habitat cell has its own pheromone value and time of last update. If the time of last update is greater than a user set parameter “Evaporation-Threshold” the cell’s pheromone will be subjected to evaporation. The evaporation formula is nonlinear and is as follows:

$$\text{pheromone} = (\text{previous_pheromone_value}) * e^{-(\text{previous_pheromone_value})(\text{pheromone_age})(\text{evaporation_rate})}$$

(3)

The pheromone age is the time difference of current time and the time of last update on the habitat cell. Where as the evaporation rate is a user set parameter.

The evaporation function for an instant of pheromone age is plotted in Figure 13 below, the other parameter values are as follows:

Evaporation Rate = 0.00015

Evaporation Threshold = 1000

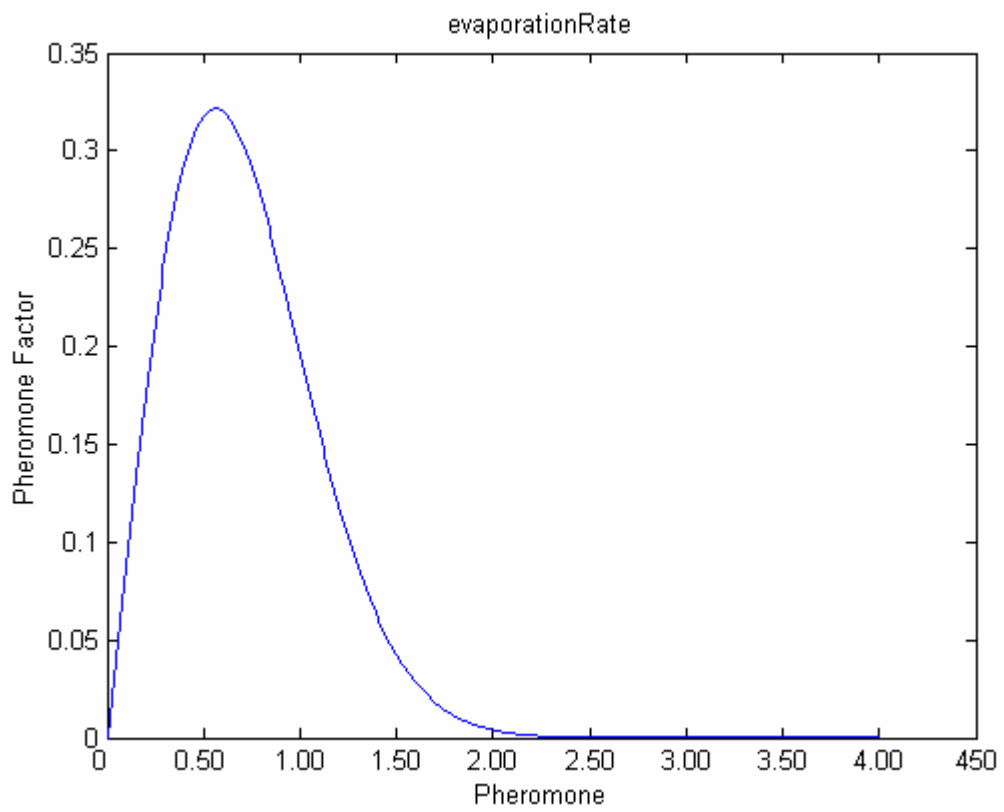


Figure 10: Nonlinear pheromone evaporation.

After exceeding the threshold time value, a sharp decrease of pheromone is encountered at lower and higher values. The evaporation function favors medium lower range of pheromone values, high values are chopped off. Similarly, this function deals with pheromone age, once exceeded the time threshold, the lower medium aged pheromone values are diminished and the very old ones are chopped off.

The motivation behind the evaporation curve shape was to use a band pass filter concept. Values of low pheromones indicate non-interesting areas of the habitat, and they should be evaporated rapidly. Values very high in pheromone indicate over exploitation and the gates may have been depleted from that region and hence, pheromone is evaporated drastically or even chopped off altogether. Only moderate pheromone values are reduced moderately. The shape of the graph in Figure 13 is the actual curve used for small circuits, the preset parameters mentioned above tune the curve shape to suit the circuit needs depending on its number of gates.

The ideal case is to implement the pheromone evaporation as a continuous process over all habitat cells. For performance purposes, the cell's pheromone is not evaporated until the Evaporation Threshold has ended, and then the evaporation happens to the cells as they are examined by the vision function of each ant.

The least visited and the most visited are evaporated heavily. Normal evaporation happens only to the moderately visited and having moderate pheromone value as well.

4.4.7 Decide on next habitat cell to move to “navigate”

The navigation function selects the habitat cell that the ant will move to. The ant can through navigation move to left, front, or right. In this function, the factors determining the direction are balanced out and the direction that is attractive the most is chosen. The factors that contribute to the direction selection are the number of gates in that direction, the pheromone values, how far the ant had traveled from its nest (this feature is used only in desperation state).

For each of the three directions (left, front, and right) a potential factor of the direction is calculated. The factor is calculated as follows:

$$\text{Direction Factor} = (\text{help weight})(\text{help aggregate of the direction}) + \\ (\text{pheromone weight})(\text{pheromone aggregate of the direction}) + \\ (\text{number of gates weight})(\text{number of gates aggregate of the direction})$$

Where: help weight, pheromone weight, and number of gates weight are preset parameters.

The aggregates are the sum of help, pheromone, and number of gates in a specific direction. The number of habitat cells whose values are to be summed is controlled by a user specified parameter (Scout Limit and Vision Limit.)

The direction with the highest direction factor is normally selected, unless the ant suspects it is running in loops. In which case, the ant adds another component to the above direction factor calculation. The additional component is whether the ant can sense a colony near by. This colony sensation is used to break the loops because the ant will seek the nearest colony and will change its pat calculation just enough to break out of loops.

Navigating based on the pheromone and the number of gates in the habitat cells can sometimes lead to infinite loops. The ant will travel through the same set of habitat cells indefinitely. To break this undesirable behavior, if the ant had traveled a considerable number of steps higher than the habitat cell number, it automatically navigates to a random habitat cell near by rather than the calculated cell. This feature helps the ants get out of trail loops during foraging.

4.4.8 Move ant

The ant's track array is updated with the navigated habitat cell pointer. The track array contains pointer reference to the center cell and the other four cells at its sides. First, the center cell reference pointer is copied to the respective track array cell, then the pointers

of the side cells references are copied similarly in their respective ant's track positions. The position of the track array is calculated using the scout cell direction symbol array. By comparing the pointer on the cell habitat to that of the original ant's track, the compass position of the ant is known.

4.4.9 Increment ant's steps counter

The ant counts the steps it moves starting from the moment it deposits the last gate at its colony, or when just after it dropped its load nearby the colony (in case of the colony had exceeded its storage capacity.) In case of a newly generated ant, the counter is incremented after the first move away from the colony center.

The steps counter helps the ant to "guess" it is going in loops. It is important to note that this method of "guessing" was meant not to be robust in order to increase the non-deterministic behavior of the heuristic.

4.4.10 Re-Orient ant

The habitat cells are interconnected via pointers in the north, south, east, and west directions. Since the habitat is represented in a doubly linked list, it is impossible for the ants to know where they are without additive information. The problem is the same with the direction. When the ant turns the direction matrices should be updated to sustain valid

orientation information. This is done by transforming to the Track and Scout orientation matrices.

When turning left the following transformations occur:

- The left columns get information of the back
- The back columns get information of the right
- The right columns get information of the front, and
- The front columns get information of the left.

And when turning right the following transformations occur:

- The front columns get information of the right
- The right columns get information of the back
- The back columns get information of the left, and
- The left columns get information of the front.

4.4.11 Select lightest weight gate

When the ant encounters a habitat cell containing more than one gate, the ant weighs the gates and selects the one with the minimum weight. This gate is the one produces the smallest number of nets cut when assigning the gate to the ant's colony partition.

The gate at hand is connected to many nets as input and output to the gate. If the other gates connected to these nets are assigned to different partitions, the net cutset will increase, which in turn increases the weight.

Since real world circuits have many gates interconnected and feedback loops, it is impossible to have zero cutset (unless the whole circuit lies in a single partition), and we look only for solutions that satisfy a predefined acceptance criteria.

4.4.12 Set habitat cell help flag

If the ant encountered a gate that is heavy enough making it undesirable to carry, it marks the habitat cell having that gate with the help flag and continues on foraging. Other ants from another colony may sense the help signal and respond to it. The gate that looks heavy for an ant from a colony will most probably look light for another ant from a different colony because each colony stores gates of a specific circuit partition. If a given gate is assigned to a partition and that assignment yielded many nets cut, most probably assigning the gate to a different partition will yield better result.

4.4.13 Pick the gate up

The gate is picked up by a probability function. It is either picked up by the ant or left in place according to the following function.

$$\text{Probability_of_Pickup} = \left(\frac{\text{Pick_up_Parameter}}{\text{Pick_up_Parameter} + \text{gate_Weight}} \right)^2 \quad (4)$$

The above probability value is compared to a random number and the gate is accepted if the random number is less than the calculated probability. If the gate is to be picked up, it would be extracted from the habitat cell bag and ready for the ant to carry. Else, foraging would continue as in Figure 10 above.

The plot of Figure 14 below shows the graph of the pickup probability for pickup parameter = 2.0.

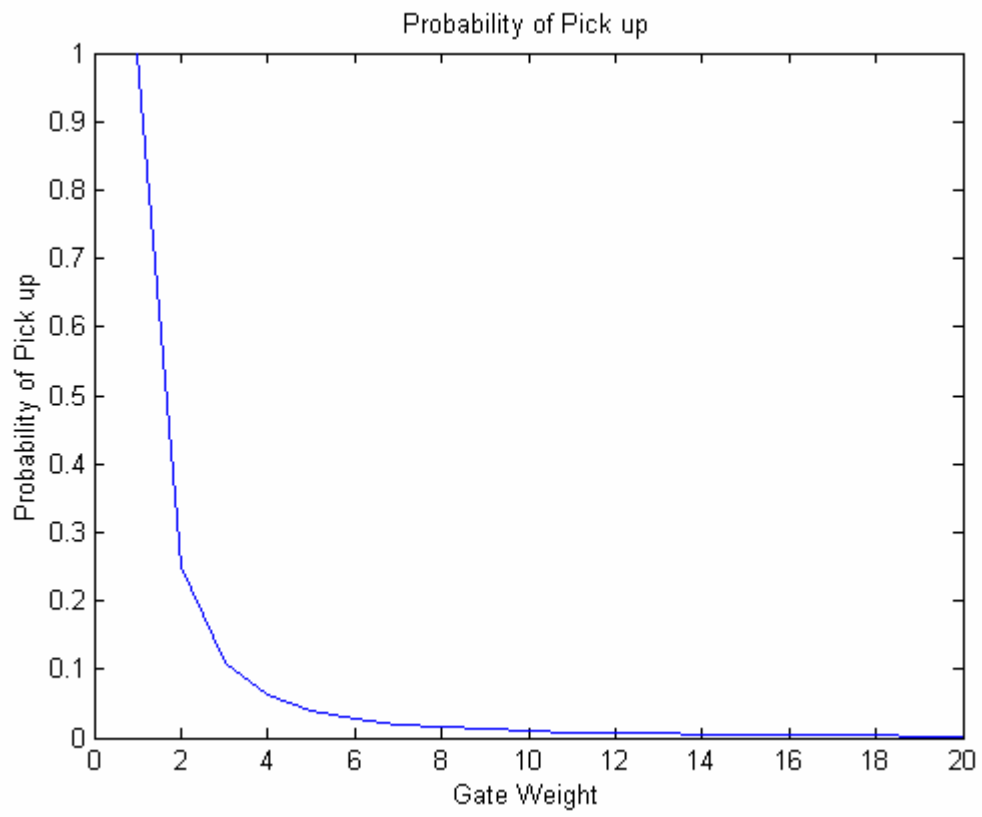


Figure 11: Pick up probability.

The curve in Figure 14 shows a probability of 1 for very light weight gates. All light weight gates will be certainly picked up. As the weight increases, the probability diminishes rapidly. Ants of other colonies may weigh the same gate differently and thus, a gate that is not picked up by an ant from a specific colony could be picked up by another very quickly.

4.4.14 Carry gate in the ant's bag

The new gate is inserted into the ant's bag. The ant's bag capacity is a predefined user parameter which has some effect on the quality of the solution. When more than one gate can be carried in a full cycle of foraging and storage, it was proven experimentally that we get better results when the foraging period is ended by picking up the first gate, and the ants try to head to their colony and pick up gates with lower weight than the ones they already carry.

Gates that are carried are either deposited into the ant's colony, or dropped nearby if the colony is full. They are not dropped in the way if the ant encountered a lighter weight gate. This helped getting out of the solution's local minima.

4.4.15 Update ant's min carried weight

Each time the ant carries a gate, it updates its own minimum gate weight for that cycle of foraging and storing.

4.4.16 Increment ant's steps counter

The ant counts the steps it makes in its foraging and storing trips. If the number of steps is larger than a ratio of the number of habitat cells, the ant uses more “guesses” to get out of a potential undesirable loop.

4.4.17 Flip ant's direction

The ant is prepared for the storing part of its cycle by flipping its direction. This is basically turning backward. To make sure the ant will behave correctly, a direction transformation is performed on the Track and Scout Cell Direction Symbol. The transformation is done as follows:

- Swap the columns representing left and right entries of the Track and Scout matrices.
- Swap the columns representing front and back entries of the Track and Scout matrices

This concludes the functions of the foraging cycle; the storing functions are similar to those of the foraging with slight modifications. The following is a list of the storing functions that was not mentioned earlier.

4.4.18 Update habitat cell pheromone “storage”

The ant deposits more pheromone on habitat cells having more gates. The pheromone is calculated by the following formula:

$$\begin{aligned} \text{pheromone} = & \text{previous_pheromone} + \\ & (\text{storage_pheromone_y_scale}) * \tanh((\text{previous_pheromone}) \\ & (\text{storage_pheromone_x_scale_factor})) \end{aligned} \quad (5)$$

Where storage pheromone x_scale, and storage pheromone y_scale are parameters to be set at the beginning of the program.

During the ant’s storage period, the habitat cell pheromone is increased by the above nonlinear function. The function is plotted in Figure 15 below by setting the parameters as follows:

Storage_Pheromone_X-Scale = 1.0

Storage_Pheromone_Y-Scale = 4.0

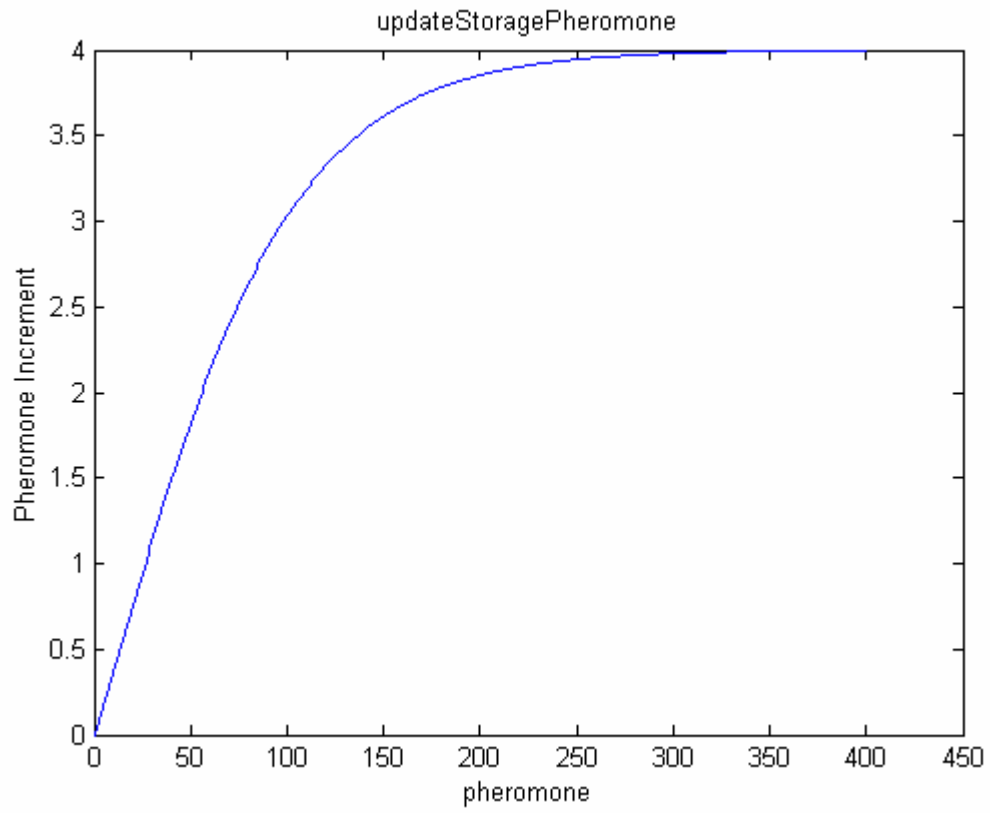


Figure 12: Storage pheromone increment.

Figure 15 shows the graph of the curve used for incrementing the pheromone during the storage cycle. The increment is very small for a smaller original pheromone value. As the original pheromone increases, the increase reaches a limit and ceases to increase further. Saturated pheromone values will soon be evaporated and cleared out after the grace period of pheromone threshold preset parameter.

The motivation of this curve shape was based on ants already finding gates in their path and they need to signal to themselves (in a future trip) and others the trail path to that habitat area. Depending on the pheromone value before the update, the ant increases the pheromone depending nonlinearly on the original value. The increase reaches saturation, and the value is increased uniformly to signal a very high important area. The high pheromone value will attract ants in nearby areas. If that did not happen timely, the pheromone will evaporate quickly, and the area will have to be re-explored.

4.4.19 Update habitat cell breadcrumbs info

When the ant starts foraging, it counts the steps it makes in its journey looking for gates. When the ant picks up a gate, it starts to decrement the steps counter. If the ant's steps counter turns negative that means the ant is lost. A user defined parameter "Lost-Ant" controls the number of steps the ant takes in storing more than those in foraging. After the Lost-Ant tolerance is exhausted, the ant labels each habitat cell it passes over by its ant steps counter value. Then if the ant reads a habitat cell value "breadcrumbs" larger

than its internal steps counter (because it is a negative value), it guesses it could be in a loop; and its navigation routine is distorted consequently.

Since all other ants are doing the same procedure this method is not exact and better described as a “guess”. It would have been more robust to store the ant’s unique thread ID in the habitat cell, but the lack of robustness in this method added to the chaotic nature of the algorithm and enhanced the local minima climbing criterion of the heuristic.

4.4.20 Move out of the colonies

As ants collect the gates, the colonies get large quickly. The colony stores the gates in the same habitat cells enabling the ants of different colonies to raid other colonies. Colony cells are only labeled differently and the ant behaves slightly different when inside one of them. One of these special behaviors is the way it travels through it. The ants traverse colony cells in spiral out fashion. This gives a chance to the ant to find a suitable gate in the other colony and picks it up. This process is called “raiding” and it improves the quality of the solution. When a relatively heavy gate is picked up and assigned to a colony, ants from different colonies may find it light (if it were to be assigned to a different partition.)

The colony has very dense pheromone concentration both inside and around. It would be very difficult to traverse the colonies and get out to other habitat cells that were not visited

before. The spiral out mechanism provides these advantages and minimizes the unnecessary looping around the colonies. The orientation transformation proved useful when spiraling because it requires frequent turns.

4.4.21 Get a colony vacant bag

The ant requests the colony to return a pointer to a habitat colony cell which has vacancies. If the colony could not find any, and the colony did not reach its gate capacity, the ant will look around the colony for a vacant habitat cell and will request the colony to add it.

4.4.22 Store the carried gates

Once the ant gets a pointer to a habitat colony cell that has vacancies, it extracts the gates from its own gate's bag and inserts them into that of the habitat colony cell's gates bag.

The circuit information is updated as follows:

- The gate is assigned to the respective partition corresponding to the colony it is deposited in.
- The gains are recalculated and the new values are reflected in the circuit data structure
- The partition information is updated and the new cutset cost calculation is prepared.

4.4.23 Put down the carried gates

If the colony is full it will not accept more gates (the colony may be raided and the number of gates will drop and consequently, the colony will accept new gates.) If the gates can not be stored in the colony, the ant will have to look around the nest for vacant and non-colony habitat cell and stores the new gates into it. The ant would extract the gates from its gate's bag, and will insert them into the habitat cell gate's bag.

The gate that was just deposited in a non-colony cell could be very attractive to be assigned to that particular partition, but the ant's effort is not totally wasted. Soon after the colony is raided, the gate would be picked up with high probability, since it is near the colony where the highest concentration of pheromone, and thus, leading the ants to it.

4.4.24 Social Ants

To further enhance the social behavior of ants, ants are created in threads that work in parallel. This dictated the locking of the resources because more than one ant could be trying to modify the information of the same resource.

Since the habitat cells values are being updated at each move the ant makes, each cell was assigned a mutual exclusive lock "Mutex." Before updating the value, the ant attempts to reserve the Mutex assigned for that particular cell. If it was not granted the reservation, it

will wait for a period named timeout. If the timeout time was exceeded, an error will be raised causing the ant to back down from that cell.

When the ant owns the Mutex, it updates the necessary information and releases the Mutex when it is done. Other ants are allowed to read the cell's values even when being locked.

Each colony will have its own Mutex. Ants deliver the gates at the colonies foot steps and the colony needs to insert these gates into the cells owned by the colony. In addition, the colony needs to update the number of gates picked and maintain the cells as well. The cell maintenance requires the colony to add a cell handed by ants to the colony; this is done by labeling the cell as a colony cell and assigning the colony ID to that of the cell. In addition the colony sets the pheromone value of the cell to a fixed value which is a user defined parameter.

All Mutexes act the same, ants wait for their turn to perform a transaction with the colonies since all transactions requires modifications. When traversing the colonies however, the Mutex is not used since the pheromone value of the colony cells is not modified by the ants.

The circuit structure is protected by a Mutex. When the ant evaluates the gates' weight it is required to set linked list pointers inside the circuit structure. Thus, other ants should wait until the Mutex is cleared.

When the gates are committed to a partition, the circuit structure is modified permanently to reflect the changes. The circuit Mutex is specially used in this case.

Colonies report their gate assignments to external files. The files access is controlled by a Mutex as well. The colonies log all gate assignments if they had reserved the Mutex otherwise wait for their turn. It was chosen that the colonies perform the logging to minimize the effort and contention resulting from many ants accessing the files at the same time.

The threading mechanism used was that of the Microsoft Class Foundation "MFC", it was not very efficient. A more efficient threading would have yielded better results.

4.5 Heuristic Related Issues

4.5.1 Controlling Greediness

During the execution of a nondeterministic heuristic, decisions on accepting values are made. Values are evaluated against a certain criterion and characterized as either good or bad. Good values are always accepted, whereas bad values are accepted by a function. The acceptance function varies the acceptance measure. Traditionally, the acceptance of bad values is high and as the heuristic progresses the acceptance is gradually reduced. This process is continued until only good values are accepted and at that stage the heuristic is said to be in greedy stage.

The Ant Colony heuristic as explained in this thesis offers an improvement over the traditional greed concept. Traditional greed factor is not varied as a function of the solution; rather it is varied as a function of execution time. In this Ant Colony heuristic however, the greed is mainly a function of the solution itself and is manifested in two different ways.

In the first greed pattern, each ant will carry additional gates only if they weigh less than the gate of the minimum weight that the ant has in its bag during the current ant round trip to its colony. Carrying only a gate that has the lowest weight is a direct implementation of the greed principal. The acceptance criteria is not a function of time, rather it is a function of the carried weight at that ant trip. This proved to be a better choice of greed than that of the traditional method.

In the second greed pattern, the colonies are progressively made to approach each other. Each time the colonies move closer, the raiding process increases significantly. Ants raid colonies that are not their own, this means ants collect gates that are costly to the entire solution. As the colonies move closer, ants will spend more time improving the solution quality by picking up gates that are already assigned to a colony. This contributes directly to the quality of the solution and is both a function of execution time and the current solution.

The integration of both of the above mentioned greed patterns contributed to the success of this heuristic. The ants are not directly aware of the solution as a whole and can not base the way they weigh the gates on the goodness of the solution as a whole, and hence, applying a traditional greediness schedule will invalidate the principle of ants being oblivious to their environment as a whole.

The heuristic runs several iterations. Each iteration is concluded when an ant picks up the last gate not belonging to a colony and deposits it into its colony. Greediness can be influenced by varying the number of gates each ant can carry and by changing the criterion limiting the weight accepted. The relative heaviness of the gate is controlled by weight being “not much heavy.” This user defined parameter can be changed by a function during run time. This will guarantee the dynamic influence on the acceptance criterion and thus the greediness of the heuristic.

4.5.2 Pheromone and Cost Optimization

Pheromone deposited on the habitat by ants form trails. Trails are sensed by ants as the difference in value ranges of pheromones on habitat cells. If a habitat cell has a pheromone value that is close to the one next to it, then they are part of the same trail. Ants will choose paths that are rich in pheromone values and those with a higher number of gates.

Experiments shown it is best to position closely connected gates in habitat cells that are close by or even in the same cell. Highly connected gates are placed in the middle between the colonies. The gates that are connected to these seed gates are positioned very close by. This forms clusters of heavily connected gates.

Ants of different colonies will generally approach clusters from different directions. Ants of the same colony will use the trail information (sensed by the pheromone) to pick up the gates and assign them to the same partition. Hence the pheromone will guide the ants to assign connected gates to the same partition and thus, minimizing the cost.

Furthermore, each ant will weigh the gate before it picks it up. The weighing function must be dependent mainly on the primary cost measure(s) the solution will be evaluated

against. The results shown in this thesis are obtained by having the cutset as the primary cost measure.

4.5.3 Implementing Different Cost Measures

Different cost functions can be implemented by directly influencing the gate clustering by that measure and by basing the weighing function on the selected measure.

In this thesis implementation the gate clusters were based on the connectivity which is directly related to the cutset. The pheromone trails acted as indications to gates heavily connected. In a sense, this implementation was an example of direct optimization based on the cutset measure. If another measure is selected the clustering should be based on the new measure.

As an example, clustering based on power would be used to direct the heuristic towards a solution optimizing the power measure. Gates would be clustered based on their power dissipation which is a function of the switching probability. Gates with high switching probability should not be of the same cluster.

Ants of the same colony will be directed by the pheromone trail to find clusters having low total power dissipation. Thus, individual low power clusters will most probably be assigned to the same partition.

The effect of the weighing function can not be under estimated. In this thesis the weighing function was primarily that of the cutset measure. The power measure can be used instead. Gates of “acceptable” power dissipation will be picked up first. The ant will pick other gates on its way back to its colony only if the new gate power is less than that of the minimum power carried so far. To simplify matters the weighing function can only be based on the switching probability.

The combination of both of the gate clustering criterion and the weighing function will definitely produce a solution that is based entirely on one measure or be biased towards one. In this thesis the solution was entirely based on the cutset and other measures are merely calculated along. The solution fitness was calculated as combination of all other measures.

To achieve the best fitness combination, the clustering and the weighing function should be influenced by all considered measures.

4.5.4 Hill Climbing

One of the most important aspects of successful nondeterministic heuristics is a property called hill climbing. It can be explained simply as the ability to accept bad solutions to avoid being stuck in a small region of the solution space. Relatively good solutions could

lead to an area of the solution space that is called a local minimum, where it is very difficult to get out from to searching for a lower cost value of the solution space.

The Ant Colony heuristic as explained in this thesis has two inherent mechanisms to traverse as much as possible of the problem solution space.

The first hill climbing mechanism, the ant picks up the first gate in its round trip to its colony based on a preset parameter guaranteeing a fresh start each time the ant heads away for foraging. Sometimes the ant raids other colonies during foraging; in this case, the ant picks up a gate of higher weight and heads back to its colony. This produces the acceptance of relatively bad solutions.

The second hill climbing mechanism, after the completion of an iteration the gates are taken from the colonies and redistributed on the habitat cells. The gates will retain their partitioning assignment and ants will attempt to collect them and reassign them again. Habitat information, including pheromone clues, is reinitialized. Ants will attempt to re-explore the habitat while colonies have different relative positions. The raiding rate will be different and the paths to the gate clusters will be different as well. Many gate partitioning reassignments will happen, and thus, bad solutions will be accepted along the way.

Chapter 4 detailed the data structures and functions used by the algorithm. The following chapter explains the measures used to evaluate the intermediate solution introduced each time an ant deposits a gate into its colony.

Chapter 5

Cost functions

The power, delay, cut-set, and imbalance cost functions are combined using fuzzy rules to formulate an integrated value serving as an indication of how good or “fit” the obtained solution is.

First, the optimum values for the above mentioned cost functions are calculated. The optimum value is obtained when there is only one partition, i.e., all nets are not cut. The base cost is calculated by dividing the current iteration cost by the optimum cost. Next, the base cost is compared to the optimum and the worst value. If current value is better than or equal to the optimum value it will have a goodness membership of 1.0. On the other hand, if the current cost is worse than the worst cost value, then it is assigned a membership value of 0.0.

However, most of the values will lie somewhere in between these extremes. In this case, the cost function is calculated as follows:

Current member value = (base value – worst value) / (1-worst value)

The lowest membership value amongst power, delay, cut-set, and imbalance cost functions is selected and used to calculate the overall solution fitness value as follows:

$$\text{Fitness} = \beta * (\text{lowest membership value}) + (1 - \beta) * (\text{membership value for power} + \text{membership value for balance} + \text{membership value for cut-set} + \text{membership value for delay}) / (4.0);$$

Where β is a parameter set to 0.4.

The individual cost functions are detailed below.

5.1 Power cost function

The power cost function calculates the power dissipated by the nets connecting to more than one partition. Figure 16 below illustrates how the power dissipation is calculated.

For each net in the circuit

 If the net crosses to another partition

 Switching probability of the crossing net* 100.

 For each gate on the net

 Add parallel input capacitance of the gates on the net.

 Add the switching factor and the capacitance factor, the sum is the net contribution.

Circuit power dissipation = $(0.5 * 25) * (\text{“Frequency”} * 100 \text{ E } +6 \text{ “100MHz”}) * (\text{net-Contribution}) * 1 \text{ E } -15$

Figure 13 Power dissipation calculation function.

5.2 Delay cost function

The delay cost function calculates the total delay for each circuit long path. The maximum path delay is calculated as in Figure 17 below.

5.3 Cut-set cost function

The cut-set cost is incremented each time a net has at least a connection in another partition. The pertinent net is checked for each node in the circuit; once found, the net is checked if it has connections in the other partition. A net that has connections in more than one partition means these connections will have to be cut, i.e. increases the undesirable cut-set.

5.4 Imbalance constraint

The partitions imbalance is the difference between the numbers of nodes of the partitions. The imbalance constraint is verified to be within a predetermined value, namely the imbalance tolerance. That tolerance value is entered by the program user at the beginning of the execution. The final solution will not be accepted unless it is within the imbalance tolerance.

5.5 Fuzzy goodness evaluation

The three measures mentioned above (cutset, power, delay, and imbalance) are combined using fuzzy logic rules. This can be simply explained as the following:

For each of the measures, a value relating the current calculation to the best and worst achievable values for that measure is established (member value.)

Next, the minimum membership value of all measures is added with a weight parameter to the weighted sum of all measures.

The steps below show the fuzzy goodness calculations.

Goal_value = worst_value / Optimum_value "before partitioning"

base_value = Current_value/Optimum_value "before partitioning"

if Current_value better than the best

member_value = 1.0

else if Current_value worse than the worst

member_value = 0.0

else

member_value = (base_value - Goal_value) / (1 - Goal_value)

Pick the minimum_member_value

“Worst case scenario, pick the one with lowest membership.”

Parameter BETA = 0.4

fitness = BETA * minimum_member_value + (1-BETA) *

(member_power + member_balance + member_cutset + member_delay)/4.0

The Following chapter details how the ant colony concept is used to solve the circuit bi-partitioning problem

Chapter 6

Ant Colony Deployment

This chapter is dedicated to the details of how the circuit bi-partitioning is solved using the ant colony heuristic. The heuristic can work in two modes, constructive and iterative. In the constructive mode the gates are distributed over the habitat and they are not assigned to a partition. In the first iteration ants pick up the unassigned gates and deposit them into their colony where they are assigned to a partition for the first time. In the iterative mode, however, the gates are assigned to a partition and then distributed over the habitat. Ants pick the gates up and deposit them into their colonies where they will be reassigned to a different partition if the colony represents a different partition than that assigned originally to the gate.

The circuit gates are distributed in two different ways random and connectivity based. In the random distribution a random number between 0 and the square root of the number of habitat cells is generated for each (x, y) cell coordinate. The designated gate is inserted into the cell's bag. Where as in the connectivity based distribution, the gate with highest number of connections is placed in the middle of the habitat, the gates connected to it are

placed around in a circular fashion. When all the connected gates are placed, another gate is selected based on its connectivity and the same process is executed in a different location of the habitat. Finally, the unselected gates are placed randomly throughout the habitat.

The experiment showed using connectivity based distribution alone produced better results than when alternating gates distribution using both random and connection based. This is a proof of the pheromone important role in the heuristic because ants lead one another to the gates by the pheromone. Ants of the same colony approach the grid center from close paths and they should find massively connected gates and pick them up, and hence these gates will be assigned to the same partition.

In initial heuristic testing gates were distributed over habitat cells first by using connectivity based distribution, and then in the following iteration using the random distribution. The alternation of these distribution methods was thought to enhance the hill climbing characteristic of the heuristic. Random distribution will distort the solution and change its status fundamentally and will definitely pull the solution out of local minima. However, experiments showed this solution distortion is far too much to enable the solution to settle on a good solution state.

Colonies are placed at opposing corners of the habitat and if the habitat cell that is supposed to be the seed cell for the colony already has some gates, these gates are redistributed randomly. The colonies will dynamically grow and shrink in size throughout the duration of the iteration. The iteration will end when an ant deposits the last unassigned gate to its colony. When the iteration concludes the colonies move closer to each other by predefined user parameter “proximity.” These iterations are repeated until the distance between the colonies is less than proximity.

Changing the position of the colonies after each iteration changes the distance between colonies and gate clusters. The time it takes ants of a specific colony to reach the ant cluster will change. Ants of different colonies will have different chances of finding the gate clusters. Hence various gate partitioning assignments will occur. This feature will add to the hill climbing feature in a since it alter the solution state. In addition it promotes better habitat exploration.

Raiding plays an important role in the success of the heuristic. In addition to solution refinement it contributes directly to the heuristic hill climbing. Ants enter other colonies and pickup attractive gates. These gates are a burden on the original colony and they contribute to the solution quality degradation. By assigning these gates to a different colony’s partition a quality improvement to the solution could happen.

Raiding depends on the relative position of the colonies. As colonies approach one another, ants of different colonies will spend more time raiding than exploring new habitat areas.

As an ant finds the location of another colony, pheromone trails will direct other ants to follow and the raiding increases rapidly in short surges. Pheromone evaporation will increase to limit the pheromone growth and sustain some balance between raiding and exploring. In most cases a very good solution emerged after changing the proximity of the colonies and thus, enhancing the raiding.

Experiments showed that careful choice of the starting location of the colonies could help the solution quality of the heuristic. This observation could be explained in terms of gate clusters. Placing a colony near a cluster will give the ants of this colony a better chance of finding the gates and store them back. Both forage and storage paths to the cluster are shortened and hence the ant will be freed to make additional trips to the same habitat region.

The formulas below show how to calculate the colonies initial positions.

The location of the first colony uses formula (6) below.

$$X = \left[\frac{Grid_Cell_Size}{Number_of_Partitions^{number_of_partitions}} + proximity \right] \quad (6)$$

$$Y = \left[\frac{Grid_Cell_Size}{Number_of_Partitions} + \frac{Grid_Cell_Size}{Number_of_Partitions^{number_of_partitions}} - proximity \right]$$

The location of the second colony uses formula (7) below.

$$X = \left[\frac{Grid_Cell_Size}{Number_of_Partitions} + \frac{Grid_Cell_Size}{Number_of_Partitions^{number_of_partitions}} - proximity \right]$$

$$Y = \left[\frac{Grid_Cell_Size}{Number_of_Partitions} - \frac{Grid_Cell_Size}{Number_of_Partitions^{number_of_partitions}} + proximity \right]$$

(7)

After the colonies positioning, ants start foraging from their nest locus. In their way, ants deposit pheromones during their trip of exploring habitat cells. When an ant finds a gate, it extracts the pointer to the gate, the habitat cell will lose the gate pointer and the ant will gain custody of it.

The ant will try to retrace its way back to the colony by sensing pheromone values it had updated on its way out. On its way back it changes the values of the pheromones on the grid cells using a different function than that it uses in its foraging cycle.

The ant reaches its colony with a probability, and the way is marked for other ants to sense. Storing the gate in the colony will mark that gate pertinent to the partition that colony subscribes to. This is done by updating the circuit structure. The ant requests a mutual exclusive lock on the circuit structure.

Upon acquiring the necessary permission, the ant initiates a gate move to the new partition. This is done by simply updating the partition value of the gate data structure.

Next, the nets that are connected to the gate are identified. Each of these nets requires the update of the respective partition number of gates. The net partitioning information is an array having a total number of elements equivalent to the number of partitions. In the bi-partitioning case the array will have two elements. Each array element stores the total number of gates of that particular net that belong to the partition representing the array index. The partitioning array belongs to the net data structure.

Finally, the gain information is calculated. Each gate data structure contains a pointer to a specific gain array (gain bucket.) The index of the gain array represents the cost gain

caused by assigning the gate to the particular partition. Each gain element of the array contains pointers to the gate structures having the same gain values. Using this gain bucket mechanism enables us to know which gates have a specific gain weight, and in the same time, what is the gain in assigning a gate to a particular partition.

For gates not assigned to a colony, the gate distribution over the habitat cells does not correspond to that of the partition. Sometimes ants pick up gates that are already assigned to their partition and deposit them in their colony. Since the colony subscribes to the same partition the gate originally assigned to, the move mentioned above and the gain are not recalculated. In this case, the gate is only assigned to the colony.

As mentioned previously, the heuristic works in two modes, constructive and iterative. In the constructive mode gates are not assigned to a partition originally, after being deposited in a colony, gates will be assigned a partition for their first time. The constructing mode is concluded when an ant picks up the last unassigned gate and deposits it in its colony and hence it is assigned to that partition.

In the second “iterative” mode, the Ant Colony Optimization heuristic iterates to improve the starting solution. Gates are redistributed (randomly or according to gate clustering criteria) on the habitat grid cells. This time the gate partition information had been

marked and the cost of assigning each gate to a partition is evaluated against the previous assignment. The heuristic iterates in this mode until no further improvement is achieved.

The heuristic's hill climbing capability (escaping local minima) is further enhanced by changing the relative location of the colonies after the conclusion of each iteration. The purpose of this technique, as explained earlier, is to dynamically influence the "Raid" function of the ants. Ants from different colonies will raid other nests and assign the most suited gates to their own partition. This results in a much better iterative improvement capability for the heuristic.

The solution fitness is then evaluated based on power, delay, and cutset and the results are combined in a fuzzy fitness measure as explained in the cost functions chapter.

The following chapter lists the results obtained when solving the circuit bi-partitioning problem with the Ant Colony algorithm as explained in this thesis.

Chapter 7

Experiments and Results

This chapter presents the test circuits which were used to test the Ant Colony heuristic and the experiment results. The results were obtained by evaluating the gate assignments based on the cutset, and then selecting the more comprehensive fitness measure to report the outcome of the assignment.

The algorithm showed sensitivity to user parameter values and thus, careful choice of the parameters boosted the resulting fitness value greatly. Two sets of parameter values produced very good results for circuits of small number of gates and those of medium size.

To help the visualization of the problem solved by this algorithm, a simple circuit was used and the heuristic was run different times. Each time the result was different both in the cutset and in the gates assigned to each partition. Both of the assignments produced cutset of 3 at perfect partition balance. The circuit has 16 nodes eight of which were assigned to the first partition (the shaded gates) and the other eight were assigned to the

second (the un-shaded.) It is clear that the two gate assignments are not just mirror image of each other. Hence, the randomness in the heuristic produces totally different results in each run.

7.1 Circuit details

ISCAS-85/89 benchmark circuits were used in testing the heuristic. Most of these circuits are sequential, and thus contain feedback loops and have delay problems. Circuit characteristics data is listed for 0.25 μ MOSIS, TSMC, and CMOS technology library [9].

To appreciate the complexity of the simplest benchmark circuit, the minimal cutset configuration obtained by the heuristic is shown in Figure 20. Circuit “s298” has total of 136 gates which are connected using 130 nets. After the heuristic is run the number of nets cut is 10. To simplify the Figure the gates directly connected by the nets that are cut are the only ones shown.

Table 7 shows circuit’s characteristics and the gates characteristics are shown in Table 8 below.

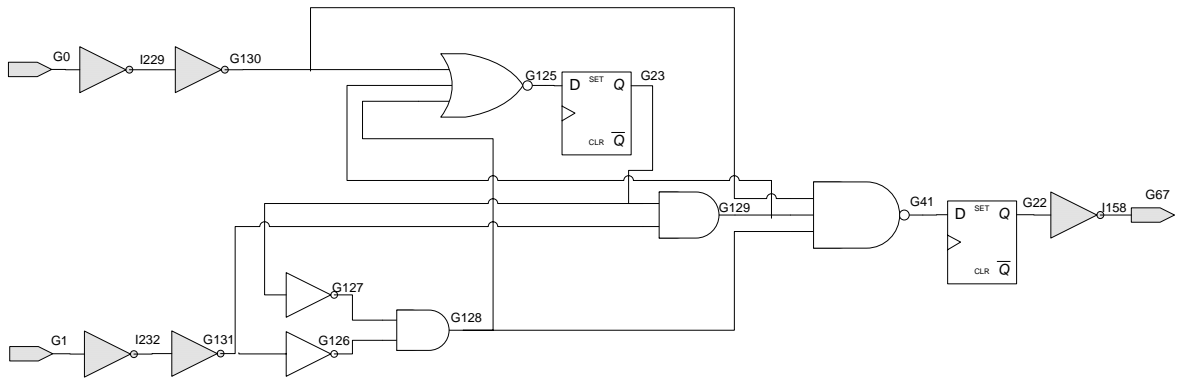


Figure 15: One gate assignment configuration.

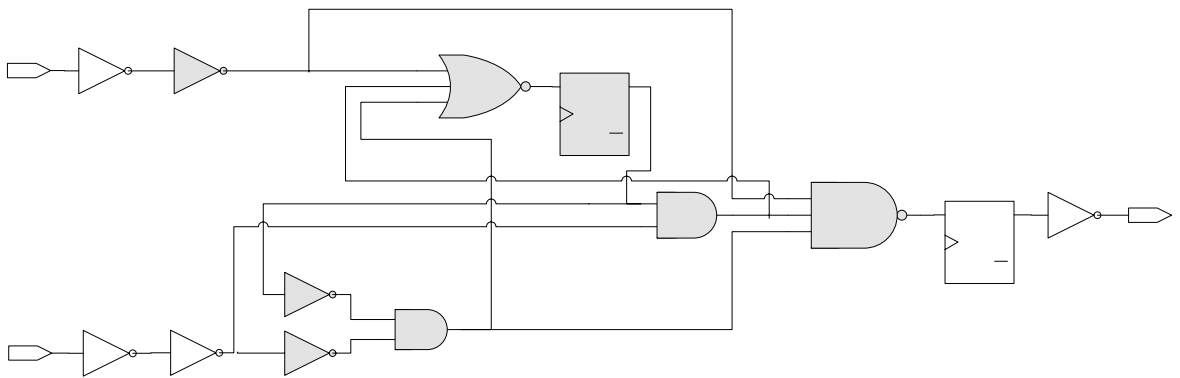


Figure 16: Another gate assignment configuration.

Name	Number of cells	Number of nets
S298	136	130
S386	172	165
S641	433	410
S832	310	291
S953	440	417
S2081	122	121
S15850	10384	10296

Table 1: Circuits characteristics.

Input	Type	Name	Width	capacitance(fF)	Load factor(ohm)	Cell Delay(ps)
1	not	i2s	2.7	2.661	516	3
2	nand	ai2s	3.24	2.661	706	5
3	nand	ai3s	4.32	2.661	661	6
4	nand	ai4s	5.4	2.661	663	7
2	nor	oi2s	3.24	2.661	694	7
3	nor	oi3s	4.32	2.661	1016	9
4	nor	oi4s	5.4	2.661	1292	17
2	xor	exors	7.02	5.321	688	12
2	xnor	exnors	6.48	5.321	551	12
2	and	a2s	4.86	2.661	438	9
3	and	a3s	5.94	2.661	607	13
4	and	a4s	7.56	2.661	688	16
2	or	o2s	4.32	2.661	707	11
3	or	o3s	5.4	2.661	943	16
4	or	o4s	6.48	2.661	1201	25
1	dff	dsr2s	18.9	2.661	432	38
1	buf	tsbuffs	3.24	2.661	370	7
0	INPUT	INPUT	0	0	0	0

Table 2: Gates characteristics.

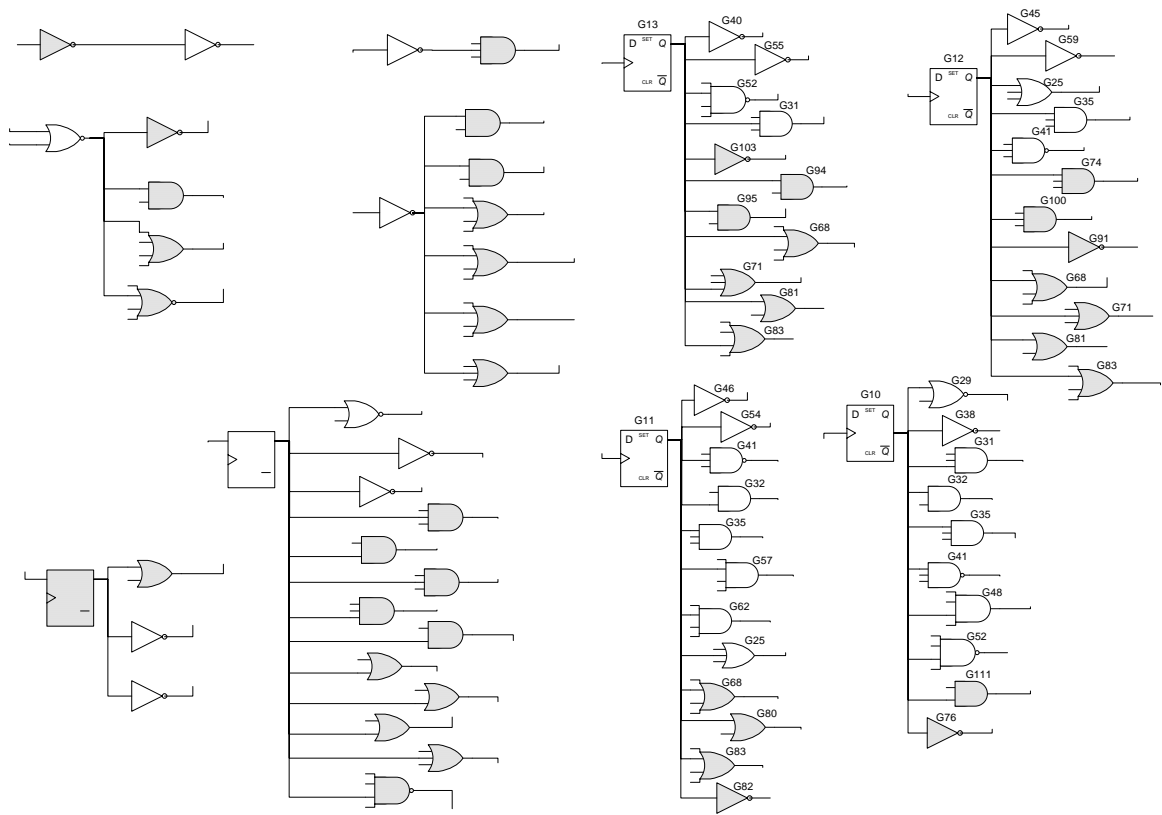


Figure 17: the minimum cutset nets of circuit s298.

Out of the 130 nets, random assignment of the gates to the partitions resulted in a cutset of 81. After the first 67 ant round trips, the cutset cost dropped to 33. On the 1303rd and up to 1307, the cutset cost reached its absolute minimum of 10. The best solution is always saved and the heuristic continues on searching the solution space.

Circuit configurations are evaluated on the cutset cost, power dissipation, and delay of the longest path through the entire circuit. Optimizing the solution based on one measure will make the result worse for the other measures.

7.2 Results and comparisons

The Ant Colony heuristic produced very encouraging results for small and medium sized benchmark circuits.

Table 9 below shows the results of test experiments for small and medium benchmark circuits. All results are obtained for imbalance criterion 0.1, which means the number of gates in partitions should be equal $\pm 10\%$. This constraint was used to enable the comparison of this work to that obtained for other algorithms Tabu Search, Genetic Algorithm, and Simulated Evolution [9].

Circuit	SOP Cutset	MOP Cutset	Imbalance	Power	Delay	Fitness
s298	10	10	5.88%	1143	232	0.787218
s386	28	29	3.48%	2001	342	0.711692
s641	44	56	9.00%	3101	1118	0.782923
s832	33	42	0.64%	3863	401	0.682903
s953	79	89	6.36%	3985	510	0.66777
s2081	10	15	1.63%	914	314	0.755474
s15850	5530	5530	1.63%	112600	3409	0.224169

Table 3: Ant Colony results.

The second column of the table is the cutset using single optimization technique "SOP" where the cutset is accepted only if it meets the imbalance constraint. These values are excellent and portray the potential of the new heuristic.

The next column is the cutset evaluated in a multi-objective way. The cutset cost is accepted only if it produces a high fitness value. Where the fitness of the solution is a composite of all other measures combined using the fuzzy rules explained earlier in the cost function chapter. The balance in these experiments was below the constraint of 10% also the power and delay costs have decreased considerably.

To get a feeling for the improvement, Table 10 below shows the original cutset, power and delay costs, the values are obtained by assigning the gates to partitions randomly.

Table 11, Table 12, and Table 13 below shows the result obtained from the previous work using Genetic Algorithm, Tabu Search "TS", and Simulated Evolution, respectively [9].

Circuit	Cutset	imbalance	Power	Delay	Fitness
s298	81	0%	2806	473	0.206977
s386	81	0%	3720	720	0.226829
s641	226	0.23%	8605	2974	0.217482
s832	148	0%	6829	750	0.223966
s953	268	0%	6429	1099	0.203726
s2081	73	0%	2499	727	0.210000
s15850	5569	0%	113016	3436	0.218873

Table 4: Original values of the circuits.

Circuit	SOP Cutset (nets)	MOP Cutset (nets)	Delay (ps)	Power (S.P.)
S298	18	19	233	1013
S386	29	36	356	1529
S641	44	45	1043	2355
S832	44	45	444	3034
S953	93	96	526	2916
S2081	16	26	302	787
S15850	2001	2183	1820	51747

Table 5: A Comparison between the quality of the best solutions obtained from GA by performing SOP for cut-only and MOP [9].

Circuit	Cutset (nets)	Cutset (nets)	Delay (ps)	Power (S.P.)
S298	13	24	197	926
S386	18	30	386	1426
S641	51	59	889	2281
S832	35	50	446	2731
S953	76	99	466	2518
S2081	5	17	225	770
S15850	1545	1671	1411	47480

Table 6: A Comparison between the qualities of the best solutions obtained from TS by performing SOP for cut-only and MOP [9].

Circuit	Delay (ps)	Cutset (nets)	Power (S.P.)	Fitness
S298	197	11	837	0.95
S386	393	28	1696	0.74
S641	886	16	1738	0.98
S832	400	39	3132	0.691
S953	476	48	2473	0.93
S2081	325	13	706	0.94

Table 7: Best solutions obtained from SimE by performing MOP [9].

The results for small and medium sized circuits are quite impressive for a new heuristic. The small circuits required different user parameter setting than the medium ones. The parameters and their values will be listed in a following section. The parameters for medium sized circuits were applied for the larger size circuit s15850, but the results were very bad. The ants were exploring only a very small area of the habitat. Further work has to be done to come up with parameters that are suitable for this class of circuits.

Another observation is the power results were consistently worse than that of the other algorithms. The reason for that is the weighing function in which the gate weight is solely dependent on the cutset. A better approach would be to evaluate the weight based on the fitness and report the results when the ants deposits the gates based on the required measure.

Because of time limitations on the development of this project, the first mode of the heuristic, which is to construct the solution, was implemented, but the results were not established. In the current version of the heuristic the program run starts with gates assigned to partitions randomly and the heuristic starts directly in the iterative mode. For the heuristic to work in the constructing solution mode, the ants must collect all the gates completely, and this requires the parameters to be set accurately enough ensuring every habitat cell is visited by many ants of every colony within reasonable time. This was only done for the benchmark circuit s298. Further improvement can be reached by setting the

user parameters programmatically using another heuristic as explained in the heuristic parameters section

Although the results are quite good, the most important aspect of this heuristic is the search trend it exhibits while it travels through the solution space. The following section is dedicated to illustrate the Ant Colony search trend.

7.3 Trend analysis

The cutset, power, delay, and fitness values are calculated under the imbalance criterion ($\pm 10\%$) and the plots are listed below.

In the analysis for s386 circuit (Figure 22) the cutset, power, and to some degree the fitness, exhibited excellent trend as a balance of random and organized search. This is quite evident from the decrease of the costs rapidly in the beginning of the run, and then the search in the remaining solution space emphasizing the hill climbing characteristic of the heuristic. The delay did not show the same search pattern, because it was not accounted for during the gate weighing, and its connection to the cutset calculation is not as strong as that of power.

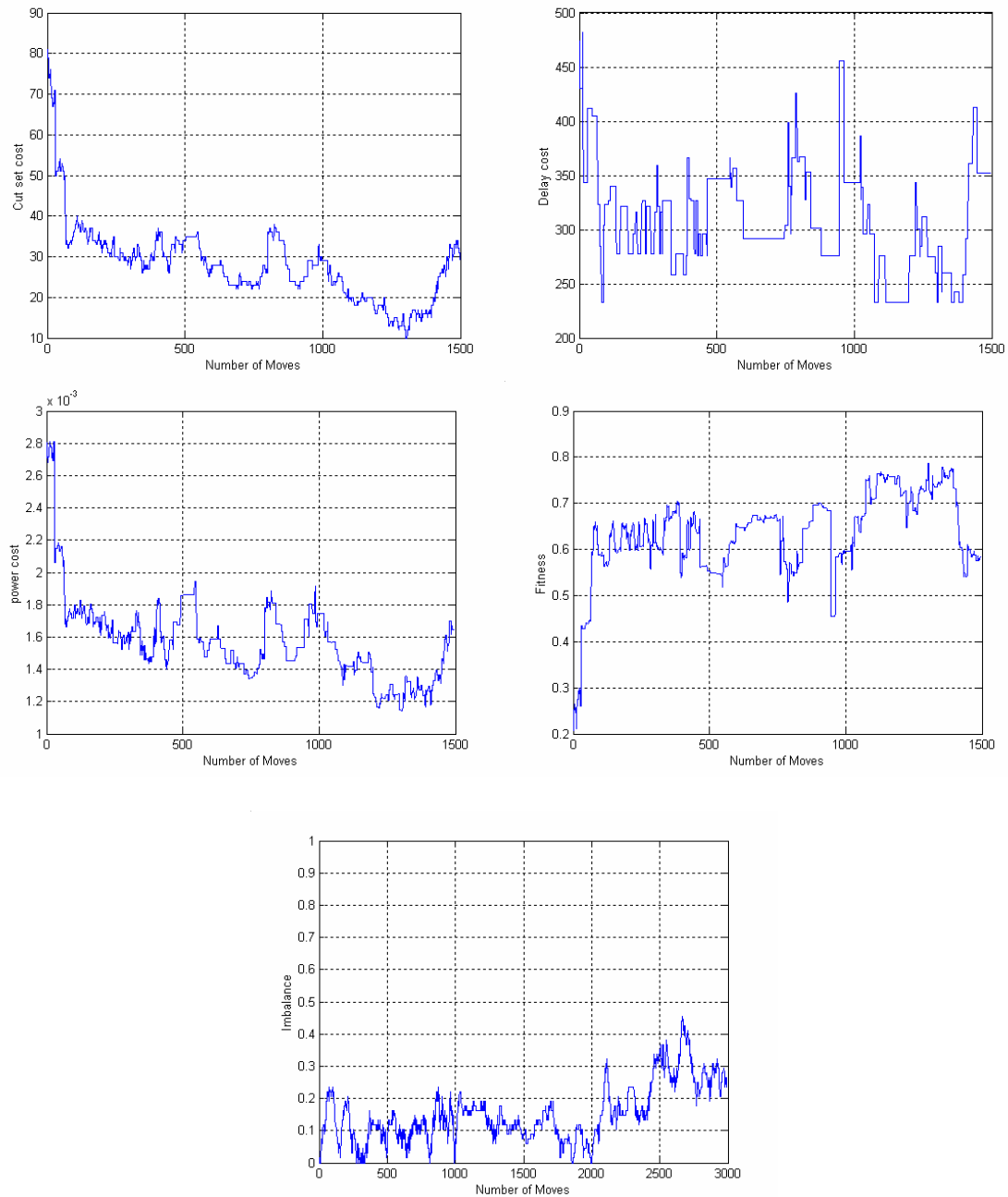


Figure 18: Cutset, delay, power, fitness, and imbalance of s298.

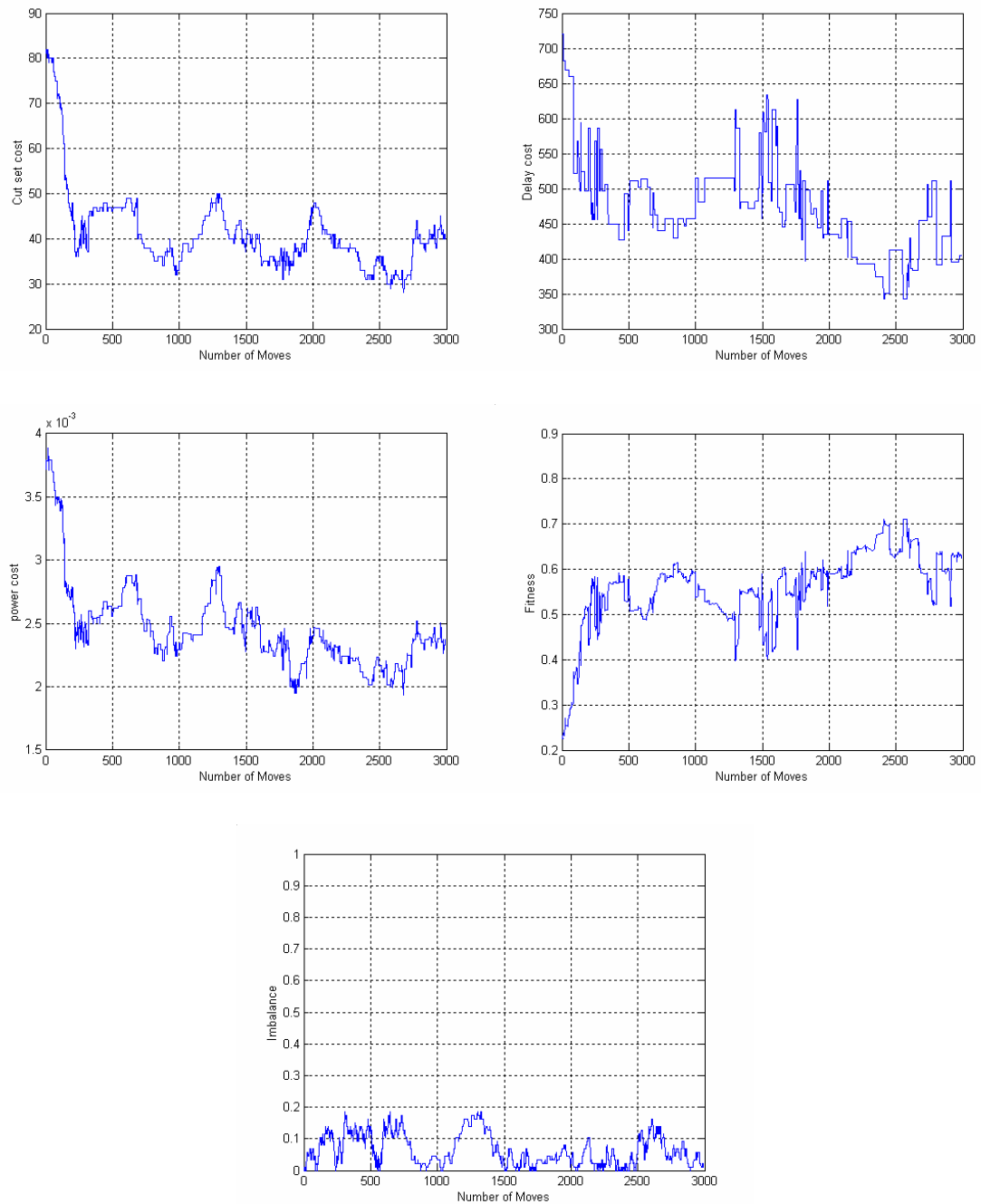


Figure 19: Cutset, delay, power, fitness, and imbalance of s386.

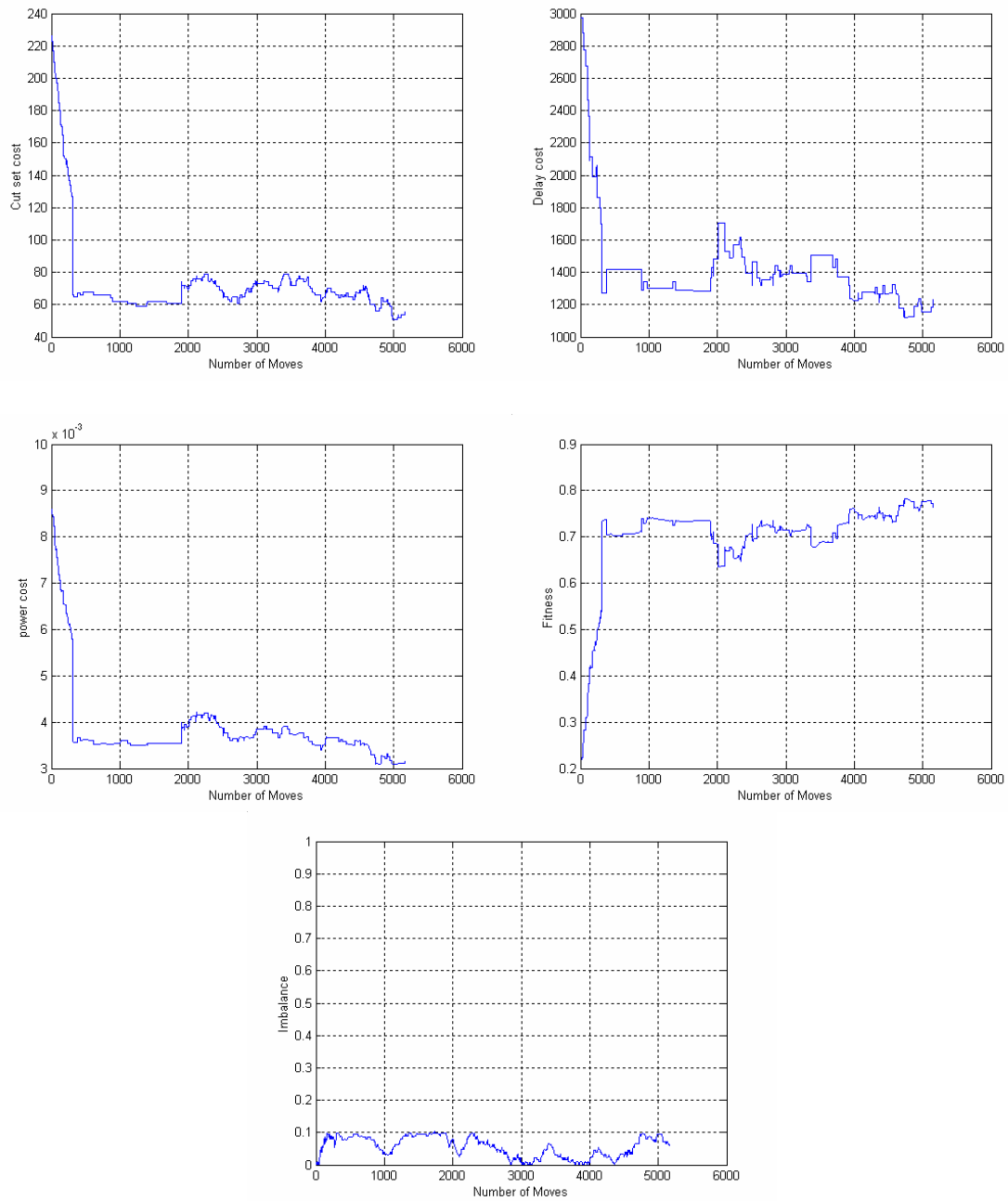


Figure 20: Cutset, delay, power, fitness, and imbalance of s641.

Circuit s641 (Figure 23) results, exhibit another good example of the heuristic characteristics. The decrease in costs is very sharp in few number of iterations, with very small range of cost variations portrayed as less noise in all the graphs. In addition, all the measures indicated high quality solution.

Circuit s832 (Figure 24) exhibits many small variations to the values of cutset and fitness which is shown as noise on these graphs. The search for delay was very much random, since no search pattern is evident in its graph. As suggested earlier, considering the delay in the weighing function will improve this important measure.

Circuit s953 (Figure 25) exhibits another good example for all measures. The bad route the heuristic took after 1500 ant trips was not a total loss because a good recovery happened around 2000 trips, and in fact that contributed to the over all success. Accepting bad decisions that may lead to better results is an important criterion of successful nondeterministic heuristics.

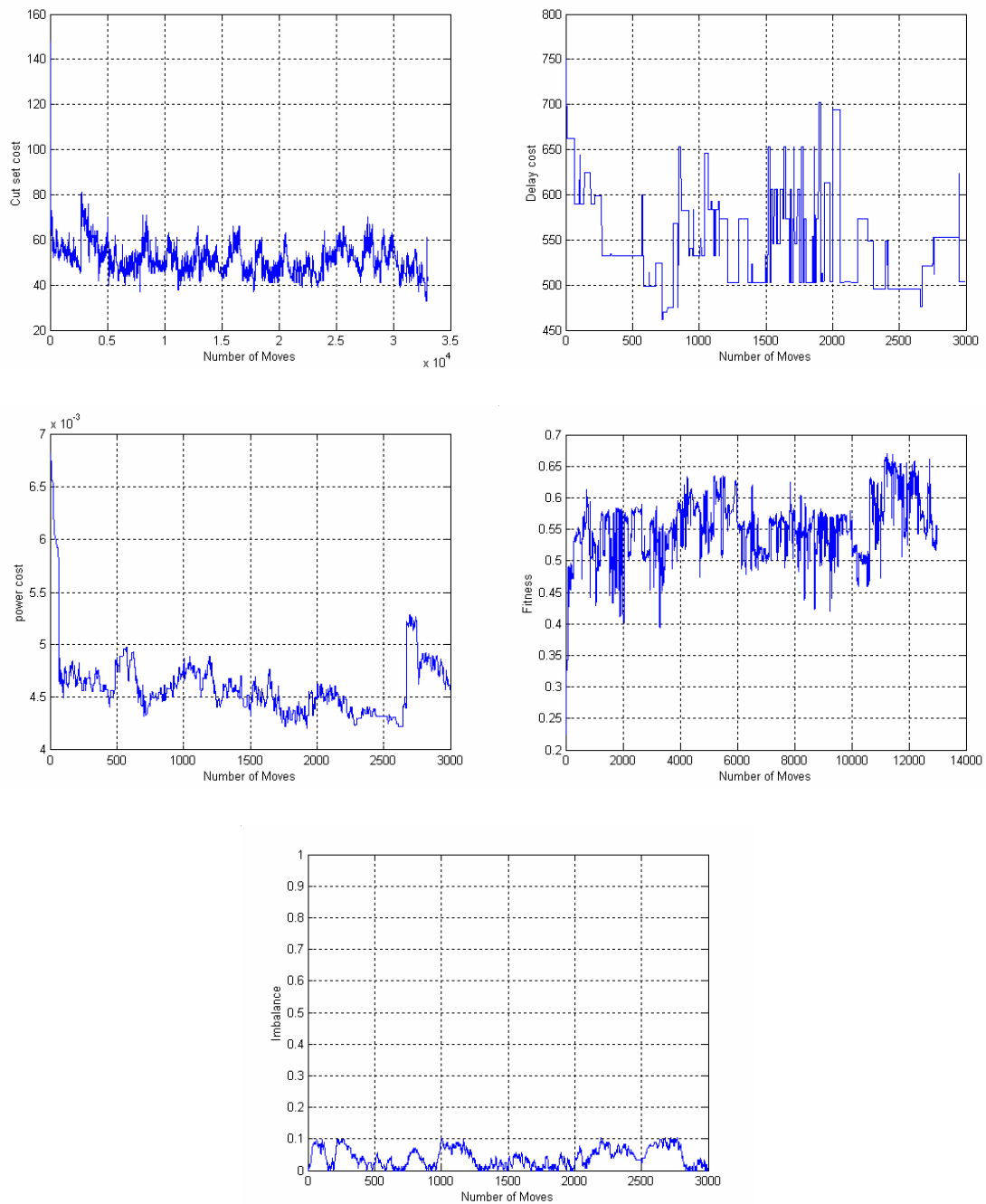


Figure 21: Cutset, delay, power, fitness, and imbalance of s832.

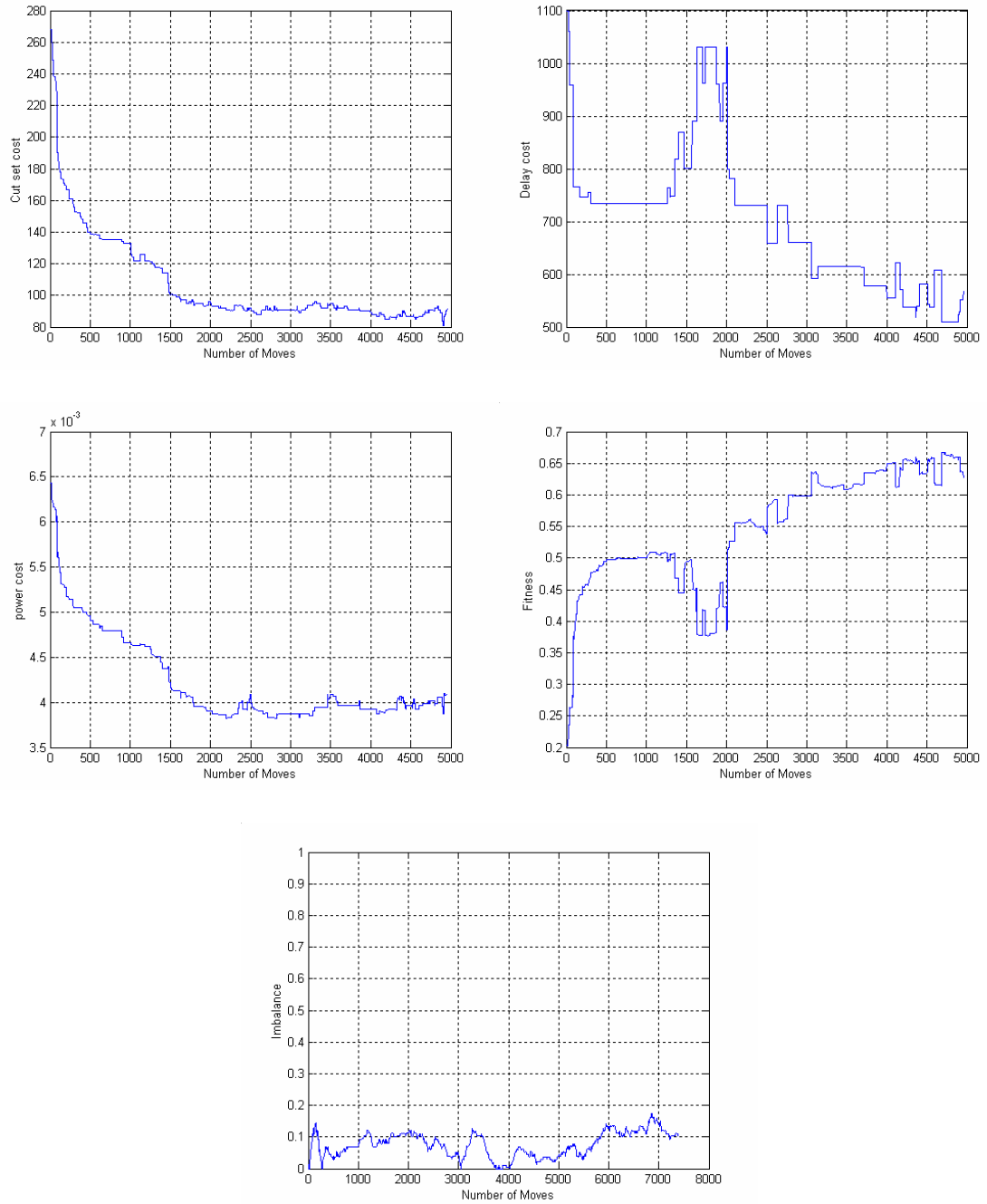


Figure 22: Cutset, delay, power, fitness, and imbalance of s953.

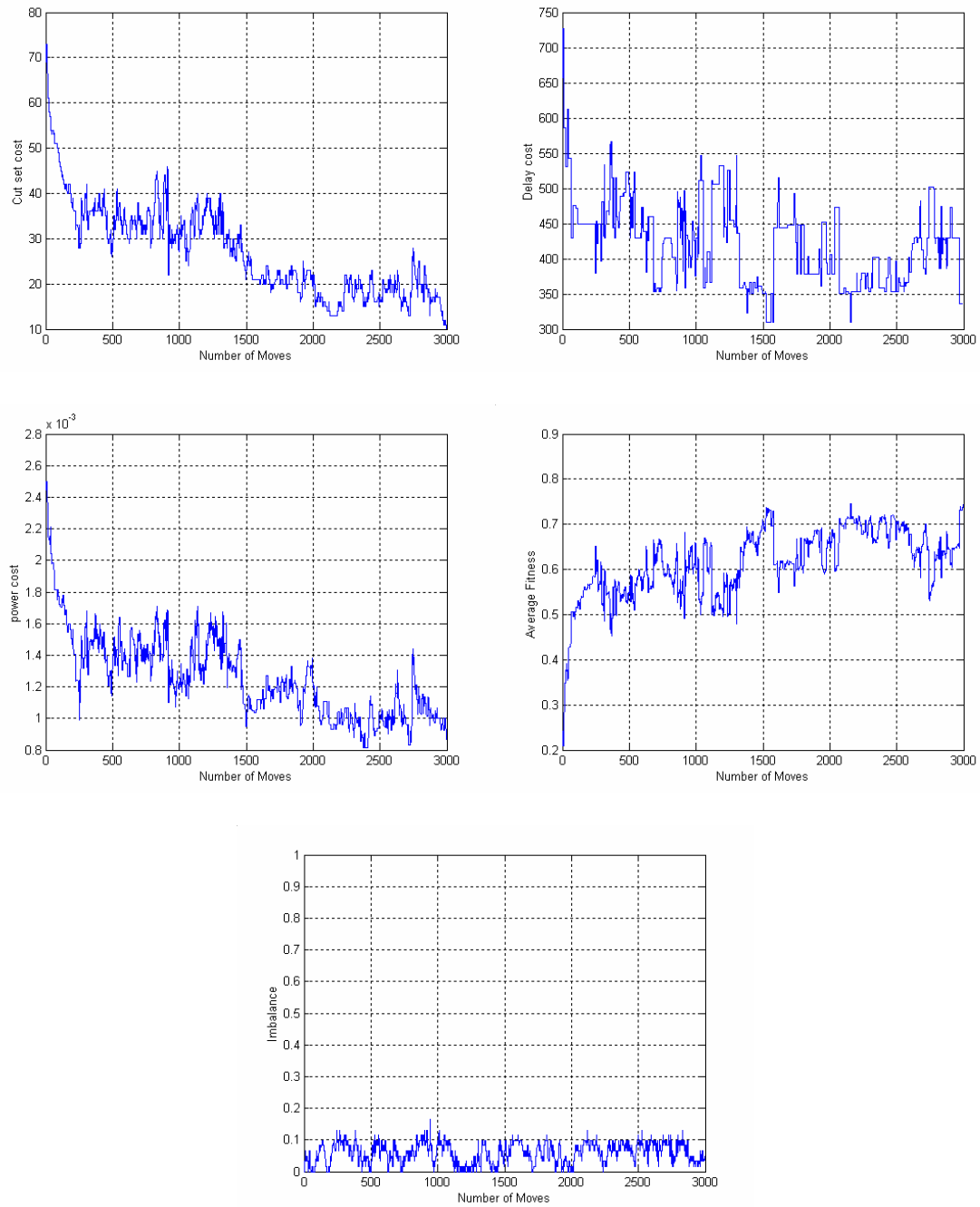


Figure 23: Cutset, delay, power, fitness, and imbalance of s2081

Circuit s2081 (Figure 26) exhibits another good example of overall measures. Some variations in the delay search affected the quality of the solution. The decrease in the cutset cost, however, is consistent throughout the entire run which is another healthy sign.

Table 14 shows that the total number of parameters is 36, twenty of which (marked in bold) were changed to provide better results for both small and medium circuits. Careful study of the heuristic would yield the elimination of some of these parameters.

To set these parameters programmatically Simulated Evolution is suggested to find a better matching set of parameters. Figure 27 below shows a proposed pseudo code.

7.4 Heuristic parameters

Parameter name	Medium Circuit	Small Circuit
Number of partitions	2	2
Habitat cell bag capacity	5	10
Initial cell pheromone	0.1	0.1
Colony cell pheromone	1	2
Initial proximity	0	0.1
Proximity increment	0.05	0.1
Habitat cell number factor	0.05	0.13
Number of colony ants	5	5
Evaporation rate	0.000015	0.000015
Evaporation threshold	10000	1000
Forage bread crumbs increment	1	1
Forage pheromone max	4	4
Forage pheromone min	0.1	0.1
Forage pheromone X scale	1	4
Forage pheromone Y scale	1.5	1.5
Help weight	1	1
k-forage	1	3

Lost ant	-1500	-200
Min pheromone level	0.1	0.05
Pheromone storage limit	4	4
Pheromone weight	1	3
Pick up parameter	1	2
Reorient penalty	2	3
Scout limit	0.1	0.4
Site colony	0	0.1
Site weight	1	1
Size weight	3	2
Step loop factor	30	50
Step time factor	10	10
Storage bread crumbs increment	1	1
Storage pheromone X scale	1	10
Storage pheromone Y scale	2	4
Not much heavy	Input - 3	Input - 3
Too Much heavy	Input - 1	Input - 1
Vision limit	0.1	0.1
Ant's payload	1	3

Table 8: The heuristic parameters for small and medium circuits.

```

Simulated Evolution (ant colony parameters, parameter values)
{
// ant colony parameters: parameters of the Ant Colony heuristic.
// parametersValues: values calculated by Simulated Evolution.
INITIALIZATIONS:

- Bias = -0.2, select more parameters for mutating reducing error in parameter optimal value.
- W “number of trials for individual selected parameters” = [-%10, +%10] of previous value
- Oi “Optimal value for individual parameter” = choose a value of a super ant.


Repeat
  Evaluate and Select
  For Each parameter in ParameterArray [Parameter, SelectedFlag, importance]
    Goodnessi = OptimumParameter /currentParameterValue
    IF Random() <= 1 - Goodnessi + Bias Then
      ParameterArray [Parameter, SelectedFlag = true, importance]
    Else
      ParameterArray [Parameter, SelectedFlag = false, importance]
    End If
  End For Each
  Sort elements of ParameterArray [Parameter, SelectedFlag, importance] by importance
  Allocation:
  For Each parameter in ParameterArray [Parameter, SelectedFlag, importance]
    IF ParameterArray [Parameter, SelectedFlag = true, importance]
      Change parameter value by W= [-%10, +%10]
      Run Ant Colony
      IF currentCutset < minimumCutset Then
        ParameterArray[Parameter, SelectedFlag=true, importance++]
      End IF
    End IF
  End For Each

Until all parameter value ranges have been tested
Return (BestSolution)
}End Simulated_Evolution

```

Figure 24: Using simulated evolution to calculate the parameter values.

Chapter 8

Conclusion

The main achievement of this endeavor is to introduce a new heuristic that explores the problem solution space and to demonstrate the capability of getting out of local minima. In addition, it exhibited a definite search pattern that is not totally random. Figure 27 below gives an example of a random search pattern. The Figure exhibits no pattern or convergence path that can be established. This is in contrast with the clear pattern established by the new Ant Colony heuristic. The Figure is reproduced below for easier comparison.

Success of the algorithm is attributed to the fine mixture of simple definite steps and decisions based on random functions. This mixture of order and randomness is a characteristic of many processes in nature and is classified as chaotic. In this way, chaos was an advantage to bring forth very good results, but as in many systems in nature, it produces complications. One such example is the heuristic parameters.

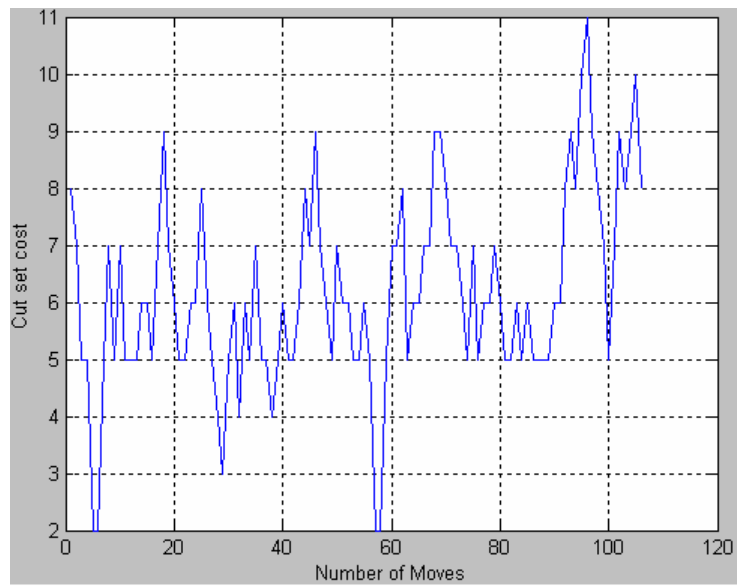


Figure 25: Random search pattern.

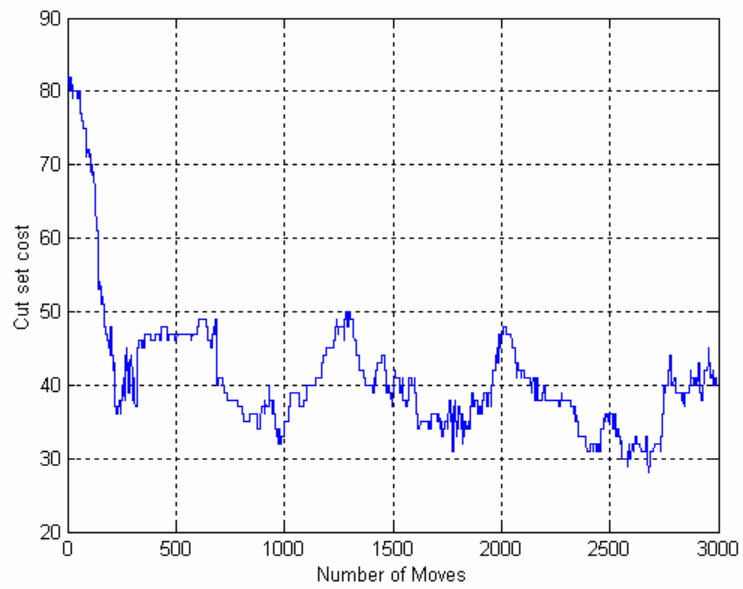


Figure 26: Definite pattern of nondeterministic hill climbing search heuristic.

Each time the heuristic is run, it produces different results, even when fixing up all the conditions. It requires careful study to isolate parameters that should not be changed for each class of circuits. One such circuit classification could be based on the number of gates.

Simulated Evolution heuristic was chosen to fine tune the Ant Colony heuristic parameters. The way the Ant Colony heuristic is run is to complete many number of iterations while progressively pulling the colonies closer to each other. The beginning of such iterations is a good time to start a new set of parameter values. This qualifies Simulated Evolution as a Meta-heuristic which can be pursued in future work.

Many of the decisions made in this work were based on common sense and intuition. Some choices could have been selected in a better way, but the final result proved the potential of the concept.

It is clear that the ants were not aware of the general search pattern made by the heuristic. The simple ant rules of instinctive behavior did not influence the solution pattern directly. Hence, the “super organism” concept was portrayed. The work can be further enhanced to eliminate the extra feature of searching for the colony when everything else fails.

The operating system and the type of thread technology used limited the number of ants that can work simultaneously. .Net Framework threads would have been a better choice under the same circumstances. But at the time of the early development, I was not familiar with these powerful tools. In addition, UNIX is known for its superior parallel processing capabilities and probably it would have been a better operating system candidate.

The pseudo random generators used had very limited unrepeated patterns. The choice of a more sophisticated random generator could have added to the power of the heuristic.

Appendix A

This appendix is dedicated for graphs of the same benchmark circuits processed at 0% imbalance criterion. That means the difference in the number of gates of the partitions would not exceed 0.005 of the total number of gates. Some benchmark circuits have odd number of gates and it is impossible to have exactly 0, for circuits of even number of gates perfect balance was achieved.

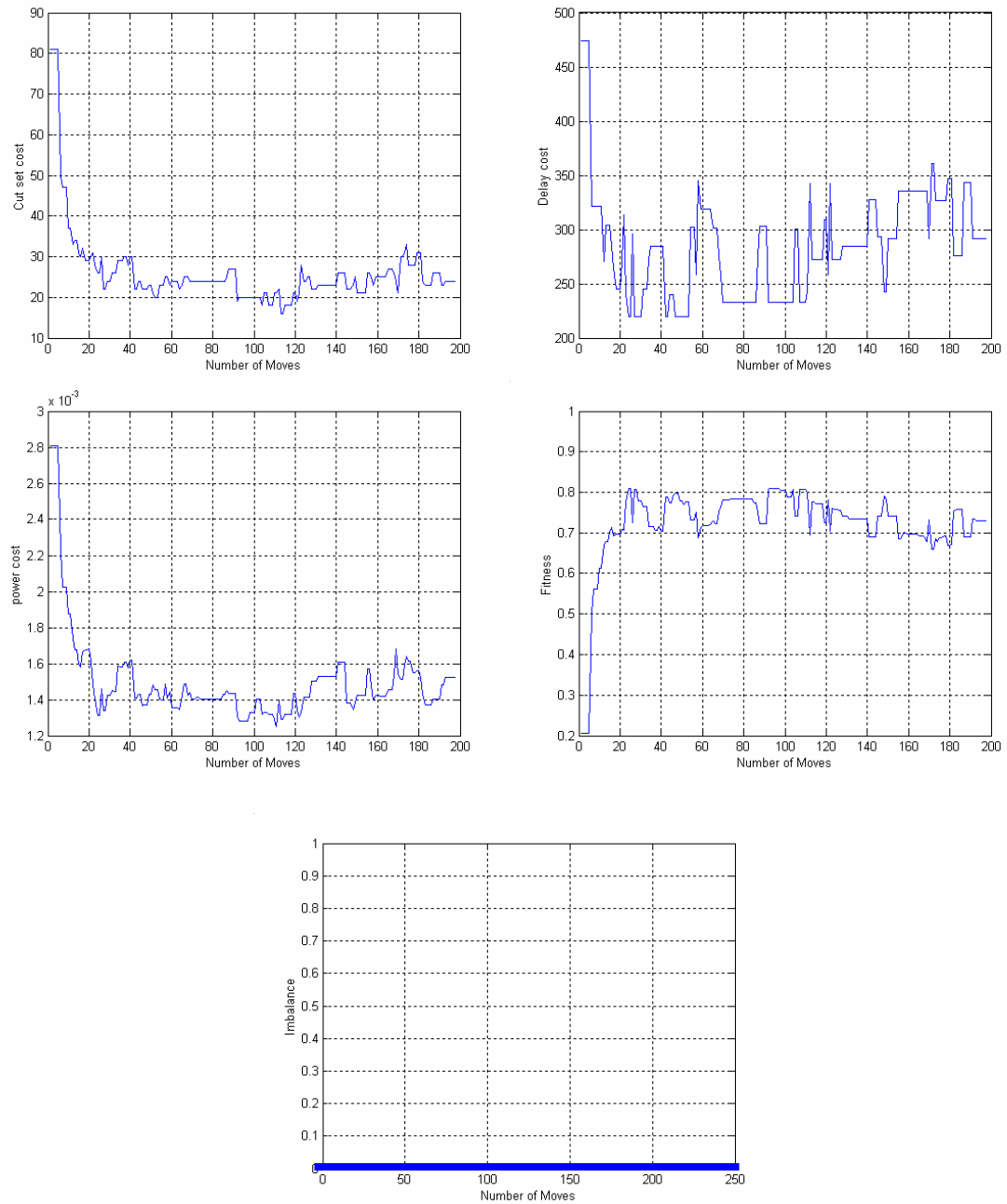


Figure 27: Cutset, delay, power, fitness, and imbalance of s298

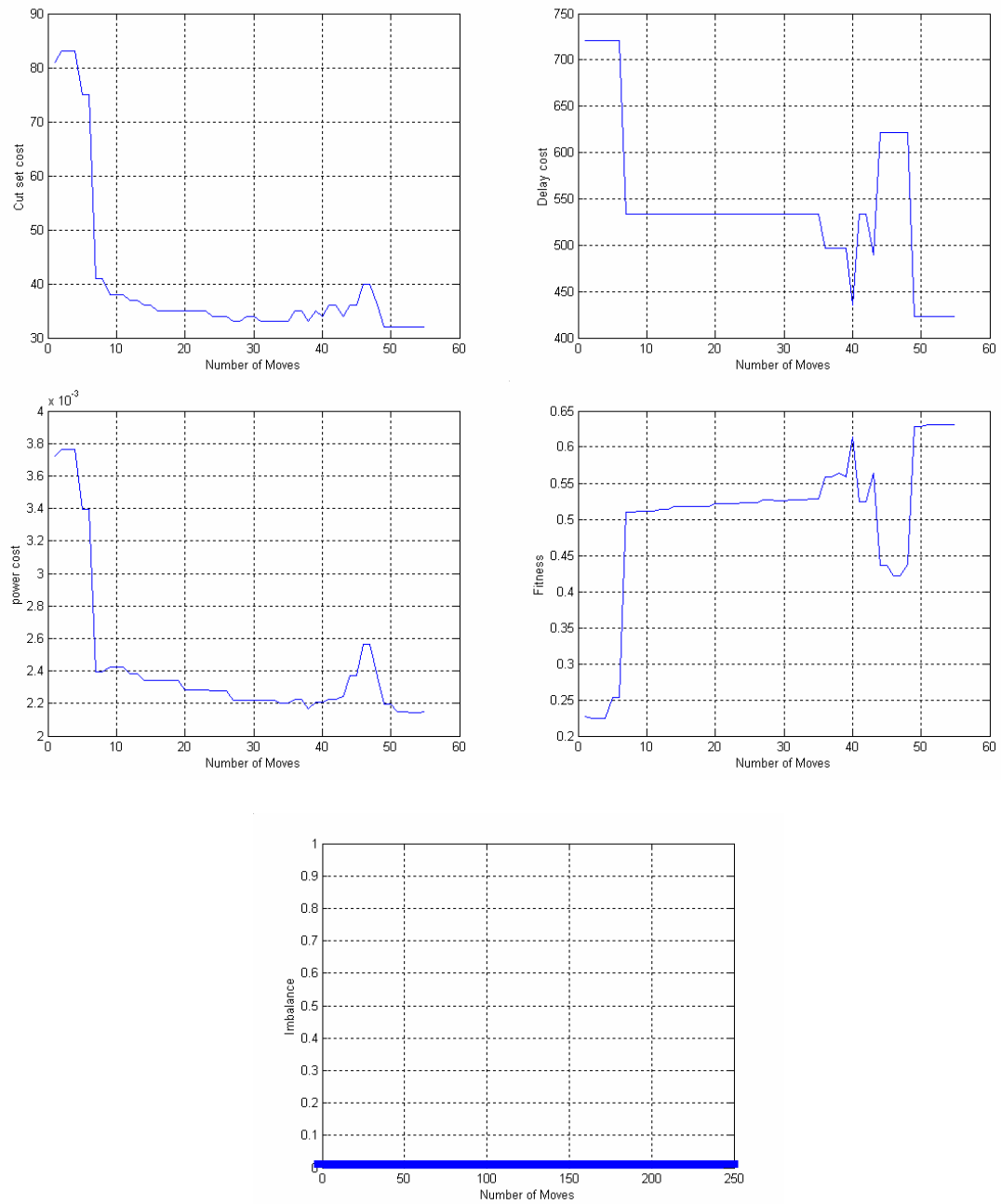
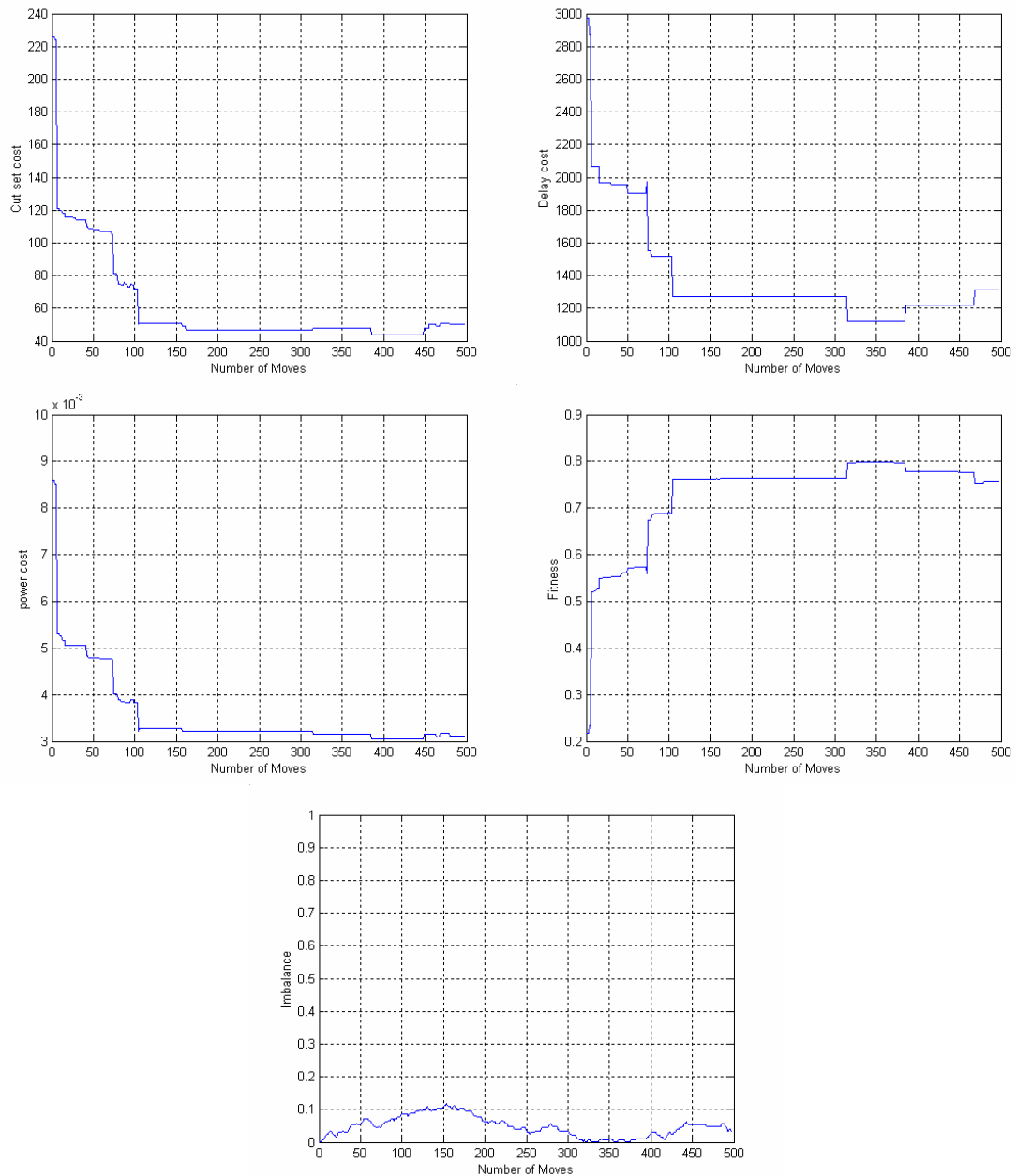


Figure 28: Cutset, delay, power, fitness, and imbalance of s386



Odd number of gates 0% imbalance can not be achieved.

Figure 29: Cutset, delay, power, fitness, and imbalance of s641

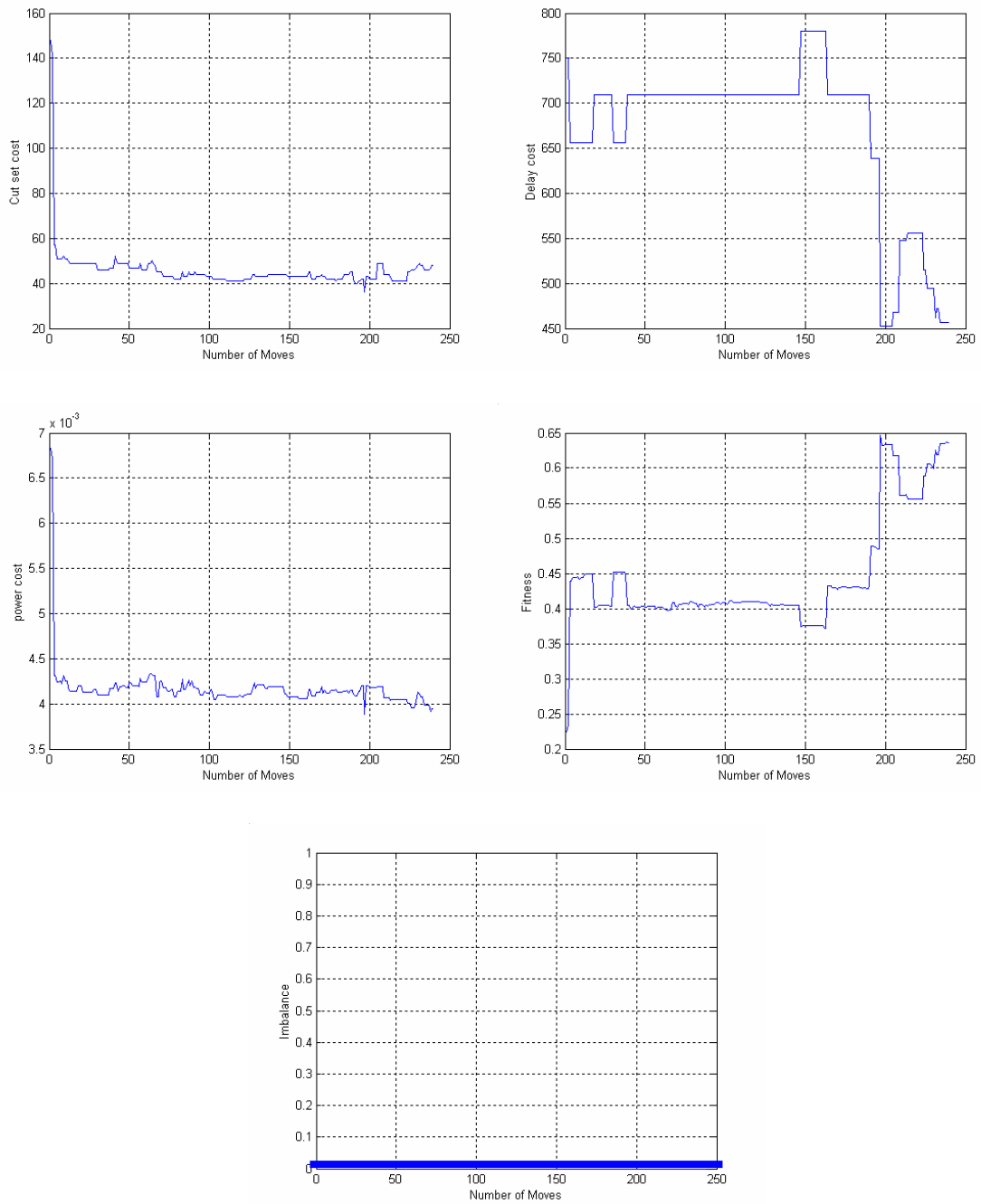


Figure 30: Cutset, delay, power, fitness, and imbalance of s832

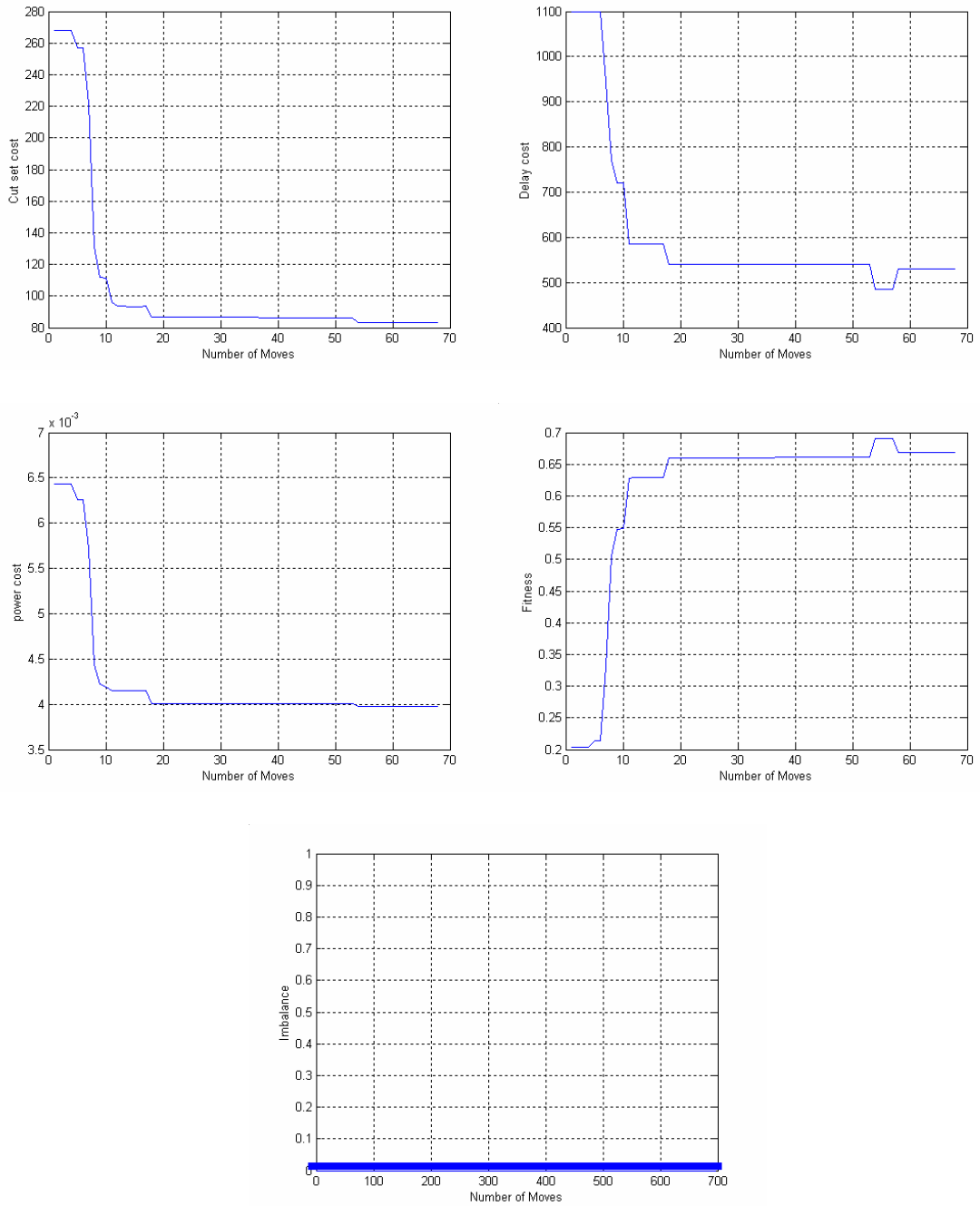


Figure 31: Cutset, delay, power, fitness, and imbalance of s953

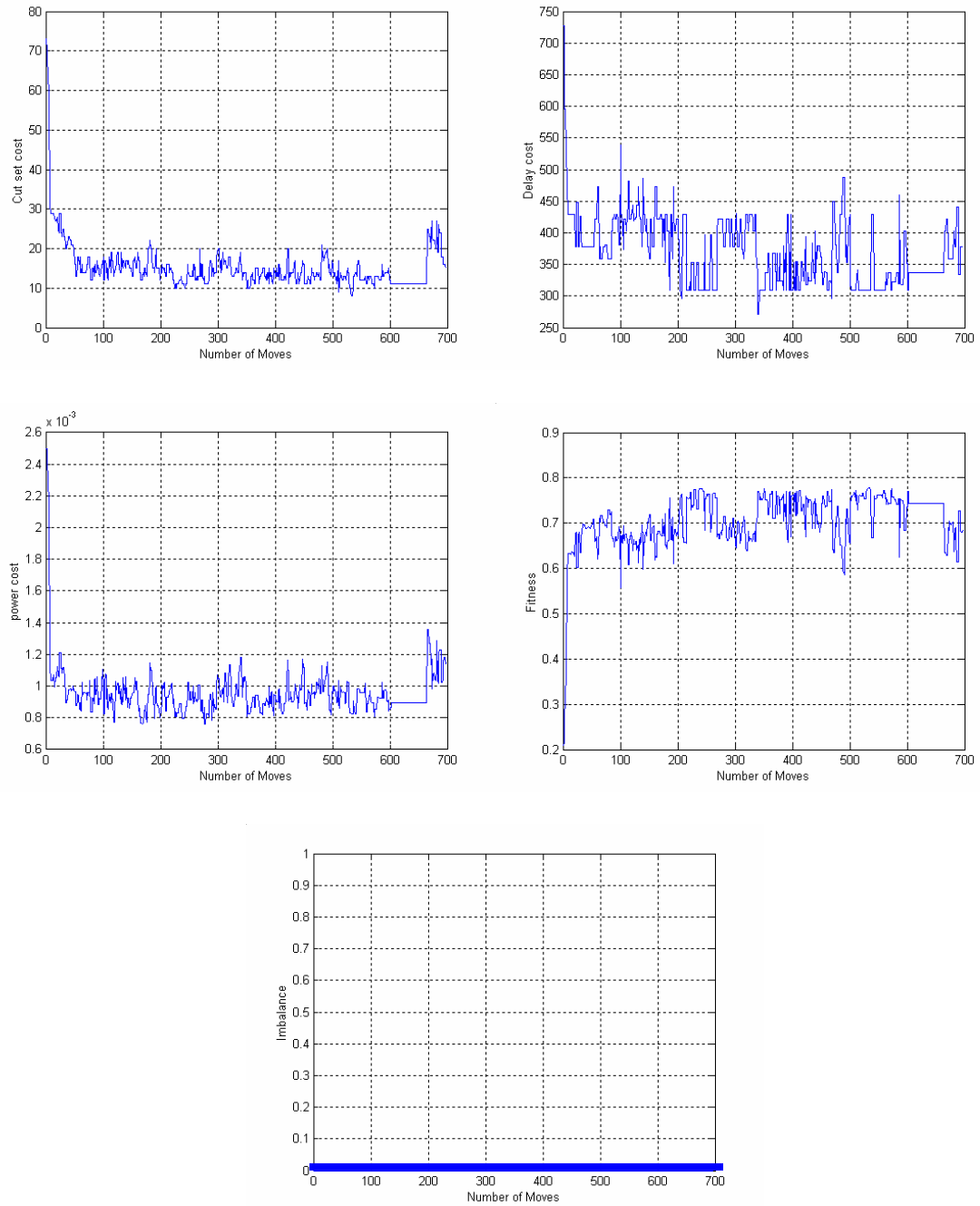


Figure 32: Cutset, delay, power, fitness, and imbalance of s2081

Circuit	SOP Cutset (nets)	MOP Cutset (nets)	Balance	Power S.P.	Delay ps	Fitness
s298	16	26	0%	1311	219	0.8
s386	32	32	0%	2145	423	0.6
s641	44	48	0.2%	3155	1117	0.8
s832	36	36	0%	3884	452	0.65
s953	83	83	0%	3973	485	0.7
s2081	8	8	0%	824	308	0.8

Table 9: Results summary for perfect partition balance (0%) for SOP and MOP.

References

- [1] Wehner, R..*Selbstorganisation im Superorganismus. Kollektive Intelligenz sozialer Insekten*. NZZ, Forschung und Technik, 14. January 1998, 61.
- [2] Rolf Pfeifer. Artificial Life “Lecture Notes”, Institute for Informatic at the University of Zurich. March 2003.
- [3] Marco Dorigo et al: Ant colonies for the traveling salesman problem, TR/IRIDIA/*University Libre de Bruxelles* Belgium. March 1996
- [4] Peter Korosec, Jurij Silc: Mesh Partitioning: A Multilevel Ant-Colony-Optimization Algorithm, Computer Systems Department, Jozef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia.
- [5] Scott Hauck, Gaetano Borriello. An Evaluation of Bipartitioning Techniques Department of EECS Department of CSE Northwestern University, University of Washington *IEEE Trans. on CAD*, Vol. 16, No. 8, pp. 849-866, August 1997.

- [6] A. E. Langham, P. W. Grant. Using Competing Ant Colonies to Solve k-way Partitioning Problems with Foraging and Raiding Strategies Department of Computer Science University of Wales Swansea Singleton Park Swansea, SA2 8PP, U.K
- [7] Sadiq M. Sait and Habib Youssef. VLSI Physical Design Automation: Theory and Practice. World Scientific Press, 2001.
- [8] Sadiq M. Sait and Habib Youssef. Iterative Computer Algorithms and their Application to Engineering. IEEE Computer Society Press, December 1999.
- [9] Raslan Hashim Al- Abaji. Evolutionary Techniques for Multi-Objective VLSI Netlist Partitioning. Master of Science Thesis, COE Department KFUPM, Dhahran. August 2002.
- [10] Massoud Pedram. CAD for Low Power: Status and Promising Directions. IEEE International Symposium on VLSI Technology, Systems and Applications, pages 331-336, 1995.
- [11] M. R. Garey and D. S. Johnson. Computers and Intractability. Freeman, San Francisco CA, 1979.

- [12] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 29(2):291-307, 1970.
- [13] C. M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. *Proc. of the 19th IEEE Design Automation Conference*, pages 175-181, 1982.
- [14] H. Vaishnav and M. Pedram. Delay Optimal Partitioning Targeting Low Power VLSI Circuits. *IEEE Trans. on Computer Aided Design*, 18(6):298-301, June 1999.
- [15] M. Holzrichter and S. Oliveira. *New Graph Partitioning Algorithms*. The University of Iowa TR-120., 1998.
- [16] L. Hagen and A. Kahng. Combining Problem Reduction and Adaptive Multistart: A new technique for Superior Iterative Partitioning. *IEEE Trans. On CAD*, 16(7):709-717, 1997.
- [17] S. Areibi and A. Vannelli. A Combined Eigenvector Tabu Search Approach For Circuit Partitioning. *Proc. of the 1993 Custom Integrated Circuits Conference (San Diego)*, pages 9.7.1 - 9.7.4., 1993.

- [18] Slawomir Koziel and Wladyslaw Szczesniak. Evolutionary Algorithm for Electronic Systems Partitioning and its Applications in VLSI Design. *IEEE Computing*, pages 1411-1414, 1999.
- [19] Shantanu Dutt and Wenyong Deng. A Probabilistic Approach to VLSI Circuit Partitioning. *Proc. Design Automation*, pages 100-105, June 1996.
- [20] Charles J. Alpert and So-Zen Yao. Spectral Partitioning: The More Eigenvectors, the Better. *Design Automation Conference*, pages 195-200, 1995.
- [21] L. Hagen and A. Kahng. New Spectral Methods for Ratio Cut Partitioning and Clustering. *IEEE Trans. CAD*, 11(9):1074-1085, September 1992.
- [22] S. Barnard and H. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101-117, 1994.
- [23] S. Areibi and A. Vannelli. Advanced Search Techniques for Circuit Partitioning, in *Quadratic Assignment and Related Problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science edited by P. Pardalos and H. Wolkowicz. 16:77-99, 1994.

- [24] H. Shin and C. Kim. A Simple Yet Effective Technique for Partitioning. IEEE Transaction on VLSI, pages 380-386, September 1993.
- [25] L. Hagen and A. Kahng. Combining Problem Reduction and Adaptive Multistart: A New Technique for Superior Iterative Partitioning. IEEE Trans. On CAD, 16(7):709-717., 1997.
- [26] L. A. Sanchis. Multiple-Way Network Partitioning. IEEE Trans. on Computers, IEEE Computer Society, Washington D.C., 38(1):62-81, 1989.
- [27] Kirkpatrick, C.D. Gelatt and Vecchi M.P. Optimization by Simulated Annealing. Science, 220(4598):671-680, May 1983.
- [28] Fred Glover. Tabu Search Part I. ORSA Journal on Computing, 1(3):190-206, 1989.
- [29] C. J. Alpert and A. B. Kahng. A Hybrid Multilevel/Genetic Approach for Circuit Partitioning. Physical Design Workshop, pages 100-105, 1996.
- [30] E. Lawler, K. Levitt, and J. Turne. Module Clustering to Minimize Delay in Digital Networks. IEEE Trans. on Computer-Aided Design, 47-57, 1969.

- [31] J. De Gruijter and A. B. McBratney. A Modified Fuzzy K-Means Method for Predictive Classification. Proc. of the First Conference of the International Federation of Classification Societies (IFCS), 1988.
- [32] C. Ball, P. Kraus, and D. Mlynski. Fuzzy Partitioning Applied to VLSI Floorplanning and Placement. Proc. IEEE Intl. Symp. Circuits and Systems, pages 177-180, 1994.
- [33] D. S. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 32(3):241-254, 1967.
- [34] S. Dey, F. Brglez, and G. Kedem. Corolla Based Circuit Partitioning and Resynthesis. ACM/IEEE Design Automation Conf., pages 607-612, 1990.
- [35] S. Dey, F. Brglez, and G. Kedem. Partitioning Sequential Circuits for Logic Optimization. IEEE Intl. Conf. Computer Design, pages 70-76, 1990.
- [36] S. Hauck and G. Borriello. An Evaluation of Bipartitioning Techniques. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):849-866, August 1997.

- [37] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Ant Colony System for the Design of Combinational Logic Circuits. *Evolvable Systems: From Biology to Hardware, Edinburgh, Scotland*, pages 21–30, April Springer Verlag, 2000.
- [38] M. Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant Algorithms for Discrete Optimization. Technical Report Tech. Rep. IRIDIA/98-10, IRIDIA, Universite Libre de Bruxelles, Brussels, Belgium, 1998.
- [39] M. Dorigo and M. Maniezzo and A. Colomi. The Ant Systems: An Autocatalytic Optimizing Process. Revised 91-016, Dept. of Electronica, Milan Polytechnic, 1991.
- [40] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. Self-organized Shortcuts in the Argentine Ant. *Naturwissenschaften*, 76:579–581, 1989.
- [41] M. Dorigo and G. Di Caro. *New Ideas in Optimisation*. McGraw Hill, London, UK, 1999.
- [42] M. Dorigo and T. Stutzle. The ant colony optimization metaheuristic: Algorithms, applications, and advances, 2002.

[43] Bambang Ali Basyah Sarif. Modified Ant Colony Algorithm for Combinational Logic Circuits Design. Master of Science Thesis, COE Department KFUPM, Dhahran. November 2003.

Vitae

Emran A. Ba-Abbad

Born in Shagra, Saudi Arabia.

Received Bachelor of Science in Electrical Engineering

Valparaiso University, Valparaiso, IN, U.S.A.

Joined Computer Engineering Department, KFUPM, in 1989.

Received Master of Science degree in Computer Engineering from KFUPM, Dhahran,
Saudi Arabia in 2005.