Ant Colony Optimization on Runtime Reconfigurable Architectures

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

von der Fakultät für Wirtschaftswissenschaften
der Universität Fridericiana zu Karlsruhe

genehmigte

DISSERTATION

von

Dipl.-Wi.-Ing. Bernd Scheuermann

Tag der mündlichen Prüfung:	20.12.2005
Referent:	Prof. Dr. Hartmut Schmeck
Korreferenten:	Prof. Dr. Georg Bol
	Assoc. Prof. Dr. Hossam ElGindy

2005 Karlsruhe

Acknowledgments

This thesis is the result of my research associated with the project "Optimization on Reconfigurable Architectures" funded by the German Research Foundation (DFG). During my doctoral studies I was employed at the Institute AIFB at the University of Karlsruhe. Three temporal positions as research fellow at the University of New South Wales (Sydney, Australia) were supported by the International Office (IB/DLR) of the German Ministry of Education and Research (BMBF) within the scope of the WTZ-project AUS 00/002.

I wish to express my gratitude to all those people, who supported my research work and the process of writing this thesis. First of all, I would like to thank my advisor, Prof. Dr. Hartmut Schmeck, for his patience and generosity, but also for the time and freedom he gave me to pursue my research interests. Many thanks also to Prof. Dr. Georg Bol and Assoc. Prof. Dr. Hossam ElGindy for reviewing my thesis and for offering important suggestions for enhancements. Moreover, I wish to thank the other members of the examination committee, Prof. Dr. Klaus Neumann and Prof. Dr. Rudi Studer.

It gave me great pleasure to work together with Prof. Dr. Martin Middendorf, who helped me solving problems at any time. I very much appreciated cooperating with Dr. Oliver Diessel, Dr. Michael Guntsch, Stefan Janson, and Keith So, who were the co-authors of many of my conference and journal articles. Their competent advice and many insightful discussions inspired and guided my work.

I am also indebted to my wife, Iris, who notably leveraged the quality of this thesis by proof-reading the entire text. Proof-reading parts of the thesis, Peter Bungert, Dr. Oliver Diessel, Ingo Pänke, and Keith So contributed further essential improvements.

I very much enjoyed working for the Institute AIFB together with my friendly colleagues in a wonderful atmosphere that inspired my work and brightened up my busy days. In particular I would like to thank Matthias Bonn, Manfred Gehann, and Frederic Toussaint for providing resilient tools and resources for extensive computational experiments. Last but certainly not least I would like to express my sincere gratitude to my family for their encouragement, patience, and understanding.

Karlsruhe, December 2005 Bernd Scheuermann

Contents

1 Introduction **1**

2 Optimizing with Ants **7**
2.1 Social Insects . . . 7
2.1.1 Characteristics . . . 8
2.1.2 Foraging Behavior of Ants . . . 9
2.1.3 Biological Experiments with Ants . . . 10
2.2 Modeling Ant Behavior . . . 12
2.3 Ant Colony Optimization . . . 15
2.3.1 Combinatorial Optimization Problems . . . 15
2.3.2 ACO Meta-Heuristic . . . 16
2.4 Applications . . . 19
2.4.1 Overview . . . 19
2.4.2 Examples . . . 21
2.5 Variants of ACO . . . 26
2.5.1 Sequential ACO Algorithms . . . 26
2.5.2 Parallel ACO Algorithms . . . 28
2.6 Analytical Results . . . 30

3 Reconfigurable Computing **31**
3.1 Introduction . . . 31
3.2 Architectures . . . 33
3.2.1 Classification . . . 33
3.2.2 Field Programmable Gate Arrays . . . 35
3.2.3 Host Coupling . . . 36
3.2.4 Granularity . . . 38
3.3 Abstract Models . . . 39
3.3.1 Overview . . . 39
3.3.2 Reconfigurable Mesh . . . 41
3.4 Runtime Reconfiguration (RTR) . . . 42
3.4.1 Reconfiguration Models . . . 43

3.4.2 Partial Evaluation . . . 44
3.4.3 Accelerating Configuration Speed . . . 45
3.5 Applications . . . 46
3.6 Trends and Emerging Directions . . . 50

4 ACO on Reconfigurable Architectures 51
4.1 Objectives and Restrictions . . . 52
4.2 Related Work . . . 53
4.3 Counter-based ACO (C-ACO) . . . 56
4.3.1 Algorithm . . . 56
4.3.2 Comparison of Standard ACO with C-ACO . . . 58
4.3.3 Heuristic Extensions . . . 65
4.3.4 Parallel Execution . . . 68
4.3.5 Comparison Modes . . . 76
4.3.6 Decision Sequencing . . . 85
4.3.7 Matrix Encoding . . . 88
4.3.8 Mapping C-ACO onto FPGA . . . 92
4.4 Population-based ACO (P-ACO) . . . 93
4.4.1 Introduction . . . 93
4.4.2 Modifications . . . 95
4.4.3 Implementation of P-ACO on FPGA . . . 97
4.4.4 Experimental Results . . . 107
4.4.5 Heuristic Extension . . . 114
4.5 Comparison of Different Approaches . . . 125
4.5.1 Algorithmic Properties . . . 125
4.5.2 Asymptotic Runtime and Resource Requirements . . . 126
4.6 Summary . . . 128

5 Concepts of Using RTR 131
5.1 Partitioning and Scheduling . . . 132
5.1.1 Counter-based ACO . . . 132
5.1.2 Population-based ACO . . . 134
5.2 Dynamic Changes of the Optimization Problem . . . 137
5.2.1 Dynamic Changes of the Problem Size . . . 137
5.2.2 Dynamic Changes of Problem Parameters . . . 139
5.3 Accelerating Convergence and Runtime . . . 141
5.4 Summary . . . 142

6 Conclusion 143
6.1 Review of Summaries and Results . . . 143
6.2 Directions for Further Study . . . 145

Bibliography **147**

Index **200**

Chapter 1

Introduction

Natural evolution has yielded biological systems in which complex collective behavior emerges from the local interaction of simple components. One example where this phenomenon can be observed is the foraging behavior of ant colonies [Deneubourg et al., 1983, Beckers et al., 1992]. Ant colonies are capable of finding shortest paths between their nest and food sources. This complex behavior of the colony is possible because the ants communicate indirectly by disposing traces of pheromone as they walk along a chosen path. Following ants most likely prefer those paths possessing the strongest pheromone information, thereby refreshing or further increasing the respective amounts of pheromone. Since ants need less time to traverse short paths, pheromone traces on these paths are increased very frequently. On the other hand, pheromone information is permanently reduced by evaporation, which diminishes the influence of formerly chosen unfavorable paths. This combination focuses the search process on short, profitable paths.

Inspired by this biological paradigm, the Ant Colony Optimization (ACO) meta-heuristic is introduced in [Dorigo et al., 1991b, Dorigo, 1992]. In ACO, a set of artificial ants searches for good solutions to the optimization problem under consideration. Each ant constructs a solution by making a sequence of local decisions guided by pheromone information and some additional heuristic information (if applicable). After a number of ants have constructed solutions, the best ants are allowed to update the pheromone information along their paths through the decision graph. Evaporation is accomplished by globally reducing the pheromone information by a certain percentage. This process is repeated iteratively until a stopping criterion is met. ACO shows a good performance on several combinatorial optimization problems, including scheduling [Merkle and Middendorf, 2001b, Merkle et al., 2002], vehicle routing [Gambardella et al., 1999a], constraint satisfaction [Solnon, 2002], and the Quadratic Assignment Problem [Gambardella et al., 1999b].

Usually, ACO algorithms are implemented in software on sequential machines.

However, if short computation times become essential, there exist mainly two options to speed-up the execution. One option is to develop parallel variants of the algorithm to be executed on multi-processor machines (see [Randall and Lewis, 2002] for an overview). The other very promising approach, as proposed in this thesis, is to directly map the ACO algorithm into hardware, thereby exploiting the parallelism and pipelining capabilities of the target architecture. Since the artificial ants construct their solutions independently, and as the core of the algorithm consists of iteratively repeated instructions, ACO is very attractive for an implementation in hardware.

Reconfigurable architectures are considered as the implementation platform, in particular Field Programmable Gate Arrays (FPGAs). Typically, FPGAs consist of an array of configurable logic blocks communicating via a network of programmable interconnect. After the FPGA has been programmed by the user, the device is ready to receive and process input data. Commonly, FPGAs can be re-programmed by the user, which allows to reuse the chip to test different variants of hardware-implemented ACO algorithms. FPGAs facilitate the development of digital systems and also allow for easy and quick design changes and verification. The algorithmic tasks within an ACO algorithm demand frequent memory accesses of different bandwidths, a large number of arithmetic and logic operations as well as a mixture of local and global communication. All these requirements together with low circuit development costs can only be addressed by a highly flexible and reusable target architectures like FPGAs. Furthermore, runtime reconfigurable FPGA devices allow to react to dynamic changes of the optimization problem by appropriately adapting portions of the implemented ACO circuit. FPGAs are established in a wide range of applications, e.g., audio processing [Melnikoff et al., 2002], video processing [Lehtoranta et al., 2005], network communication [Wolkotte et al., 2005], and cryptography [Nibouche et al., 2004]. Other publications illustrate that FPGAs are suitable as implementation platform for machine learning and meta-heuristic algorithms including Neural Networks [Thoma et al., 2003], Evolutionary Algorithms [Bland and Megson, 1998b], and Simulated Annealing [Abramson et al., 1998a].

A concept for an implementation of the ACO algorithm on reconfigurable architectures is presented in [Merkle and Middendorf, 2001b, 2002a, Janson et al., 2002, 2003]. The proposed algorithm targets the Reconfigurable Mesh (RMesh), a standard model for reconfigurable processor arrays, in which the processors are connected via a dynamically reconfigurable bus system [Miller et al., 1993, Jang et al., 1994]. This abstract model efficiently supports algorithmic tasks that are typical of ACO algorithms such as bit-summation and finding the rank of a number in a set. In this thesis, it is shown that ACO can also be implemented on commercially available FPGAs leading to significant speedups in runtime compared to the execution in software on general-purpose processors. Parts of this

thesis are based on previous work by the author [Diessel et al., 2002, Scheuermann et al., 2003, 2004a,b, Scheuermann and Middendorf, 2005, Scheuermann et al., 2005].

During hardware design one has to consider the constraints imposed by the available resources on chip. Various operations (e.g. multiplications, exponentiations) and data types (like floating-point numbers) that are typically required by ACO algorithms would demand a very large amount of chip resources and comparatively long computation times on fine-grained architectures like FPGAs. Therefore, it is an interesting topic to explore alternative variants of ACO, which better fit the architectural hardware constraints. Two alternative hardware-oriented ACO algorithms, Counter-based ACO (C-ACO) and Population-based ACO (P-ACO), are examined in this thesis.

As a new ACO variant C-ACO represents pheromone information by integer values instead of floating-point numbers. Furthermore, C-ACO applies a modified evaporation procedure: With every ant decision, an individual pheromone value is decremented by a constant amount (local evaporation), whereas the standard ACO algorithm would reduce all entries in the pheromone matrix by a certain percentage (global evaporation). Local evaporation is faster and requires less logic and routing resources on the FPGA. C-ACO allows to systolically pipe a sequence of artificial ants through a grid of processing cells, which promises a very efficient hardware realization. In experimental studies comparing C-ACO with the standard ACO algorithm, the proposed variant shows a competitive or even better optimization performance.

The P-ACO algorithm is examined as a further hardware-oriented ACO variant. Originally developed for the sequential execution in software and targeting dynamic optimization problems [Guntsch and Middendorf, 2002a], P-ACO also offers certain properties making it very attractive for a realization on FPGAs. Pheromone information is replaced with a small set (population) of good solutions discovered during the preceding iterations. Accordingly, the combination of pheromone updates and evaporation is substituted for the insertion of a new good solution into the population thereby replacing the oldest solution contained in the population. Experimental results indicate that P-ACO performs at least as well as the standard ACO approach [Guntsch and Middendorf, 2002b].

For both algorithms, C-ACO and P-ACO, a range of new algorithmic techniques is developed, which support the parallel, systolic solution construction and adapt the algorithm to the architectural requirements. These new techniques are tested in experimental environments considering various combinatorial optimization problems like the Traveling Salesperson Problem (TSP), the Quadratic Assignment Problem (QAP), and the Single Machine Total Tardiness Problem (SMTTP). The P-ACO algorithm is implemented on an FPGA, and it is shown that the hardware realization leads to a significant speedup compared to the soft-

ware counterpart on a workstation. Even though both algorithms are designed and examined with an FPGA implementation in mind, they may also be of interest as alternative ACO algorithms in software.

Common to both hardware-oriented variants is that specific operations typically executed in software on sequential machines are suitably modified for an accelerated execution on FPGAs. The attainable speedup originates from different data types, adapted algorithmic procedures, and certainly from a high degree of concurrency. Some reconfigurable architectures, however, offer the opportunity of further improvements: Runtime reconfigurable devices allow to dynamically change a portion of the implemented circuit while the rest continues operating. This capability can be exploited to speed-up the execution and to reduce space requirements as well as power consumption. With respect to the parallel ACO implementation, several concepts are presented demonstrating the usage of runtime reconfiguration. In the case of large ACO applications, runtime reconfiguration allows to implement the circuitry, which would otherwise not entirely fit onto the available resources. Furthermore, it is described how reconfiguration at runtime enables the hardware algorithm to efficiently adapt to dynamic changes of the optimization problem. Finally, runtime reconfiguration can be applied to accelerate the convergence and execution speed of the algorithm.

The remainder of this thesis is structured as follows:

- Chapter 2 introduces the principles of optimizing with ant-based algorithms by describing how the foraging behavior of real ants inspired the definition of artificial ants. Important biological experiments as well as descriptive mathematical models are mentioned, which created the fundamentals for the introduction of ACO as a generic framework for ant algorithms. An overview of applications and well-known variants of ACO is provided.

- An introduction to reconfigurable computing is given in Chapter 3. The characteristics of reconfigurable computing are described and distinguished from other conventional computing techniques. A classification of various reconfigurable architectures is provided, followed by an overview of reputed abstract models. FPGAs are described in greater detail as they represent the target platform chosen for the examinations in this thesis. Several models, benefits, and problems concerning runtime reconfiguration are discussed. An overview of various important applications and main trends are given as well as an outlook on emerging directions.

- Chapter 4 deals with the implementation of ACO algorithms on reconfigurable architectures. The objectives, opportunities as well as the challenges and restrictions are described, which are connected with this task. A brief overview of related work by other authors is provided. Afterwards, the

alternative hardware-oriented ACO algorithms, C-ACO and P-ACO, are described. For both approaches, various algorithmic modifications and new techniques are proposed and examined in experimental studies. Software simulations are conducted comparing C-ACO with standard ACO. P-ACO is actually implemented on an FPGA measuring the attainable speedup and the amount of resources required. Regarding P-ACO, a further main contribution consist in the Time-Scattered Heuristic as an alternative approach of supporting ant decisions by means of heuristic information. Thereafter, the properties of the proposed hardware-oriented variants are compared with the standard ACO algorithm and the ACO implementation on the RMesh.

- Chapter 5 discusses several concepts of applying runtime reconfiguration to the ACO implementation on FPGAs. Three different applications of runtime reconfiguration are identified covering temporal partitioning and scheduling of the circuit, dynamic changes to the optimization problem as well as accelerating convergence and execution speed.

- Finally, Chapter 6 concludes this thesis with a review of the results and a discussion of their significance followed by an outline of unresolved issues and directions for further study.

Chapter 2

Optimizing with Ants

The daily problems a colony of ants has to face include searching for food, assigning labor to individuals, constructing and extending the nest, feeding offspring, raising alarm, reacting to external events, defending the colony etc. Counterparts of these problems can be found in many fields of engineering, economics and computer science, e.g., the search for shortest tours, classification of customers, task allocation to mobile robots and many others. The focus of this chapter is put on the foraging behavior of ants, which is the inspiring source for the development of a new meta-heuristic called *ant algorithm* presented in [Dorigo et al., 1991b, Dorigo, 1992]. In ant algorithms, artificial ants search for good solutions to an instance of a given optimization problem. The following sections describe how the insights into real ants' behavior lead to the definition of artificial ants. First the characteristics of social insects are introduced in general. Afterwards the main emphasis is put on the foraging behavior of ants which are observed in biological experiments. These observations are described and analyzed in mathematical models creating the fundamentals for the introduction of *Ant Colony Optimization (ACO)* as a generic framework for ant algorithms. Applications and well-known variants of ACO are given as well as a brief overview of analytical examinations.

2.1 Social Insects

Such insects are said to be *social* that live in colonies, exhibit cooperative behavior and division of labor among distinct castes. Social insects which are further characterized by parental care of young, overlapping generations, and reproductive division of labor are called *eusocial* insects. These include all species of ants and termites as well as some species of bees, wasps, aphids and thrips. Common to all species of eusocial insects is that they can perform difficult tasks (like

ants finding shortest paths, termites constructing complex nests, or honey bees dividing combs into concentric regions) which far exceed the capabilities of a single insect. Furthermore, the colonies completely lack the existence of supervising individuals. Every single insect seems to pursue its own plan, however, the whole colony looks very well organized. How this phenomenon can be explained is outlined in the subsequent sections.

2.1.1 Characteristics

Many of the complex activities by social insects are based on the principles of *self-organization (SO)*. Studies on SO originate from the realm of physics and chemistry [Nicolis and Priogogine, 1977, Haken, 1983]. These theories describe the *emergence* of macroscopic patterns resulting from interactions on a microscopic level. The principles of SO and emergence can also be applied to social insects [Deneubourg et al., 1989]. Mainly based on local information they perform simple actions, but by the interplay of many individuals, they produce a rather complex global effect.

In order to perform their tasks, social insects need to communicate [Bonabeau et al., 1999]. *Direct* forms of communication include antennation, trophallaxis (exchange of liquid or food), visual or mandibular contact as well as chemical contact like the odor of surrounding nestmates. *Indirect communication* is based on the concept of *stigmergy*[1] as introduced by [Grassé, 1959, 1984]: Two insects communicate indirectly if one individual modifies its environment, these modifications are then perceived by the other individuals thereby possibly triggering further actions.

Stigmergy and SO enable insect colonies to react in a very flexible and robust way. *Flexibility* means that social insects are able to respond appropriately to perturbations of their environment. The insects react as if these environmental changes were modifications performed by the colony members themselves. *Robustness* allows the colony to remain functional even if some individuals fail to execute their tasks.

These characteristics make the principles of social insect very attractive to engineers and computer scientist, because modeling insect behavior would allow to build resilient decentralized systems consisting of simple cooperating *agents* in order to solve problems. Indirect interactions between these agents would also help reducing communication. So far only a few *swarm-intelligent systems* have been developed. Depending on the respective application, designing and programming such systems is very challenging, because often the problem solving actions of swarms are not predefined but emergent and strongly influenced by the

[1] stigmergy: Greek stigma = sting and ergon = work

interactions among individuals or between individuals and their environment. Building swarm-intelligent systems requires a very concise knowledge of the local interactions needed to produce the desired emergent behavior.

The term *swarm intelligence* is introduced in [Beni, 1988, Beni and Wang, 1989, 1991, Hackwood and Beni, 1991, Beni and Hackwood, 1992, Hackwood and Beni, 1992] within the scope of cellular robotic systems. A generalized definition of swarm intelligence is given in [Bonabeau et al., 1999] comprising all algorithms or distributed systems solving problems by methods, which are inspired by the collective behavior of social insects and other animal societies. Emergence and swarm intelligence are also central concepts, which are examined and applied in a new field of research called *Organic Computing* [Arbeitsgruppe Organic Computing, 2002, Hofmann et al., 2002]. Organic computers are systems which allow to dynamically adapt to changing environments and which fulfill the self-x requirements: self-organizing, self-reconfigurable, self-optimizing, self-healing, and self-protecting.

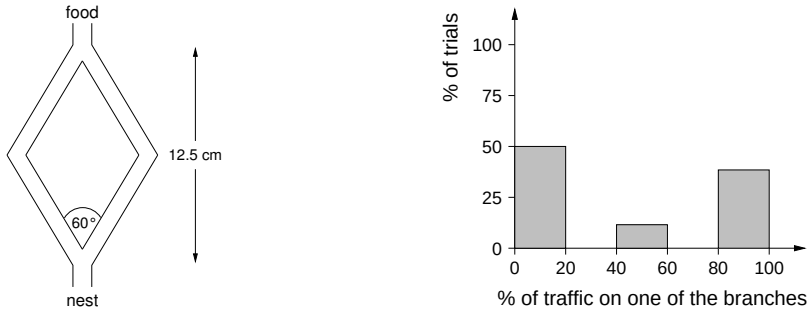
2.1.2 Foraging Behavior of Ants

When searching for food, some species of ants (e.g. Argentine ant Iridomyrmex humilis or Lasius niger) exhibit extraordinarily good capabilities in finding shortest paths, for instance, between the nest site and food sources. Despite their very limited visual perception (some species are completely blind) and restricted cognitive capacities, ants accomplish this task even in unknown or changing environments [Bonabeau et al., 1999, Dorigo and Stützle, 2004].

Finding shortest connections between two points is not the goal of a single ant, but rather the emergent collective result of cooperation in self-organized ant colonies. Ants mainly communicate via chemicals called *pheromones* as opposed to other higher species (including humans) whose communication is primarily based on vision and acoustics. There exist many different types of pheromones, however, with respect to the task of finding short paths, the most important one is the *trail pheromone*. In the sense of stigmergy, ants dispose this pheromone as they walk along a path from the nest to food sources. Ants can smell these pheromones guiding them to the food sources found by others. If an ant encounters multiple alternative pheromone trails, it is likely to choose the one with the highest pheromone concentration, thereby further reinforcing the respective concentration of pheromone. This amplification of earlier beneficial decisions is also known as *positive feedback*. As liquid substances pheromones underly the effect of *evaporation*, which gradually reduces the intensities of pheromone trails in the course of time. Hence, infrequently chosen paths become less appealing such that unprofitable former walks can be forgotten (*negative feedback*) [Bonabeau et al., 1999, Dorigo and Stützle, 2004].

2.1.3 Biological Experiments with Ants

To further investigate the trail-laying activities, several experiments with ants have been conducted in laboratory environments. The experimental setup applied by [Goss et al., 1989] and [Deneubourg et al., 1990] consists of a bridge connecting a nest of ants (Iridomyrmex humilis) and a food source. A bridge is made of two identical modules arranged in a line (double bridge), and each module has two branches, which can be of different lengths. The length of the longer branch is denoted by l_l, the length of the shorter one by l_s, and the ratio of both lengths is determined by $r = l_l/l_s$. Three experiments with different configurations of bridges are conducted. Every experiment is repeated with n trials and 11 different colonies of the Argentine ant Iridomyrmex humilis. At the beginning of every trial the bridge is free of pheromone and the ants leave their nest crossing the branches to search for food. Traffic is recorded in the interval between 30 and 40 minutes after placing the bridge.



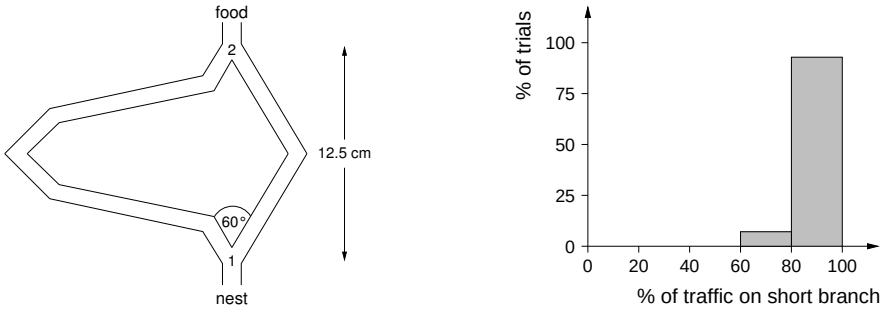
(a) Bridge module. (b) Histogram for experiment with $n = 26$ trials.

Figure 2.1: Setup and results obtained for bridge experiment with branches of equal length ($r = 1$) [Goss et al., 1989].

In the first experiment, a double bridge with branches of equal length (i.e. $r = 1$) is presented to the ants as visualized in Figure 2.1a. Initially, the ants randomly choose both branches with equal probability thereby disposing pheromones on their paths. Due to random fluctuations, at some stage, one of the branches contains a higher pheromone concentration than the other. This branch is thereafter selected with a higher probability causing a further amplification of the respective pheromone intensity. Eventually, almost all ants prefer the same branch (see Figure 2.1b).

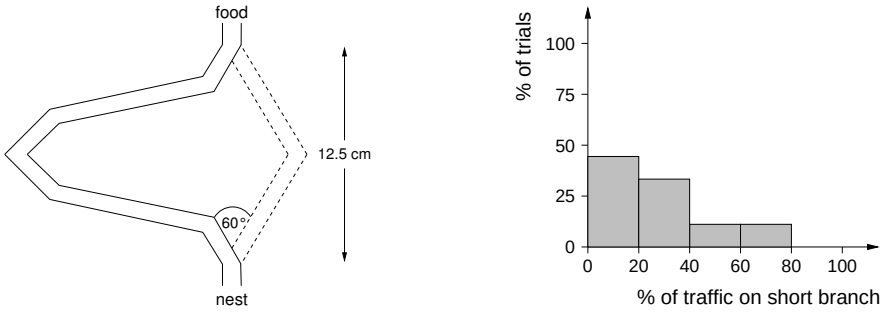
In the following experiment, one branch is twice as long as the other ($r = 2$). When the ants first reach bifurcation 1 (see Figure 2.2a), as before, the ants start to explore both branches with equal probability. However, the ants on the short branch reach the food source earlier, and therefore return earlier. When reaching

bifurcation 2 on their way back, they perceive a higher amount of pheromone on the shorter branch, which is then likely to be preferred. In contrast to the first experiment, almost in all trials, the ants converge to the shorter branch (see Figure 2.2b), although in some trials, a few ants still walk the longer one. This effect can be considered as a form of exploration of alternative paths.



(a) Bridge module. (b) Histogram for experiment with $n = 14$ trials.

Figure 2.2: Setup and results obtained for bridge experiment with branches of different length ($r = 2$) [Goss et al., 1989].



(a) Bridge module. (b) Histogram for experiment with $n = 18$ trials.

Figure 2.3: Setup and results obtained for bridge experiment with branches of different length ($r = 1$) and the short branch added after 30 minutes [Goss et al., 1989].

A final experiment is conducted to test the flexibility of the collective ant behavior. The bridge is the same as in the previous experiment except that at the beginning, the short branch is removed for 30 minutes to allow the ants to dispose an intense pheromone trail. Afterwards the shorter branch is added as shown in Figure 2.3a. Interestingly, only a few ants choose the shorter branch (see

Figure 2.3b). Obviously, the pheromone concentration on the longer branch is too strong to allow the colony to learn the new shorter connection. Furthermore, compared to the duration of a complete trial, evaporation is too slow to help the ants to forget earlier (now suboptimal) decisions. This experiment is repeated with the ant Lasius niger, which is capable of converging to the new shorter branch with high probability. This different behavior can be explained by the diverging characteristics of the two ant species, in particular the frequency of pheromone disposals or the capability to mark paths toward the food differently from paths leading back to the nest [Beckers et al., 1992].

2.2 Modeling Ant Behavior

Based on experimental studies with real ants, such as in the previous section, several attempts have been made to describe ants behavior in mathematical models. These models try to explain the mechanisms of the natural system, which is a prerequisite for the further design of decentralized, adaptive, flexible, and robust artificial systems capable of solving optimization problems. Two classes of models can be distinguished.

The first class comprises such biologically-oriented models describing the behavior of natural ants (refer to [Dorigo and Stützle, 2004] for an overview). Common to these models is that they do not integrate the effect of pheromone evaporation as they solely try to describe ant activities as observed in the bridge experiments. In these experiments evaporation does not play an important role, because the mean lifetime of the pheromone is very high compared to the time the ants needed to converge to the shortest path [Goss et al., 1989, Beckers et al., 1993].

The other class contains models devoted to the design of *artificial ants*. These models are stronger decoupled from the underlying metaphor in as far as the artificial ants can perform actions far beyond the capabilities of real ants. To this class of models belongs, e.g., the *Simple Ant Colony Optimization* (S-ACO) algorithm proposed by [Dorigo and Di Caro, 1999, Dorigo and Stützle, 2004]. S-ACO aims to apply the shortest path finding behavior of ants with the objective to tackle optimization problems. Hence, S-ACO represents the linkage between the biologically-oriented models and the generic framework of ACO introduced in Section 2.3. The artificial ants in S-ACO can be considered as *software agents* which autonomously move on a decision graph.

As a straightforward extension to the bridge experiments, S-ACO is applied to the related problem of finding the shortest path between a given source node and a given destination node in a graph $G = (N, E)$ with $n = |N|$ nodes, and edges $(i, j) \in E$ with equal length $c_{ij} = 1$. Obviously, the Shortest Path Problem can

be solved in polynomial time by other algorithms, e.g., [Dijkstra, 1959]. However, as a didactic example, this optimization problem is very well suited to explain the fundamentals of ACO.

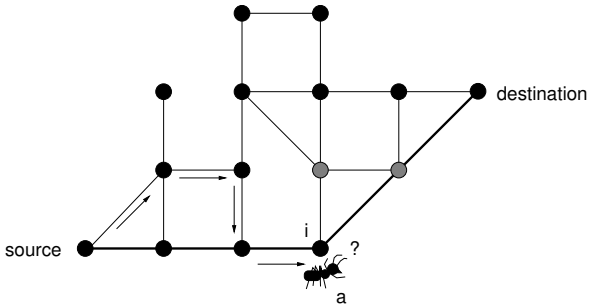


Figure 2.4: Example: Graph for the shortest path problem. Current position i of an ant a and its path (indicated by arrows) are given. Grey nodes represent the present neighborhood $\mathcal{N}_i^a$. Shortest path drawn with bold lines.

In S-ACO, m artificial ants start from a source node searching for short paths to the destination node, and they are guided by *artificial pheromone* τ_{ij} preceding ants have disposed on the edges. Pheromones on all edges are initialized by the same constant value (e.g. $\tau_{ij} = 1$). In detail, each ant performs the following sequence of actions:

1. Stepping from node to node, the ant moves forward from the source toward the destination, thereby constructing a solution (path). When located on node i, ant a senses the pheromones on all edges leading to the nodes in the *neighborhood* $\mathcal{N}_i^a$, which contains all nodes directly connected to node i excluding the immediate predecessor on the path being currently constructed (see Figure 2.4). If the neighborhood is an empty set (dead end in the graph), then the immediate predecessor is included. The next node $j \in \mathcal{N}_i^a$ to be visited is determined probabilistically according to the following distribution:

$$p_{ij}^a = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in \mathcal{N}_i^a} \tau_{il}^\alpha} & : \quad j \in \mathcal{N}_i^a \\ 0 & : \quad \text{else.} \end{cases} \tag{2.1}$$

   Weight α (external parameter) describes the degree of nonlinearity. The ant memorizes all decisions and appends the selected node to the sequence of all nodes visited so far. Note that this procedure does not prevent the construction of loops. While moving forward the ant does not manipulate the pheromones on the traversed edges.

2. After reaching the destination, the ant scans the constructed solution and removes loops from the path.

3. Ant a evaluates the solution constructed in order to determine the respective path length L^a.

4. Starting from the destination node, the ant returns to the source by deterministically following its loop-free path. When traversing an edge (i, j), the corresponding pheromone value is updated according to

$$\tau_{ij} := \tau_{ij} + \Delta^a, \tag{2.2}$$

where Δ^a denotes the amount of update added by ant a. Two variants of *pheromone update* are implemented in S-ACO: The update value either depends on the length L^a of the path ($\Delta^a = 1/L^a$), or it is chosen to be the same constant value for all ants. In the latter case, the emergent optimization behavior only relies on the *differential path length*, i.e. ants on short paths can deposit pheromone earlier than ants on long paths.

Additionally to all the individual actions of the artificial ants, pheromones are evaporated after all ants have moved to the next node. Evaporation is accomplished by reducing all pheromone values by the same percentage, which is called *evaporation rate* $\rho \in [0, 1)$:

$$\forall (i, j) \in E : \tau_{ij} := (1 - \rho) \cdot \tau_{ij}. \tag{2.3}$$

Inspired by natural evaporation, pheromone values are reduced to allow the colony to forget earlier unfavorable decisions. By appropriately tuning parameter ρ, the *exploration* of so far unvisited paths is supported, and the likelihood of an early convergence to a local optimum is decreased. Furthermore, evaporation is important to restrict the range of pheromone values.

The results retrieved from the experimental studies with S-ACO can be summarized as follows [Dorigo and Di Caro, 1999, Dorigo and Stützle, 2004]:

- To converge to the shortest path, the algorithm requires a higher number of ants m (resulting in longer simulation runs), when applying the differential path length strategy. Including the path length into update values leads to better results compared to constant update values. This effect becomes even more apparent with increasing problem complexity.

- Evaporation is essential to help the algorithm finding the shortest path. With increasing complexity of the problem, a careful tuning of parameter ρ becomes crucial. Too small evaporation rates may lead to sub-optimal solutions, whereas very high values of ρ can unnecessarily retard convergence.

- Choosing parameter $\alpha = 1$ results in the best optimization performance. Higher values of α amplify initial random fluctuations of pheromone values, which mislead the ongoing search process.

2.3 Ant Colony Optimization

Based on the behavior of real ant colonies, many variants of ant algorithms have been proposed to search for good solutions to numerous $\mathcal{NP}$-hard combinatorial optimization problems. A large fraction of these variants is unified in a common framework, which is called the Ant Colony Optimization (ACO) meta-heuristic.

2.3.1 Combinatorial Optimization Problems

Solutions to combinatorial optimization problems are described by a composition of integer variables which are limited to a finite range. Combinatorial optimization problems include, for instance, routing problems, assignment problems, and scheduling problems. A combinatorial optimization problem is defined by a triple $\Pi = (\mathcal{S}, f, \Omega)$ with $\mathcal{S}$ denoting the set of candidate solutions, f representing the objective function, and Ω a set of constraints. The objective function determines a function value $f(s)$ for every candidate solution $s \in \mathcal{S}$. Such solutions $\tilde{s} \in \tilde{\mathcal{S}}$ with $\tilde{\mathcal{S}} \subseteq \mathcal{S}$ which satisfy the set of constraints Ω are called feasible solutions. Function f is either to be minimized or maximized, i.e. the objective is to find an optimal solution $s^* \in \tilde{\mathcal{S}}$ such that $\forall s \in \tilde{\mathcal{S}} : f(s^*) \leq f(s)$ in the case of a minimization problem, or $\forall s \in \tilde{\mathcal{S}} : f(s^*) \geq f(s)$ for a maximization problem.

For some combinatorial optimization problems, e.g., the Shortest Path Problem [Dijkstra, 1959] or the Chinese Postman Problem [Kwan, 1962], exact algorithms are known, that allow to find the optimal solution in a time which is polynomially dependent on the size of the problem instance. More challenging are $\mathcal{NP}$-hard problems (see, e.g., [Garey and Johnson, 1983] for a concise introduction to the intractability of optimization problems), for which it is yet unknown if there exists an algorithm that is capable of finding the optimum in polynomial time. Such problems include, e.g., the Traveling Salesperson Problem (TSP) [Lawler et al., 1985] or the Quadratic Assignment Problem (QAP) [Cela, 1998]. Many years of research in the field of complexity theory suggest that solving $\mathcal{NP}$-hard problems to optimality always requires exponential runtime, although it has never been proved. Considering $\mathcal{NP}$-hard combinatorial optimization problems, exact algorithms guarantee to find the optimum, but in the worst case the search requires exponential time. Such exact algorithm include, e.g., Complete Enumeration, Branch-and-Bound, Branch-and-Cut [Neumann and Morlock, 2002]. On the other hand, many algorithms have been developed which afford only polyno-

mial time. These approximate algorithms, also called *heuristics*, search for good solutions to the optimization problem, however, they cannot guarantee to find the optimum. Heuristics can be further sub-divided into constructive or local search methods. Constructive methods start from an initially empty solution and build up a complete solution by successively adding further components, e.g., in TSP, a solution is generated by repeatedly selecting an as yet unvisited city until the tour is complete. Local search methods start from a complete solution which is subject to local changes that iteratively try to improve the solution quality. The choice of the next local change can be guided by hill-climbing or gradient-descent strategies.

Many heuristics are problem-dependent, i.e. they exploit problem-specific knowledge and can therefore often provide good solutions in reasonably short time, although these heuristics are often very specialized for one sort of problem and can only hardly if ever be applied to others. In this context so-called *meta-heuristics* play an important role, since they provide a generic framework for the creation of problem-specific heuristics. For some applications, the techniques of meta-heuristics offer the only way for an efficient optimization, when other heuristics cannot be properly adapted. Some examples of popular meta-heuristics include Evolutionary Algorithms [Fogel et al., 1966, Holland, 1975, Rechenberg, 1973, Schwefel, 1981, Goldberg, 1989, Michalewicz, 1999], Simulated Annealing [Cerný, 1985, Kirkpatrick et al., 1983], Iterated Local Search [Lourenço et al., 2002], Tabu Search [Glover, 1989, 1990, Glover and Laguna, 1997], and Ant Colony Optimization, whereof the latter one is introduced in the following section.

2.3.2 ACO Meta-Heuristic

In [Dorigo and Di Caro, 1999, Dorigo et al., 1999, 1996, Dorigo and Stützle, 2002], the Ant Colony Optimization (ACO) meta-heuristic is introduced as a generic framework for many ant algorithms, and can be applied to a wide range of combinatorial optimization problems. An instantiation of the ACO meta-heuristic is called an *ACO algorithm*. An ACO algorithm is considered as a system of cooperating artificial ants constructing solutions by a sequence of stochastic local decisions based on pheromone values, which are subject to the processes of pheromone update and evaporation as feedback strategies. Consequently, every ACO algorithm is also an ant algorithm. However, not every ant algorithm complies the requirements of ACO algorithms. For instance [Gambardella et al., 1999b] present an ant algorithm, which iteratively modifies solutions (instead of constructing them) in the sense of local search and is therefore not considered as an ACO algorithm as defined above.

The ACO meta-heuristic, as listed in Algorithm 2.1, can be considered as a

generalization of the S-ACO algorithm described Section 2.2. In ACO, pheromones τ_{ij} are typically associated with the edges of the respective decision graph, although in some cases, e.g., subset problems [Leguizamón and Michalewicz, 1999], pheromones τ_i may also be disposed on the nodes. In the following, it is assumed that pheromones are always related to edges. When starting the algorithm, all pheromone values are assigned initial values $\tau_{ij} := \tau_0$.

Algorithm 2.1 ACO meta-heuristic.

```
1: initialize pheromone values
2: repeat
3:     schedule ant activities
4:         construct solution
5:         evaluate solution
6:         update pheromone values
7:         perform daemon actions
8:     end schedule
9: until stopping condition met
```

Algorithm 2.2 Procedure construct solution.

```
1: initialize s as empty solution
2: while s not complete do
3:     compute index i of next decision
4:     randomly select j ∈ N_i according to transition rule
5:     add j to solution s
6: end while
```

After initialization, the artificial ants iteratively search for good solutions to the optimization problem. In every iteration, the activities of m working ants have to be scheduled (line 3 in Algorithm 2.1). Scheduling ant activities is related to two different aspects: to the synchronization of the m ants, and to the sequence of individual ant activities. The synchronization of the ants can be completely parallel, or they work sequentially one after the other. The individual ant activities mainly consist of four procedures: construction and evaluation of solutions, pheromone update, and daemon actions. Depending on the respective instantiation of the meta-heuristic, the order in which these activities are scheduled may differ, or they can be interleaved with each other.

The ant activities performed during solution construction are described in Algorithm 2.2. An ant starts from a source node on the decision graph with an initially empty solution s. When located on a node i, ant a randomly decides to move to an adjacent node $j \in \mathcal{N}_i^a$ (line 4 in Algorithm 2.2). The selection

probability p_{ij}^a is a function of pheromone value τ_{ij} and an optional heuristic value η_{ij}. This heuristic value provides problem-dependent information to guide the ant to promising areas of the search space. This *heuristic information* can be *static*, i.e. the heuristic values do not change at runtime, which allows them to be computed offline in the initialization phase of the ACO algorithm. Heuristic information may also be *dynamic* if the heuristic values depend on the current state in the process of solution construction and therefore require an online computation. Usually, η_{ij} represents the cost, or an estimate of the cost, of adding j to the solution being currently constructed.

Probability p_{ij}^a is commonly expressed by the *random proportional transition rule*:

$$p_{ij}^a = \begin{cases} \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in \mathcal{N}_i^a} \tau_{il}^\alpha \cdot \eta_{il}^\beta} & : \quad j \in \mathcal{N}_i^a \\ 0 & : \quad \text{else,} \end{cases} \tag{2.4}$$

where α and β represent weights for the pheromone and the heuristic values, respectively. The most common choice is setting $\alpha = 1$ (cf. Section 2.2), but also values $\alpha > 1$ have been investigated to enforce a quicker convergence [Fenet and Solnon, 2003], but eventually, resulted in poorer solution qualities. Choosing parameter $\beta = 0$ eliminates the influence of heuristic information and leads to the selection probability used in S-ACO (cf. Equation 2.1). It has been shown that in many cases, the integration of heuristic information with $\beta > 1$ yields better results, e.g., $\beta = 2$ [Dorigo and Gambardella, 1997b, Stützle and Hoos, 2000] or $\beta = 5$ [Bullnheimer et al., 1999c]. In [Dorigo and Gambardella, 1997b], the *pseudo-random proportional transition rule* is presented, which allows to model the relative importance of *exploration* versus *exploitation*. With probability q_0 (external problem parameter) an ant selects the next node $j \in \mathcal{N}_i^a$, which offers the largest product of weighted pheromone and heuristic values:

$$j = \arg\max_{l \in \mathcal{N}_i^a} \tau_{il}^\alpha \cdot \eta_{il}^\beta. \tag{2.5}$$

This deterministic choice of the next node j is considered as the exploitation of previously learned knowledge to concentrate the search process on the neighborhood of the best-so-far solution. With probability $(1 - q_0)$ the ant performs a biased probabilistic decision according to Equation 2.4, which corresponds to an exploration of the search space.

The constructed solutions are evaluated to determine their solution qualities (line 5 in Algorithm 2.1). Depending on the optimization problem, this evaluation is done either online or offline. *Online evaluation* means that the current (partial) solution quality is calculated during solution construction, i.e. the ant immediately determines how much a decision contributes to solution quality. *Offline evaluation* processes the solution quality after solution construction has been

completed.

The solution quality may then be employed to determine the degree of pheromone update, which generally, consists of a combination of negative and positive feedback. Negative feedback is mostly realized in terms of evaporating pheromone values as presented in S-ACO (cf. Equation 2.3). This method is closest to the natural paradigm, although other approaches exist, where such pheromone values are reduced, which are associated to unfavorable or older solutions, e.g., [Maniezzo, 1999, Guntsch and Middendorf, 2002b]. Positive feedback increases pheromone trails by a value Δ^a if they have either been used by many ants or by ants which have produced good solutions. It is generally required that Δ^a is a non-increasing function of the solution quality in the case of a minimization problem, or vice versa. Depending on the respective instantiation, update activities can be interleaved with solution construction, or they are completely decoupled.

Daemon actions (line 7 in Algorithm 2.1) are centralized operations, which are beyond the capabilities of a single ant. Such actions include, e.g., the application of local optimizer or the comparison of solution qualities to determine the best ants, which are allowed to update pheromones. The algorithm is terminated as soon as a specific *stopping condition* is met, e.g., a predefined maximum number of iterations has been executed, or the best yet reached solution quality has not changed over a certain number of iterations.

Considering the sequential software implementation of the ACO meta-heuristic, every solution construction consists of a sequence of n ant decisions each of them requiring an asymptotic runtime of $O(n)$. The overall complexity therefore mainly depends on the total number of z solutions generated and on the complexity of the evaluation function and daemon actions. Assuming that the time for these operations does not exceed $O(n^2)$ per constructed solution, then the total runtime can be expressed as $O(z \cdot n^2)$.

2.4 Applications

The ACO meta-heuristic has been applied to a wide range of optimization problems, in many cases with great success. This section first gives an overview of the most common applications and subsequently describes some examples of ACO algorithms in greater detail.

2.4.1 Overview

The following overview of ACO applications represents an update and supplementation of a collection provided by [Dorigo and Stützle, 2004].

Routing	
Traveling salesperson	[Dorigo et al., 1991a,b, 1996, Dorigo, 1992] [Gambardella and Dorigo, 1995] [Dorigo and Gambardella, 1997a,b] [Stützle and Hoos, 1997, 2000, Bullnheimer et al., 1999c] [Cordón et al., 2000, Bianchi et al., 2002] [Eyckelhof and Snoek, 2002]
Vehicle routing	[Forsyth and Wren, 1997, Bullnheimer et al., 1999a,b] [Gambardella et al., 1999a, Wade and Salhi, 2001] [Reimann et al., 2002a,b, Ellabib et al., 2002, 2003] [Doerner et al., 2002, 2004, Chitty and Hernandez, 2004]
Capacitated arc routing	[Doerner et al., 2003]
Sequential ordering	[Gambardella and Dorigo, 1997, 2000]
Assignment	
Quadratic assignment	[Maniezzo et al., 1994, Dorigo and Gambardella, 1997b] [Stützle, 1997, Taillard and Gambardella, 1997] [Gambardella et al., 1999b, Maniezzo and Colorni, 1999] [Maniezzo, 1999, Stützle and Dorigo, 1999] [Stützle and Hoos, 2000, López-Ibáñez et al., 2004]
Graph coloring	[Costa and Hertz, 1997, Vesel and Zerovnik, 2000]
Graph matching	[Sammoud et al., 2005]
Generalized assignment	[Lourenço and Serra, 1998, 2002]
Frequency assignment	[Maniezzo and Carbonaro, 2000]
University course timetabling	[Socha et al., 2002, 2003]
Scheduling	
Job shop	[Colorni et al., 1994, Dorigo et al., 1996, Teich et al., 2001]
Open shop	[Pfahringer, 1996]
Flow shop	[Stützle, 1998a, Onwubolu, 2000]
Total tardiness	[Bauer et al., 2000]
Total weighted tardiness	[den Besten et al., 2000b, Merkle and Middendorf, 2000, 2003] [Gagné et al., 2002]
Project scheduling	[Merkle et al., 2000, 2002]
Power plant maintenance	[Foong et al., 2005]
Group shop	[Blum, 2002, 2003]
Subset	
Multiple knapsack	[Leguizamón and Michalewicz, 1999, Fidanova, 2002]
Max independent set	[Leguizamón and Michalewicz, 2000]
Redundancy allocation	[Liang and Smith, 1999]
Set covering	[Leguizamón and Michalewicz, 2000, Hadji et al., 2000] [Lessing et al., 2004]
Weight constrained graph tree partition	[Cordone and Maffioli, 2001]
Arc-weighted l-cardinality tree	[Blum and Blesa, 2003, Bui and Sundarraj, 2004]
Maximum clique	[Fenet and Solnon, 2003, Bui and Rizzo, 2004]
Machine learning	
Classification rules	[Parpinelli et al., 2002]
Bayesian networks	[de Campos et al., 2002]
Fuzzy systems	[Casillas et al., 2000]
Network routing	
Connection-oriented network routing	[Schoonderwoerd et al., 1996, 1997, White et al., 1998] [Di Caro and Dorigo, 1998b, Bonabeau et al., 1998] [Fujita et al., 2002]
Connectionless network routing	[Di Caro and Dorigo, 1997, 1998a,c, Subramanian et al., 1997] [Heusse et al., 1998, van der Put, 1998]
Optical network routing	[Navarro Varela and Sinclair, 1999]
Others	
Data warehouse, data mining	[Maniezzo et al., 2001a,b, Stubs et al., 2001]
Shortest common supersequence	[Michels and Middendorf, 1998, 1999]
Constraint satisfaction	[Solnon, 2000, 2002]
Protein folding	[Shmygelska et al., 2002, Angel et al., 2005]
Bin packing	[Levine and Ducatelle, 2004, Brugger et al., 2004]

Successful applications of ACO include various scheduling problems like Job Shop [Colorni et al., 1994], Flow Shop [Stützle, 1998a], Total Weighted Tardiness [den Besten et al., 2000b, Merkle and Middendorf, 2003], and the Resource-Constrained Project Scheduling Problem [Merkle et al., 2002]. The results on the Shortest Common Supersequence Problem [Michels and Middendorf, 1999], the Constraint Satisfaction Problem [Solnon, 2002], the Vehicle Routing Problem [Gambardella et al., 1999a], and the Quadratic Assignment Problem [Gambardella et al., 1999b] are also very promising. ACO algorithms have exhibited a good performance on dynamic optimization problems, e.g. the Dynamic TSP [Guntsch et al., 2000, Guntsch and Middendorf, 2000, Guntsch et al., 2001, Guntsch and Middendorf, 2002a] or the Probabilistic TSP [Branke and Guntsch, 2003], and Network Routing [Schoonderwoerd et al., 1996, Di Caro and Dorigo, 1998a].

2.4.2 Examples

The Traveling Salesperson Problem (TSP), the Quadratic Assignment Problem (QAP), and the Single Machine Total Tardiness Problem (SMTTP) are chosen as examples of ACO applications, since they are also applied with respect to the hardware implementation described in the subsequent chapters. The algorithms described below are standard approaches, which have shown a good performance on the respective problem classes. Note that for the described applications there exists a range of variations of the standard approach. The main differences are outlined in Section 2.5.

2.4.2.1 Traveling Salesperson Problem

Given a graph $G = (N, E)$ with $n = |N|$ nodes (cities), edges $(i, j) \in E$, and distances d_{ij} between cities i and j for $i, j \in \{0, \ldots, n-1\}$, the objective of the Traveling Salesperson Problem (TSP) is to find a distance minimal *Hamiltonian cycle*. A Hamiltonian cycle is a closed walk (tour) on the graph G, such that each city is visited exactly once. One can distinguish between symmetric and asymmetric TSP. In the symmetric case, the distances between cities i and j are irrespective of the direction of traversing the arcs connecting them, i.e. $d_{ij} = d_{ji}$ for every pair of cities. A TSP is called asymmetric if $d_{ij} \neq d_{ji}$ for at least one pair of cities.

The TSP is an $\mathcal{NP}$-hard optimization problem [Lawler et al., 1985], which has attracted a reasonable amount of research activities, e.g., [Reinelt, 1994, Johnson and McGeoch, 1995]. With regard to ACO, the TSP has always played an important role, because of its intuitive close relation to the shortest path finding behavior observed in ant colonies, and was therefore chosen as the application of the

first ACO algorithm, called Ant System (AS), introduced in [Dorigo et al., 1991a, Dorigo, 1992, Dorigo et al., 1996]. Many subsequent variants of ACO algorithms have also (initially) been developed for the TSP, e.g., Ant-Q by [Gambardella and Dorigo, 1995, Dorigo and Gambardella, 1996], the Ant Colony System by [Dorigo and Gambardella, 1997a,b], the Rank-based Ant System by [Bullnheimer et al., 1997, 1999c], and also the most successful approach, the MAX-MIN Ant System ($\mathcal{MM}$AS) in combination with local search by [Stützle and Hoos, 1996, Stützle, 1999, Stützle and Hoos, 2000]. Although ACO algorithms perform reasonably well on TSP, there exist other better heuristics like [Lin and Kernighan, 1973] or polynomial time approximation schemes for Euclidean TSP [Arora, 1998]. One motivation of using TSP as the ACO standard application is that this problem is easily understandable. Furthermore, ACO strategies can intuitively be applied as the objective of TSP is closely related to the natural paradigm of ants finding shortest paths. New algorithmic ideas, which have shown a good performance on TSP, could often successfully be adapted to other optimization problems, for which ACO algorithms belong to the best known approaches.

Algorithm 2.3 shows an application of ACO for the asymmetric TSP as proposed by [Dorigo, 1992]. Pheromone information is stored and maintained in an $n \times n$ pheromone matrix $[\tau_{ij}]$, where every entry τ_{ij} expresses how attractive it is for an ant located in city i to select j as the next city to be visited. This means that the pheromone matrix is encoded in an item×item fashion with an item corresponding to a city. *Item-item encoding* is employed whenever the relation between predecessor and successor plays an important role, e.g., TSP, the Sequential Ordering Problem (SOP), or scheduling problems with setup costs. At the beginning of the algorithm, all values in the pheromone matrix are set to an initial value $\tau_0 > 0$ (line 4). Note that the actual values of the diagonal elements are irrelevant due to the tour construction mechanism, which prevents an ant from returning to a previously visited city.

After initialization, the ants iteratively construct solutions (tours). In every iteration, m tours are constructed, one per ant. Starting from an initially empty tour, each ant successively selects cities until a tour is complete. To avoid the creation of loops, an ant a located in city i is only allowed to choose the next city from a neighborhood $\mathcal{N}_i^a$ called the *selection set* S, which contains all so far unchosen cities. At the beginning of each tour construction, S contains all city indices from the interval $\{0, \ldots, n-1\}$ (line 10). Whenever a city j is selected, index j is removed from S (line 16).

The first selection is made for start city c, which is determined either randomly or by a deterministic rule (line 11). All following selections in a row i are performed according to a specific transition rule (line 15). Selection probabilities are biased by the pheromone information stored in the i-th row of the pheromone matrix.

Algorithm 2.3 ACO algorithm for asymmetric TSP.

```
1: /* initialize pheromone matrix */
2: for i := 0 to n − 1 do
3:     for j := 0 to n − 1 do
4:         τ_ij := τ_0
5:     end for
6: end for
7: /* start iterations */
8: repeat
9:     for a := 0 to m − 1 do
10:        S := {0, ..., n − 1}                    /* initialize selection set */
11:        select start city c ∈ S
12:        S := S \ {c}; i := c
13:        for h := 0 to n − 1 do
14:            if h ≠ n − 1 then
15:                j := select(i)                  /* probabilistic selection */
16:                S := S \ {j}
17:            else
18:                j := c                          /* finally, return to start city */
19:            end if
20:            π^a(i) := j
21:            i := j                              /* move to next city */
22:        end for
23:    end for
24:    a* := arg min_{a∈{0,...,m−1}} L^a           /* determine best tour */
25:    π* := π^{a*}
26:    /* pheromone update */
27:    for i := 0 to n − 1 do
28:        for j := 0 to n − 1 do
29:            τ_ij := (1 − ρ) · τ_ij              /* evaporation */
30:        end for
31:        τ_{iπ*(i)} := τ_{iπ*(i)} + Δ            /* intensification */
32:    end for
33: until stopping condition met
```

Additionally, heuristic information may also be integrated. In TSP, the heuristic value of choosing to visit city j after the last chosen city i is commonly considered to be inversely proportional to the distance separating them: $\eta_{ij} = 1/d_{ij}$.

The selected city j is then added to the tour being currently under construction (line 20). A tour can be expressed by a permutation π of city indices $j \in \{0, \ldots, n-1\}$, whereby an entry $\pi(i) = j$ indicates that in tour π city j is visited immediately after city i. Afterwards the ant moves to city j, i.e. it steps to row number j of the pheromone matrix (line 21). By the end of each solution construction, the ant returns to the start city to complete its tour (line 18).

All m tours constructed are evaluated to calculate their tour lengths $L^a = \sum_{i=1}^{n} d_{i\pi^a(i)}$ and to determine the best tour π^* of the current iteration (lines 24 and 25). The pheromone matrix is then updated in two steps: First all pheromone values in the matrix are evaporated by a certain percentage ρ (line 29). Afterwards the pheromone values along the best tour π^* are increased by a fixed amount Δ (line 31). In the case of symmetric TSP, one has to ensure that all operations sustain the symmetry of the pheromone matrix. Hence, during pheromone update, pheromone value $\tau_{i\pi^*(i)}$ and also $\tau_{\pi^*(i)i}$ are increased by $\Delta/2$. The algorithm is terminated as soon as a certain stopping condition is fulfilled.

2.4.2.2 Quadratic Assignment Problem

Given n locations, n facilities, and two $n \times n$ matrices $[d_{ij}]$ and $[f_{hl}]$, with d_{ij} being the distance between locations i and j, and f_{hl} denoting the flow between facilities h and l, the goal of the Quadratic Assignment Problem (QAP) is to find an assignment of locations to facilities, i.e. a permutation π of $\{0, \ldots, n-1\}$, such that the sum of distance-weighted flows between facilities $F = \sum_{h=0}^{n-1} \sum_{l=0}^{n-1} d_{\pi(h)\pi(l)} f_{hl}$ is minimized.

QAP is known to be an $\mathcal{NP}$-hard optimization problem [Sahni and Gonzalez, 1976], which plays an important role in theory and in many practical applications. A great variety of ACO algorithms has been developed for the QAP including the Ant System (AS) [Maniezzo et al., 1994, Maniezzo and Colorni, 1999], ANTS [Maniezzo, 1999] and $\mathcal{MM}$AS [Stützle, 1997, Stützle and Hoos, 2000]. ACO belongs to the best-performing meta-heuristics known for the QAP. Other ant-based algorithms, though not ACO algorithms, for QAP comprise the Fast Ant System (FANT) [Taillard, 1998] and the Hybrid AS (HAS) [Gambardella et al., 1999b].

An application of ACO to the QAP is listed in Algorithm 2.4. The pheromone matrix is encoded in a *place-item* fashion, i.e. row indices i represent places (facilities) in the solution vector whereas column indices j stand for items (locations) to be assigned to places. Note that there also exist other implementations with swapped interpretations of places and items. First, all entries in the phe-

romone matrix are initialized by a value $\tau_0 > 0$. Afterwards m ants iteratively start constructing solutions by successively assigning locations to facilities. For every assignment, a location $j \in S$ is randomly selected according to a specific transition rule (line 7).

Algorithm 2.4 ACO algorithm for the QAP.

```
1:  initialize pheromone matrix
2:  /* start iterations */
3:  repeat
4:      for a := 0 to m − 1 do
5:          S := {0, ..., n − 1}                    /* initialize selection set */
6:          for i := 0 to n − 1 do
7:              j := select(i)                      /* probabilistic selection */
8:              S := S \ {j}
9:              π^a(i) := j
10:         end for
11:     end for
12:     a* := arg min_{a∈{0,...,m−1}} F^a           /* determine best tour */
13:     π* := π^{a*}
14:     update pheromone matrix
15: until stopping condition met
```

ACO applications to QAP often use no heuristic at all. Alternatively, heuristic guidance based on two types of potential values derived from the flow and distance matrices are proposed in [Dorigo et al., 1996]. *Flow potential* $\bar{f}_h = \sum_{l=0}^{n-1} f_{hl}$ determines the sum of flows associated with facility h. The higher the flow potential of a facility the more important this facility is in the system of flow exchange. *Distance potential* $\bar{d}_i = \sum_{j=0}^{n-1} d_{ij}$ indicates the relative position of location i with respect to all other locations. The lower the distance potential of a location the more centrally it is situated. The aim of the QAP heuristic is to support the assignment of locations, which have low distance potentials, to facilities with high flow potentials. Therefore, one possible approach is to sort the facilities in non-increasing order of their flow potentials such that facilities with high flow potentials are assigned earlier in the process of solution construction. The reciprocal values of the distance potentials are used as heuristic information $\eta_{ij} = 1/\bar{d}_j$, which is independent of row index i.

Constructed solutions are evaluated and the solution with the lowest objective function value F^{a^*} is used to update the pheromone matrix. This is done in the same way as in Algorithm 2.3. This process is iteratively repeated until some stopping condition is met.

2.4.2.3 Single Machine Total Tardiness Problem

For the Single Machine Total Tardiness Problem (SMTTP), n jobs are given to be processed on a single machine. Every job j is described by its due date d_j and processing time p_j. If C_j denotes the completion time of job j in a schedule then $L_j = C_j - d_j$ defines its lateness and $T_j = \max(0, L_j)$ its tardiness. The objective is to find a schedule minimizing the total tardiness of all jobs $\sum_{j=0}^{n-1} T_j$.

SMTTP has been shown to be an $\mathcal{NP}$-hard optimization problem [Du and Leung, 1990] even though from the theoretical point of view, SMTTP can be solved in pseudo-polynomial time [Lawler, 1977]. Powerful branch-and-bound algorithms are reported to solve SMTTP to optimality up to problem sizes of $n = 500$ [Szwarc et al., 2001]. An ACO application to SMTTP is presented in [Bauer et al., 2000]. Note that the related problem with weighted tardiness (SMTWTP) is $\mathcal{NP}$-hard in the strong sense [Lenstra et al., 1977]. Applications of ACO to SMTWTP are described in [den Besten et al., 2000a], and the ACO algorithm presented by [Merkle and Middendorf, 2003] belongs to the best-performing heuristics for this problem.

The ACO algorithm for SMTTP, as used in this thesis, is very similar to Algorithm 2.4, and is therefore not listed explicitly. As in the case of QAP, the pheromone matrix is also encoded in a place-item fashion, and solutions (schedules) are constructed by successively assigning items (jobs) to places in the solution vector. Heuristic information can be defined via priority rules. According to the *earliest due date rule* (EDD), heuristic information $\eta_{ij} = 1/d_j$ prefers jobs with low due dates. Next to due dates, the heuristic information $\eta_{ij} = \frac{1}{\max\{\mathcal{T}+p_j, d_j\}}$ used by the *modified due date rule* (MDD) also considers the individual processing time p_j of job $j \in S$ and the total processing time $\mathcal{T}$ of all as yet scheduled jobs.

2.5 Variants of ACO

Since the first publication of an ant algorithm [Dorigo et al., 1991a] many new algorithmic concepts have been developed and tested. These approaches mainly differ in the type of pheromone update, or in their mode of execution, either sequential or parallel. Some of the most interesting variants of ACO algorithms are presented in this section.

2.5.1 Sequential ACO Algorithms

The *Ant System* (AS), as presented by [Dorigo, 1992] for the TSP, calculates an update value Δ_{ij}^a for every ant a in the current iteration. If a pheromone value τ_{ij} is not related to a tour π^a constructed by ant a, then τ_{ij} does not receive

an update, i.e. $\Delta_{ij}^a = 0$. Otherwise the update value depends on the update strategy applied: The *Ant-density* strategy assigns a constant value $\Delta_{ij}^a = \Delta$; using *Ant-quantity*, the update value $\Delta_{ij}^a = 1/d_{ij}$ is inversely proportional to the distance between two cities; the update value $\Delta_{ij}^a = Q/L^a$ for *Ant-cycle* is inversely proportional to the tour length L^a with Q being a constant parameter. The actual update value $\Delta_{ij} = \sum_{a=0}^{m-1} \Delta_{ij}^a$ is the sum over the individual update values of all m ants in the iteration. Furthermore, an *elitist strategy* is introduced, which allows the ants that found the so far shortest tours to perform an additional pheromone update. In experimental studies, using a small number of such elitist ants produces better results.

In [Dorigo and Gambardella, 1995, Gambardella and Dorigo, 1995, Dorigo and Gambardella, 1996], the Ant System algorithm is extended by Q-learning [Watkins, 1989] techniques. Accordingly, this variant is called *Ant-Q*. The authors apply Ant-Q to the TSP and propose two types of pheromone update (reinforcement): The immediate reinforcement modifies pheromones directly after an ant decision, whereas using delayed reinforcement, the ants perform an update after a construction of a complete tour. The amount of pheromone update also depends on the length of the respective tour. The Ant-Q algorithm is extended by a so-called belief factor, which reflects the confidence an ant has in the perceived pheromone concentration [Monekosso et al., 2002]. During early phases of the optimization process, this factor makes an ant believe to a lesser degree in the pheromone information because the system is still in a explorative state.

Based on Ant-Q, the *Ant Colony System* (ACS) is introduced in [Gambardella and Dorigo, 1996, Dorigo and Gambardella, 1997a,b], which is also applied to the TSP. Each time a city is selected, a so-called local pheromone update reduces the concerned pheromone value by a certain percentage and performs a partial reset to the initial value: $\tau_{ij} := (1 - \rho)\tau_{ij} + \rho\tau_0$. A further global pheromone update is executed after the construction of a tour. However, in contrast to AS, not all ants are allowed to update, but only the best ant of the current iteration or the elitist ant (or both of them). In experimental studies, this version of global pheromone update yields very good results [Stützle and Hoos, 2000, Merkle et al., 2002].

Another modification of the Ant System, called *AS-rank*, is presented by [Bullnheimer et al., 1999c]. The k best ants of the current iteration, together with a certain number of elitist ants, are allowed to execute a global pheromone update. The amount of pheromone they are allowed to update is proportional to their individual rank of solution quality.

In [Stützle and Hoos, 1997, Stützle, 1999, Stützle and Hoos, 2000], the *MAX-MIN Ant System* is introduced and successfully applied to several optimization problems. The authors propose to restrict the pheromone values to an interval $[\tau_{min}, \tau_{max}]$ thereby bounding the selection probabilities in Equation 2.4 if the heuristic values are also limited. This strategy avoids an early convergence to

a local optimum, and in combination with local search and elitist ants, leads to very good results on TSP and QAP.

Other variants of update strategies have been presented, which do not include an explicit evaporation. In [Maniezzo, 1999], it is proposed to compare the solution quality of an ant with the average quality of the m preceding ants. If the solution quality is better, then the corresponding pheromone values receive a positive update, and a negative update otherwise. The substitution of evaporation by a combination of positive and negative update is also a key concept of the *Population-based ACO* (P-ACO) by [Guntsch and Middendorf, 2002b]. P-ACO keeps and maintains a population of the k best solutions from the k preceding iterations. A positive update is realized by an insertion of a new solution into the population, a negative update is accomplished by removing a solution from the population. From this population the corresponding pheromone matrix can be derived. A detailed description of P-ACO is provided in Section 4.4.

2.5.2 Parallel ACO Algorithms

So far the described ACO variants try to achieve an improved optimization performance mainly by a modification of the pheromone update techniques. Another intuitive way to improve the performance is to parallelize the execution of the ACO algorithm. For a survey of parallel implementations of ACO and various decomposition techniques the reader is referred to [Randall and Lewis, 2002].

A very fine-grained version of parallelization with only one single ant per processor is investigated by [Bolondi and Bondaza, 1993]. Due to a very high rate of communication, scaling the algorithm to a high number of processors deteriorates the optimization performance. Better results can be obtained by a coarse-grained variant implemented on a transputer cluster [Bolondi and Bondaza, 1993, Dorigo, 1993]. Both variants can be considered as direct parallelizations, which do not adapt the underlying ACO algorithm with respect to the parallel architecture.

In [Bullnheimer et al., 1998] several ant colonies are distributed over the processors of a parallel machine. These colonies exchange information after a predefined fixed number of iterations. In simulations, the authors only examine the reduction of runtime. However, it is not investigated how much impact this new kind of parallel ACO algorithm has on optimization performance.

In [Stützle, 1998b] it is examined, in which cases the execution of multiple short runs of an ACO algorithm (optionally started with different parameter settings) performs better than the execution of a long run of the algorithm requiring the same time as the short runs altogether. This work does not make use of some interesting features a parallelization can offer, like an early termination of unpromising algorithm runs.

The parallel ACO algorithm presented by [Talbi et al., 1999] is based on a

master-worker model. Every worker processor is assigned exactly one ant constructing one solution, which is then submitted to the master processor. The master collects all received solutions to calculate a new pheromone matrix, which in turn is sent to all workers. The algorithm is applied to the QAP, but the optimization performance is not compared with other parallel approaches. In [Rahoual et al., 2002], the same master-worker model is applied to the Set Covering Problem (SCP) [Balas, 1983], and the computation time is compared to a multi-start variant of multiple parallel ACO algorithms, which do not exchange any information.

In [Iredi et al., 2001] as well as in [Merkle and Middendorf, 2001a] a coarse-grained parallel variant of ACO implemented on a workstation cluster is presented. Multiple ant colonies work in parallel to find good solution to the bicriterion single machine total tardiness problem (SMTTP) with changeover costs. In continued studies [Middendorf et al., 2002], different strategies of cooperation between ant colonies with a variable frequency and quantity of exchange of locally good solutions are compared. It turns out that an infrequent exchange of small quantities of information positively affects optimization performance. Whereas most parallelizations of ACO are based on multiple processors cooperating by message passing, [Delisle et al., 2005] present an implementation with ants communicating via a shared memory.

Another fine-grained parallel ACO approach that is suitable for the Reconfigurable Mesh (RMesh) is proposed in [Merkle and Middendorf, 2001b, 2002a]. The RMesh is a standard model of a reconfigurable processor array, in which the processors are connected by a dynamically reconfigurable bus system. Refer to Section 3.3.2 for a detailed explanation of the RMesh model. The authors propose to map the pheromone matrix onto the array of reconfigurable processors. Solutions are generated by systolically piping the ants top-down through the mesh. Asymptotically, this variant leads to a significant asymptotic speedup with respect to the software implementation of standard ACO. An extended description of this approach is presented Section 4.2. Based upon this RMesh implementation of ACO, [Janson et al., 2002, 2003] describes a strategy to enforce the convergence of the algorithm by repeatedly deleting those rows and columns from the pheromone matrix, which are associated with pheromone values exceeding a certain threshold value.

Whereas the RMesh is an abstract computational model, an implementation of ACO on commercially available Field-Programmable Gate Arrays (FPGAs) is presented by the author of this thesis, cf. [Diessel et al., 2002, Scheuermann et al., 2003, 2004a,b, Scheuermann and Middendorf, 2005, Scheuermann et al., 2005]. An FPGA consists of an array of configurable logic blocks, which are linked by programmable interconnect. Users of FPGAs can configure and reconfigure the chip for various applications. The authors show that – with several

restrictions and modifications – ACO can be implemented on FPGAs, leading to a significant speedup in runtime compared to implementations in software on sequential machines. A detailed description of these approaches is given in Chapter 4. For deeper insights into FPGA technology refer to Section 3.2.2.

2.6 Analytical Results

Most publications in the field of ACO algorithms describe new algorithmic variants or applications. However, so far only a few works exist, which analyze ACO algorithms from the theoretical point of view. Possible reasons for the lack of theoretical examinations are that, in contrast to other meta-heuristics like Evolutionary Algorithms, the ACO field of research is rather young, and the interdependencies between individual ant decisions render a mathematical description very difficult. All previously presented proofs imply simplified versions of ACO, and do not give direct guidelines for real-world usage, nevertheless they are interesting due to the ascertainment of general properties of the algorithms considered. These algorithms include, for instance, the *Graph-based Ant System* (GBAS) [Gutjahr, 1998, 2000, 2002, 2003b], ACO for stochastic combinatorial optimization [Gutjahr, 2003a], and the MAX-MIN Ant System [Stützle and Dorigo, 2002], where a proof of convergence is given. As opposed to AS, these variants of ACO are based on the assumption that only such ants are allowed to update pheromones, which found a new solution that is at least as good as the so far best found solution. Given a problem instance with a single global optimum, it is shown that both algorithms can find the optimum with a probability, which is either equal 1 or can be made arbitrarily close to 1.

Other heuristics like the *Cross-Entropy-Method* [Rubinstein, 1999] or the *Hypercube Framework for ACO* [Blum et al., 2001] are proposed, which show a close relation to ACO algorithms and also allow to give a theoretical proof of convergence. A deterministic model for the analysis of ACO is presented in [Merkle and Middendorf, 2001d, 2002c,b, Merkle, 2002]. Assuming an average expected behavior of the algorithm, the authors analytically demonstrate how the properties of the pheromone matrix influence ant decisions. The results explain the complex dynamic behavior of ACO even in the case of just one ant per iteration, i.e. without competition.

Chapter 3

Reconfigurable Computing

Optimization algorithms, such as the previously introduced ant algorithms, traditionally are of significant interest to computer scientists, whereas the development and production of hardware circuits and devices has for a long time been the responsibility of electrical engineers. Reconfigurable computing as a new albeit rapidly establishing paradigm integrates both disciplines and demands a great deal of research in the field of reconfigurable technology as well as tools supporting the efficient design and execution of applications based on reconfigurable systems.

This chapter introduces reconfigurable computing, describing its characteristics and differentiating it from other conventional computing techniques. Thereafter, a classification of various reconfigurable architectures and an overview of abstract models is given. The main emphasis is thereby put on the architectural domain, especially on Field-Programmable Gate Arrays (FPGAs), which have provided the basis for the most part of recent advances in reconfigurable computing. This is followed by a discussion of the models, benefits, and problems involved with reconfiguring hardware at runtime. Finally, an overview of various important applications of reconfigurable computing systems, a summary of main trends as well as an outlook on emerging directions are provided.

3.1 Introduction

The term *reconfigurable computing* refers to operations performed on programmable hardware, which is customized via a number of physical control points distributed over the target device. If the hardware permits, these control points can be reconfigured, i.e. programmed multiple times to adapt the functionality of the device to changing applications. Some devices even allow the settings of a subset of the available control points to be changed during the runtime of the imple-

mented circuit, such that parts of a running application can be altered while the rest continues to operate. This way of changing the structure and functionality of an application is known as *partial reconfiguration*.

The concept of reconfigurable computing in a wider sense has been known for a considerably long time [Estrin et al., 1963]. Even in current general-purpose processors some sort of reconfiguration is applied in as far as several computational components can be reused for independent computations with multiplexers controlling the routing to these components. Reconfigurable computing in the stronger sense, however, as agreed by current researchers, refers to such systems incorporating a certain degree of hardware programmability as explained above. Essentially driven by the introduction of FPGA technology in the mid-1980s, reconfigurable computing has attracted a great deal of research. However, reconfigurable computing is still a maturing field.

Being different from the von-Neumann computing paradigm, reconfigurable computing requires new computational models, which vary substantially from the conventional models (mainly relying on a processor fetching and executing instructions from a sequence). Also tools are needed, which continuously support the design of applications based on reconfigurable systems. This lack of models and tools is known as the *2nd design crisis* in Makimoto's wave describing and predicting a cyclic development within the semiconductor market [Makimoto, 2000].

A variety of devices is currently available for developing and implementing digital systems. Usually, a designer can choose from a wealth of software-oriented devices like general-purpose-processors, micro-controllers, digital signal processors, or Application-Specific Instruction Set Processors (ASIPs). On the other hand, hardware devices – so-called Application-Specific Integrated Circuits (ASICs) – are available, which are designed for predefined processing tasks. Reconfigurable computing systems can be considered to be located at the intersection between software and hardware-oriented systems.

General-purpose processors offer the highest degree of flexibility. By changing certain instructions in a program running on a processor the software can easily be altered with no need to redesign the hardware. However, for many applications, single processors – even very expensive high-performance chips – are often too slow: Every instruction must be read from memory, be decoded and then executed thereby causing a high execution overhead. With a power consumption often in the range around 100W and costs of possibly thousands of dollars per device, such processors are economically not justifiable for many embedded applications, where small form factors and low power consumption are important issues.

ASICs are constructed to solve specific problems. Their ability to use parallelism and pipelining can increase processing performance significantly, but their lack of flexibility is their major disadvantage. Due to their cost structure – mainly

influenced by the mask production cost (ranging up to $1 million [Saxe and Faith, 2004]) – ASICs are suited for mass production but they are inappropriate for prototyping and development. Their long developing and manufacturing times extend the duration of early phases in the product life cycle and may therefore not meet time-to-market requirements.

By providing a programmable selection of alternate logic and routing structures, *reconfigurable computing systems* allow circuits to be designed and implemented very rapidly, thereby avoiding the up-front cost of designing custom circuits and providing a quick means of correcting design errors. After configuring the circuit onto the hardware, it is switched into operational mode, upon which the inherent parallelism and pipelined design style can offer considerable speedup over instruction stream processors. Reconfigurable computing systems like FPGAs facilitate system development, they also allow for easy and quick design changes and verification. Not only are they suitable for rapid prototyping, they can also substitute for standard logic and gate array solutions in small and medium volume productions. However, their high level of flexibility demands additional switches and routing, which in turn increase circuit delays and chip area relative to custom fabricated circuits. Since their introduction FPGAs have run through a rapid progress in execution speed, memory technology, capacity and versatility. If this process continues together with the reduction in price the attractiveness of reconfigurable computing will be further increased.

3.2 Architectures

This section describes the characteristics of reconfigurable computing architectures from a system level as well as from a component level point of view. Since in most reconfigurable computing systems one or more FPGAs create the reconfigurable part, field programmable technology is explained in greater detail.

3.2.1 Classification

A wealth of reconfigurable architectures has been developed by industry and the research community. These architectures can be classified using several parameters. A choice of appropriate distinguishing parameters is listed below [Bondalapati and Prasanna, 2002, Compton and Hauck, 2002]:

Host coupling: Commonly, reconfigurable computing systems are more or less tightly coupled to a host processor, which is amongst other tasks responsible for configuring the reconfigurable fabric, scheduling input and output data, executing control functions, or interfacing external devices. Depending on

the respective applications, the bandwidth and speed of the communication channel between CPU and reconfigurable fabric has a strong impact on the achievable execution speed. Section 3.2.3 differentiates reconfigurable systems by their degree of coupling to a host processor.

Granularity: The *granularity* of reconfigurable architectures is determined by the smallest structure that can be addressed by mapping tools. Granularity refers to the size of logic elements as well as connective structures. *Fine-grained* architectures allow for a high degree of flexibility, and the circuits can be suitably tailored according to the specification demands. However, when composing large designs, the fine-grained structures may possibly cause a considerable overhead slowing down computational speed. Therefore, some fine-grained architectures are enhanced with special-purpose logic and routing like fast carry chains, which allow large arithmetic circuits to be composed from very small functional units. *Coarse-grained* architectures provide components specialized in word-width computations such that certain applications can be executed very quickly, but as their components are static, they are unable to properly adapt to optimizations in the size of operands. Other *medium-grained* architectures offer a compromise between these two extremes. More and more architectures come up with components of *mixed granularity*, they combine very fine-grained bit-level logic with embedded special-purpose word-width blocks. Examples of configurable architectures with different granularity are given in Section 3.2.4.

Reconfiguration method: The speed of configuring and reconfiguring a device or parts of a circuit depends on the configuration interface. Some devices support bit-parallel, others allow for bit-serial configuration. The attainable configuration speed (often in the magnitude of many milliseconds) is crucial and influences the applicability of circuit reconfigurations. Depending on the application and capabilities of the device, reconfigurations can be performed at different stages. Reconfiguration can take place prior to compiling the circuit (*compile time reconfiguration*), or the configuration stream is adapted on-the-fly when uploading the configuration data (*load time reconfiguration*), or during the application runtime (*runtime reconfiguration*). Runtime reconfiguration is one of the most promising features of reconfigurable computing systems as it can potentially help saving chip area, energy, and execution time. Reconfiguration at runtime is further discussed in Section 3.4.

Memory: Reconfigurable architectures provide different types of memory to store computation and configuration data. Off-chip memory offers high

capacity storage at low cost, however, communication delays can be prohibitively long. On-chip memory is commonly smaller and more expensive, but is also more flexible and can be accessed very quickly. It is therefore important to store frequently needed data in close proximity to the computation blocks. Modern architectures provide on-chip storage by configuring logic resources as distributed memory or in the form of large embedded RAM blocks.

3.2.2 Field Programmable Gate Arrays

FPGAs are the most common micro-chips commercially available providing the reconfigurable fabric of a reconfigurable computing system. Originally, FPGAs were produced as a hybrid architecture located between Programmable Array Logic (PAL) and Mask Programmable Gate Arrays (MPGAs), and were primarily used as a glue-logic replacement or for rapid-prototyping purposes. However, as the capacity, flexibility, and performance of the devices grew, FPGAs come to be considered as an interesting platform for high-performance and reconfigurable computing.

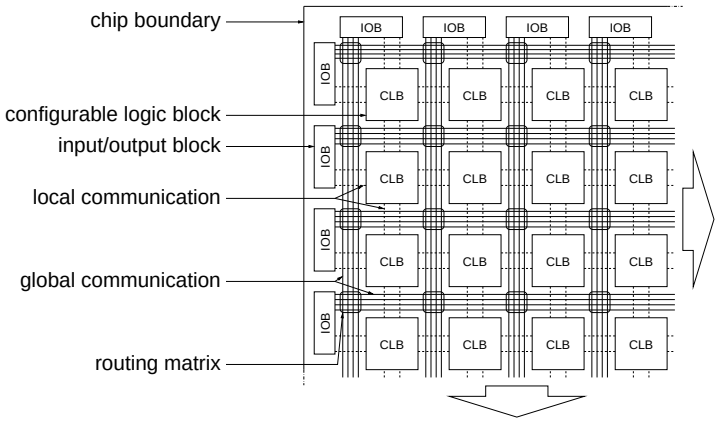


Figure 3.1: Simplified schematic of an FPGA, top-left corner.

Figure 3.1 depicts a simplified representation of an FPGA comprising the three major configurable elements commonly present:

- *Configurable logic blocks* (CLBs) provide the basic functional components for implementing logic and registers.
- *Input/output blocks* (IOBs) interface the routing network and the package pins.

- The *routing network* consists of horizontal and vertical multi-track channels and configurable switches that allow logic blocks to be interconnected to form complex computational structures.

Usually, CLBs consist of two or three look-up tables (LUTs) and two flip-flops. Any LUT can be configured to either compute an arbitrary boolean function with a restricted number of input signals, or they can be used as a small RAM providing storage for up to 16 bits. Horizontal and vertical communication resources provide configurable connections among CLBs or between CLBs and IOBs. Global buses of different length connect components across longer distances, whereas local wires allow for a fast communication between direct neighbors. User-defined linkages between bus lines can be established either within routing matrices or at specific programmable interconnect points (omitted in Figure 3.1).

3.2.3 Host Coupling

Typically, a reconfigurable computing system comprises components like processors, reconfigurable fabrics, memory/caches, interfaces, and a communication network as visualized in Figure 3.2. Depending on their degree of host coupling, reconfigurable computing systems can be divided into five different classes. In an ascending order of coupling degree these are the stand-alone processing unit, the attached processing unit, the co-processor, the reconfigurable functional unit (RFU), and the embedded processing unit [Compton and Hauck, 2002, Todman et al., 2005].

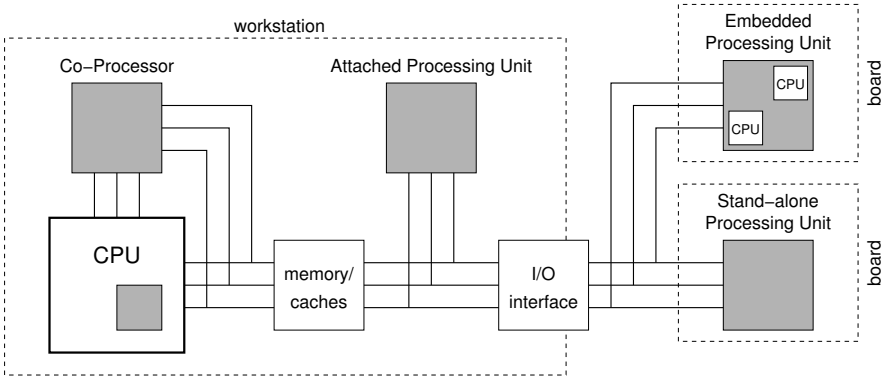


Figure 3.2: Different types of host coupling of reconfigurable logic on system level. Reconfigurable components are shown as shaded boxes. Complemented and updated based on [Compton and Hauck, 2002].

Stand-alone Processing Unit: Commonly, a stand-alone processing unit consists of one or more reconfigurable fabrics accommodated on an external board communicating with the host processor (if present) via an I/O interface. As the data transfer is relatively slow, this most loosely coupled type of system configuration only makes sense if the application on the reconfigurable fabric can be executed with only little processor intervention. However, stand-alone processing units are very popular, especially for emulation purposes, e.g., Cadence Palladium II [Cadence Design Systems Inc., 2005] or Mentor Graphics VStation Pro [Graphics, 2003].

Attached Processing Unit: Circumventing peripheral I/O interfaces an attached processing unit behaves like an additional processor in a multiprocessor system or like an additional compute engine. Typically attached to internal high speed communication buses, however, without access to the host processor memory cache, this kind of reconfigurable computing system is well-suited for CPU-independent computations with only semi-frequent communications [Vuillemin et al., 1996, Annapolis Microsystems Inc., 1998, Laufer et al., 1999, Singh et al., 2000, Goldstein et al., 2000, Leong et al., 2001, Schmit et al., 2002].

Co-Processor: Reconfigurable components can also act as a co-processor, which at start-up is initialized by the host processor and can perform then more or less independent operations. Generally, the host CPU sends the processing input data directly to the co-processor or provides the addresses of the memory locations holding the required information. In contrast to the attached processing unit, the co-processor can also make use of the CPU memory caches [Wittig and Chow, 1996, Hauser and Wawrzynek, 1997, Miyamori and Olukotun, 1998, Rupp et al., 1998, Chameleon Systems Inc., 2000].

Reconfigurable Functional Unit: As an embedded component on the processor chip itself, the reconfigurable functional unit (RFU) represents the tightest possible coupling with the host. Sharing common registers to exchange input and output parameters, the RFU executes custom instructions and allows to use traditional programming environments [Razdan and Smith, 1994, Mirsky and DeHon, 1996, Hauck et al., 1997, Marshall et al., 1999, Taylor et al., 2002].

Embedded Processing Unit: A further emerging variant of tightest host coupling are CPU blocks embedded into the reconfigurable fabric. These blocks can be either hardware cores (i.e. physical CPU blocks) [Altera Corp., 2002, Xilinx Inc., 2005b], or software cores (i.e. synthesizable circuit descriptions)

[Seng et al., 2000, Fidjeland et al., 2002, Leong and Leung, 2002, Seng et al., 2002, Altera Corp., 2005, Xilinx Inc., 2005a, Dimond et al., 2005]. Given that such platforms have in-built CPUs, they can work completely stand-alone, however, it may also be possible to couple them in different ways to a workstation processor.

3.2.4 Granularity

In most cases reconfigurable hardware consists of an array of reconfigurable logic blocks linked by a network of programmable interconnect. The granularity of these structures may vary from very small components executing bit-wise operations up to large components like arithmetic logic units (ALUs) or multipliers. The following survey gives a brief summary of the characteristics and some examples of architectures with different degrees of granularity [Compton and Hauck, 2002, Todman et al., 2005].

Fine-grained: The most common example of fine-grained logic in reconfigurable logic are the lookup tables (LUTs) inside configurable logic blocks of FPGAs, e.g., [Xilinx Inc., 2002, Actel Corp., 2004]. Typically, LUTs operate on inputs of three to four bits and can also be combined to compute more complex functions. Extremely fine-grained logic blocks could be found in the FPGAs series Xilinx XC6200 [Xilinx Inc., 1996]. However, the production and sale of this FPGA family has been discontinued to the regret of many researchers, who favored the chip for its accessibility and directly addressable configuration registers. Fine-grained architectures are flexible and well-suited to perform bit manipulations, but produce much overhead when implementing more complex functions like multiplications or exponentiations. Also computations on complex data types, e.g., floating-point numbers consume an enormous amount of chip area. Although there exist floating-point encodings adapted to fine-grained architectures [Shirazi et al., 1995b], and FPGAs can very well compete with CPUs in terms of floating-point execution speed [Underwood, 2004], due to space requirements, applications should avoid a large number of parallel floating-point circuits or seek for an alternative approximation by integer numbers, or move such operations into specialized logic if available.

Medium-grained: Architectures with medium-grained technology are built of components with a larger number of input bits and may efficiently support more complex functions at the cost of reduced flexibility. Such architecture may be composed of configurable ALUs or multiplier blocks [Hauser and Wawrzynek, 1997, Haynes and Cheung, 1998, Marshall et al., 1999, Altera Corp., 2004].

Coarse-grained: Devices based on coarse-grained logic block are primarily devoted to special computations on word-width data and can perform these operations much quicker consuming less chip area than finer-grained architectures. However, computations on smaller than the intended bit-width lead to a waste of computing resources, and the variety of efficiently programmable functions is much more restricted. Architectures of this types may be composed of word-sized adders, multipliers, shifters, and ALUs [Ebeling et al., 1996, Mirsky and DeHon, 1996, Laufer et al., 1999, Goldstein et al., 2000, Chameleon Systems Inc., 2000, Singh et al., 2000, Becker and Glesner, 2001, Schmit et al., 2002, Elixent Corp., 2003, Mei et al., 2003, Veredas et al., 2005], or they consist of an array of small processors [Moritz et al., 1998, Miyamori and Olukotun, 1998].

Mixed-grained: One way to circumvent the disadvantages inherent to homogeneously grained architectures is to combine components of different granularity into a heterogeneous structure. Most often an array of fine-grained logic is enhanced by embedded blocks for special purposes like memory blocks [Haynes and Cheung, 1998, Chameleon Systems Inc., 2000, Xilinx Inc., 2002, Atmel Corp., 2004, Xilinx Inc., 2005e,f,d], multipliers [Xilinx Inc., 2005e,f,d], digital signal processing blocks [Altera Corp., 2004, Xilinx Inc., 2005d], or CPUs [Altera Corp., 2002, Xilinx Inc., 2005f,d]

3.3 Abstract Models

Reconfigurable computing is not restricted to the advances of semiconductor technology. A range of models exists satisfying the special needs for examining the capabilities of reconfigurable computations. The following survey does not seek to cover every related model. It rather presents a selection of some interesting and important contributions in this research sector. Due to its relevance for the implementation of ACO algorithms, the RMesh model is described in greater detail.

3.3.1 Overview

Several approaches are closely FPGA-oriented providing an abstract, virtual view on the logic and routing resources with the aim to achieve a device-independent and portable way of programming or to enhance the restricted capabilities of the physical hardware.

A *virtual FPGA*, very similar to the concept of virtual machines, is proposed by [Lagadec et al., 2001]. The virtual FPGA is considered to be a simplified

variant of a physical FPGA. The model consists of a regular pattern of virtual configurable logic blocks arranged in a so-called virtual layer. Applications are mapped, placed and routed onto this device-independent layer. The resulting portable code can then be transformed to be implemented on the target platform (physical layer) using vendor-specific tools.

Another variant of a virtual FPGA is presented by [Fornaciari and Piuri, 1998]. As opposed to the previously described model, the goal is not to provide a device-independent programming interface, but to extend the capabilities of the physical hardware. It is intended to realize very large applications with no need for a large FPGA or multiple connected FPGAs. Furthermore, a multitasking and time-shared operating system should allow multiple applications to run in parallel. These applications have only a virtual view of the FPGA, the operating system is responsible for mapping the applications onto the physical device in a way similar to virtual memory. A further model to virtually enhance the hardware capabilities is proposed by [Babb et al., 1997]. The authors suggest to overcome the communication bottleneck on I/O pins of FPGAs by a concept called *Virtual Wires*. The idea is to multiplex each physical wire among multiple logical wires. Communication via these connections is then pipelined at the maximum attainable clocking frequency.

In [Bolotski et al., 1994] a computational model is presented, which unifies SIMD (Single Instruction Multiple Data) arrays and FPGAs. The model is composed of a grid of array elements with interconnection resources linking these elements together. Within a single clock cycle, every array element can perform a simple function depending on several states and input data coming from the array, and can update its own state to store its own processing results or share these results with other array elements.

This model has some similarities with the *Reconfigurable Mesh* (RMesh) [Miller et al., 1993, Jang et al., 1994], which also consists of a grid of processing elements with local communication buses (see Section 3.3.2). The RMesh can further create global communication buses, which can be changed dynamically. In contrast to the preceding models, the abstract RMesh model is stronger decoupled from physical hardware (e.g. FPGAs). It is the reconfigurable bus system that makes the RMesh a powerful computational model, but there exist also other concepts for reconfigurable bus networks, e.g., the Reconfigurable Multiple Bus (RMB) [ElGindy et al., 1996] or the Segmentable Bus [Trahan et al., 1996].

The *Hybrid System Architecture Model* (HySAM) [Bondalapati, 2001] consists of a von-Neumann style processor, a configurable logic unit (CLU), memory, and a configuration cache linked together via an arbitrary communication network. Computations on this model comprise a declarative aspect and a generative aspect. The declarative aspect refers to the specification of the parameterized hybrid architecture model and creates the basis for algorithmic analysis. The

generative aspect describes how the declarative aspect can evolve by the use of generative functions, which execute rule based transformations of model parameters.

3.3.2 Reconfigurable Mesh

The Reconfigurable Mesh represents an abstract model of a runtime reconfigurable architecture [Miller et al., 1993, Jang et al., 1994]. A brief overview of research and applications on the RMesh is provided by [Bondalapati and Prasanna, 1997]. The RMesh is distinguished by its reconfigurable bus systems connecting an $n \times n$ grid of processing elements (PEs).

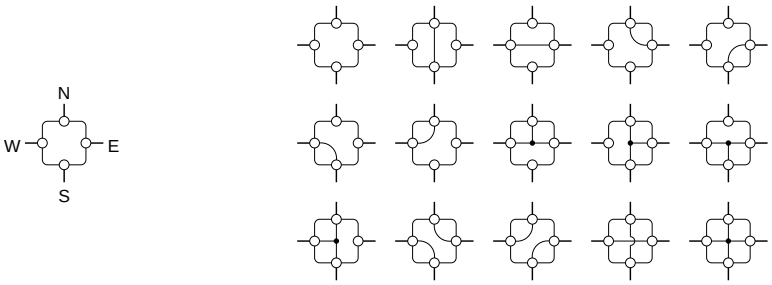


Figure 3.3: RMesh PE (left) and its 15 possible configurations (right).

Each PE contains four I/O ports labelled north (N), east (E), south (S), and west (W) as visualized in Figure 3.3. Via these ports every PE can communicate with its immediate neighbors. Furthermore, the ports of every PE can be connected internally by switched lines according to one of 15 possible configurations. The PEs are allowed to configure the internal connections autonomously based on local state information. By means of these internally wired ports the RMesh can create a runtime reconfigurable communication bus thereby forming a dynamic topology amongst the PEs (see Figure 3.4 for example of an RMesh with a configured bus topology).

The PEs are supposed to operate synchronously. Within every machine cycle, each PE is allowed to locally configure the bus, to read/write from/to one of the buses it is connected to, and to perform an arithmetic, logic or control operation. There exist different assumptions on the further capabilities and restrictions of the RMesh resources. With respect to the bus arbitration, the PE may communicate as per a concurrent read exclusive write policy (CREW), i.e. all connected PEs may simultaneously read from the bus, but it must be ensured that at most one at a time writes to the bus. Following the CRCW policy, the PEs can under certain conditions also perform a concurrent write operation, e.g., the write

conflict can be resolved by forming the bitwise OR over the written output. Also different delay models may apply, e.g., the *unit time delay model* assumes that all broadcasts along a bus require $\Theta(1)$ time irrespective of the actual bus length, whereas in the *log-time delay model*, a broadcast takes $\Theta(\log s)$ with s denoting the maximum number of intermediate switches between two PEs communicating via the bus.

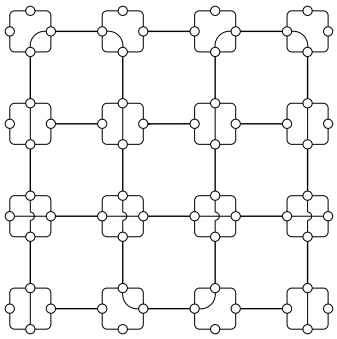


Figure 3.4: Example of a 4×4 RMesh with configured bus.

The RMesh, a powerful computation model, has attracted a great deal of research. However, so far there exists no realization of this model in real hardware, although it is possible to develop RMesh programs targeting simulation and visualization tools proposed by [Steckel et al., 1998, Bordim et al., 1999, Miyashita and Hashimoto, 2000].

3.4 Runtime Reconfiguration (RTR)

Reconfiguring hardware at runtime means that the programming bits of the configured circuit are changed while the implemented application is being executed. Runtime reconfiguration can be beneficial if the application is (or multiple parallel applications are) too large to fit entirely onto the available resources. This means that during the runtime, repeatedly different configurations are swapped in and out of the reconfigurable device as they are required by the application. Furthermore, runtime reconfiguration allows for new algorithmic techniques, which accelerate the application by transferring certain operations into the structure of reconfigurable logic and routing resources.

The following sections introduce various models of runtime reconfiguration and discuss the techniques, the challenges, and the problems involved with reconfiguration at runtime. The major problem identified is the slow configuration

speed. Therefore, several strategies are introduced to cope with this configuration latency.

3.4.1 Reconfiguration Models

Loading configuration data or reconfiguring part of a circuit requires considerably long time. To efficiently exploit the advantages of dynamic reconfiguration, it is therefore necessary that configuration changes occur not too often or can be hidden behind computation. The methods of reconfiguration can be classified into four different reconfiguration models: single context, multi-context, partial, and pipeline reconfiguration. The following overview briefly describes the characteristics of these models and gives examples of related systems:

Single context reconfiguration: Commonly, FPGAs are configured by uploading a serial stream of programming bits. Such devices solely supporting single context reconfigurations do not allow to individually access the configuration registers. Any configuration change affords an entire reprogramming of the complete chip. Single context devices get by with relatively primitive reconfiguration logic, however, complete context swaps incur a high overhead if only small portions of the configuration memory actually need to be changed. Single context devices currently available include, e.g., Altera Flex10K [Altera Corp., 2003] or Xilinx Spartan II [Xilinx Inc., 2004].

Multi-context reconfiguration: In Multi-context FPGAs, the configuration points possess multiple configuration bits. At any time, one of these bits is considered as active, i.e. is loaded into the configuration point. Multi-context devices can quickly switch between several configuration bits each belonging to a different context. Depending on the specific device, it is possible to load configuration data into inactive contexts in the background while the active context is being executed. However, as in the case of single context devices, changing only a portion of configuration bits also requires reloading an entire context. While the loading time for a new context is also in the range of milliseconds, the switching between different loaded configurations can be accomplished within nanoseconds. Examples of multi-context systems are presented in [Trimberger et al., 1997, Scalera and Vazquez, 1998, Chameleon Systems Inc., 2000, Puttegowda et al., 2003].

Partial reconfiguration: Using partial reconfiguration only a portion of the configuration bits on the chip are altered. It comes in very handy that unaltered portions of the circuit can continue operating allowing an overlap between computation and reconfiguration. It is not necessary to reprogram the complete device. As opposed to single and multi-context devices, partial

reconfiguration requires the configuration registers to be addressable similar to memory. This means that programming a partially reconfigurable device is potentially subject to longer configuration delays as not only configuration data, but also addresses have to be loaded. The Xilinx XC6200 FPGA [Xilinx Inc., 1996] for instance allowed a very convenient way of partial reconfiguration, since all configuration bits could be addressed separately. Xilinx has substituted this series with the Virtex family of FPGAs [Xilinx Inc., 2002, 2005e,f,d], which only allow for reconfigurations of complete columns. Other partially reconfigurable systems include, e.g., Chimaera [Hauck et al., 1997], PipeRench [Goldstein et al., 2000], or PACT XPP [Baumgarten et al., 2003].

Pipeline reconfiguration: Pipeline reconfigurable systems apply the concept of pipelining to processing data as well as to configuration data. Such systems (also called striped FPGAs) contain circuits executing on pipelined processing data while other portions undergo partial reconfiguration. As soon as reconfiguration is complete, the configured circuitry starts operating while another portion is being terminated and then reconfigured. Therefore, computation can overlap with reconfiguration in a pipelined fashion. Examples for pipeline reconfiguration are given in [Luk et al., 1997c, Goldstein et al., 2000, Dasasathyan et al., 2002]

3.4.2 Partial Evaluation

A circuit mapped onto a device should occupy as little area as possible. Furthermore, the processing time for the circuit is to be minimized. *Partial evaluation* is a method to reduce the hardware requirements by performing optimizations based on static constants or infrequently changing values [Compton and Hauck, 2002]. This optimization can be applied at compile time and at runtime (by means of runtime reconfiguration). At compile time, the idea is to propagate constant values through the gates and to only map those portions of the circuit into hardware, which depend on time-variant input data. The resulting circuit is likely to be smaller and faster than a design that assumes all input values to be variable.

Partial evaluation at runtime can consider input values, which are supposed to change only very infrequently. Such input values are compiled similar to constant values, however, when at a time the respective input is changed, the affected portion of circuitry is partially reconfigured so as to continue operating on a different constant. This method gives partial evaluation an advantage over ASIC devices, which always have to provide logic for generalized input values. Examples for applications using partial evaluation at runtime are constant coefficient multi-

pliers or key-specific reconfigurable encryptors/decryptors. *Constant coefficient multipliers* can substitute general-purpose multipliers for a sequence of additions on the variable input, which is repeatedly shifted by static lengths. Changes to the constant coefficient invoke an alteration of the multiplier by partial reconfiguration [Payne, 1997, Wirthlin and McMurtrey, 2001]. *Key-specific encryption* and *decryption* are optimized for a specific constant key, but can be adapted to different keys using runtime reconfiguration [Leonard and Mangione-Smith, 1997].

3.4.3 Accelerating Configuration Speed

As mentioned above, the re-/configuration of currently available reconfigurable architectures takes relatively long time in the range of many milliseconds. Therefore reconfiguration causes an overhead, which may under some circumstances abolish the gain of execution speed. Hence, the acceleration of configuration times is a crucial aspect in the area of reconfigurable computing, and various strategies have been developed to reduce the incurred overhead: prefetching, compression, and caching of configurations as well as rearrangement and defragmentation:

Prefetching: Considering a cooperative system consisting of a host CPU and reconfigurable device, it should be prevented that the CPU execution is stalled to wait for the configuration to complete. Configuration prefetching [Hauck, 1998, Resano et al., 2005] therefore tries to maximize the overlap of reconfigurations and useful computations on the host processor. It predetermines the time when to initiate reconfigurations and minimizes the chance of false prefetchings, which would overwrite configurations actually executed next.

Compression: Another way to speed-up the reconfiguration is to reduce the amount of data transferred in the configuration bit-stream by compression techniques as discussed, e.g., in [Hauck and Wilson, 1998, Dandalis and Prasanna, 2001]. Compression algorithms are actually applied to purchasable FPGAs like Altera Stratix II [Altera Corp., 2004], but also the wildcarding techniques [Hauck et al., 1998, James-Roxby and Cerro-Prada, 1999, Li and Hauck, 1999] for the obsolete Xilinx XC6200 series [Xilinx Inc., 1996] have been largely investigated. A further possibility to reduce the amount of data sent is to reuse configuration bits already stored within the array and to only toggle those bits which really need to be changed. This kind of incremental reconfiguration is proposed in [Luk et al., 1997b, Shirazi et al., 1998] and is used by, e.g., Xilinx Spartan 3 FPGAs [Xilinx Inc., 2005c]. Hyper-reconfiguration as another concept to reduce the

amount of reconfiguration information is introduced by [Lange and Middendorf, 2004]. The main idea is that typically applications running on a device do not use dynamic reconfiguration all the time. The authors therefore regard the potential of reconfiguration itself as reconfigurable. By this means reconfiguration operations are accelerated whenever only a fraction of the total flexibility is actually needed.

Caching: A great proportion of configuration delay is caused by the distance and small bandwidth between reconfigurable device and host processor or due to high access time when reading from files or memory. Storing and buffering configuration data in a fast cache directly attached to the reconfigurable unit can therefore help reducing reconfiguration delays [Deshpande et al., 1999, Compton et al., 2000].

Rearrangement and defragmentation: Partially reconfigurable FPGAs can be shared among multiple independent applications (tasks). An attached host processor controlling the execution of tasks on the FPGA can decide online where and when to place new tasks. Since online allocation suffers from fragmentation of the chip area, tasks can end up waiting despite there being sufficient, albeit non-contiguous resources available to service them. Rearranging a subset of the tasks executing on the FPGA often allows the next pending task to be processed sooner such that configuration latency is also reduced. Evolutionary Algorithms and several other heuristics for the rearrangement of tasks and defragmentation of the FPGA area are presented, for instance, by the author of this thesis in [Scheuermann, 1999, Diessel et al., 2000, ElGindy et al., 2000, 2002].

3.5 Applications

The following list gives a survey of applications of reconfigurable architectures and models.

Audio processing	
Digital-to-analog-Converter	[Cheung et al., 2003b, Ludewig et al., 2004]
Speech recognition	[Brucke et al., 1999, Melnikoff et al., 2000, 2001, 2002]
	[Stogiannos et al., 2000]
Synthesizer	[Raczinski and Sladek, 2000, Saito et al., 2001, Sheidaei et al., 2001]
Video processing	
Encoding	[Schoner et al., 1995, Sanz et al., 1996, Lienhart et al., 2001]
	[Ramachandran and Srinivasan, 2001, Roma et al., 2003]
	[Denolf et al., 2005, Janiaut et al., 2005, Lehtoranta et al., 2005]
Decoding	[Verderber et al., 2003, Gorgon et al., 2005]
Convolution	[Ratha et al., 1995, Haynes et al., 1999, Sedcole et al., 2003]
3D-Vision	[Dunn and Corke, 1997, Bellis and Marnane, 2000]
	[Arias-Estrada and Xicotencatl, 2001]
	[Miyajima and Maruyama, 2003]

Image processing	
Line/shape detection	[Abbott et al., 1994, Pan, 2000, Guo et al., 2004]
	[Nagata and Maruyama, 2004]
Image recognition	[Villasenor et al., 1996, Rencher and Hutchings, 1997]
	[Zakerolhosseini et al., 1998, McCready, 2000]
	[Williams et al., 2002, Paschalakis and Bober, 2003]
	[Yamada et al., 2003, Maruo et al., 2004]
Filter	[Datta, 1999, Voß and Mertsching, 2001, Coric et al., 2002]
	[Srivastava et al., 2003, Fahmya et al., 2005]
Rendering/ray tracing	[MacVicar et al., 1999, Ye and Lewis, 1999, Fender and Rose, 2003]
Compression	[Fagin and Chintrakulchai, 1994, Nagano et al., 1999]
	[Ritter and Molitor, 2001, Simpson et al., 2001]
	[Boluda and Pardo, 2003, Gangadhar and Bhatia, 2003]
Other digital signal processing (DSP)	
Fourier transform	[Panneerselvam et al., 1995, Shirazi et al., 1995a, Dick, 1996]
	[Choi et al., 2003, Ebeling et al., 2003, Uzun et al., 2003]
Modulation	[Rowley and Lyden, 1996, Ramírez et al., 2000, Pan et al., 2001]
	[Jamin and Mähönen, 2002]
Demodulation	[Meyer-Bäse, 1996, Jong et al., 1997, Cardells-Tormo et al., 2002]
	[Dick and Harris, 2002]
Filter	[Dick and Harris, 1996, Do et al., 1998, Haynes et al., 1999]
	[Ting et al., 2001, Carreira et al., 2002, Turner et al., 2002]
	[Yeung and Chan, 2002, Benkrid et al., 2003]
Communication	
Routing	[Haddy and Skellern, 1996, Lockwood et al., 2000, 2001]
	[Braun et al., 2001, Majer et al., 2005, Schelle and Grunwald, 2005]
	[Wolkotte et al., 2005]
Wireless network	[McDermott et al., 1997, Rabaey, 2000, Swaminathan et al., 2002]
	[Oliver and Akella, 2003, de Souza et al., 2003, Canet et al., 2004]
	[Esquiagola et al., 2005, Saito et al., 2005]
Mobile communication	[Becker et al., 2000, Blaickner et al., 2000, Revés et al., 2000]
	[Shankiti and Leeser, 2000, Pionteck et al., 2003, Chugh et al., 2005]
Stream scanning	[Ditmar et al., 2000, Miyazaki et al., 2002]
	[Sugawara et al., 2004, 2005]
Error correction	[Babvey et al., 2005]
Network security	*IPsec*
	[Bellows et al., 2003, Lu and Lockwood, 2005]
	Intrusion detection
	[Clark and Schimmel, 2003, Sourdis and Pnevmatikatos, 2003]
	[Song and Lockwood, 2005]
	Firewalls
	[McHenry et al., 1997, Lee et al., 2003]
Cryptography	
A5	[Charlwood and James-Roxby, 1998]
AES	[Elbirt and Paar, 2000, Chodowiec et al., 2001]
	[McLoone and McCanny, 2001, McMillan and Patterson, 2001]
	[Tong et al., 2002, Adachi et al., 2003, Charot et al., 2003]
	[Järvinen et al., 2003, Saggese et al., 2003, Saqib et al., 2003]
	[Standaert et al., 2003a, Pramstaller and Wolkerstorfer, 2004]
	[Zambreno et al., 2004]
Blowfish	[Charlwood and James-Roxby, 1998]
Camellia	[Ichikawa et al., 2001, Denning et al., 2004]
CAST-128	[Nastou and Stamatiou, 2002]
DES/Triple-DES	[Hauser and Wawrzynek, 1997, Leonard and Mangione-Smith, 1997]
	[Chodowiec et al., 2001, Cheung and Leong, 2002]
	[Rouvroy et al., 2003, Standaert et al., 2003b, 2004]
ECC	[Leung et al., 2000, Bednara et al., 2002, Leong and Leung, 2002]
	[Daly et al., 2003, Nguyen et al., 2003, Kumar and Paar, 2004]
	[Telle et al., 2004, Cheung et al., 2005]
IDEA	[Gonzalez et al., 2003, Michalski et al., 2003]
RC6	[Chodowiec et al., 2001, Beuchat, 2003]
RSA	[Kim and Mangione-Smith, 2000, Ploog et al., 2000]

	[Chodowiec et al., 2001, Standaert et al., 2003b, Tang et al., 2003]
	[Crowe et al., 2004, Nibouche et al., 2004]
SHACAL	[McLoone and McCanny, 2003]
Twofish	[Chodowiec et al., 2001]
Optimization problems	
Boolean satisfiability	[Suyama et al., 1996, 1998, 1999, 2001, Abramovici and Saab, 1997]
	[Hamadi and Merceron, 1997, Zhong et al., 1998, Yung et al., 1999]
	[Abramovici and de Sousa, 2000, Platzner, 2000]
	[Boyd and Larrabee, 2000, Redekopp and Dandalis, 2000]
	[Dandalis et al., 2001, de Sousa et al., 2001]
	[Skliarova and Ferrari, 2002, Yap et al., 2003, Sklyarov et al., 2004]
	[Tavares et al., 2004, Kanazawa and Maruyama, 2005]
Clustering	[Estlick et al., 2001]
Golomb ruler derivation	[Dollas et al., 1998, Sotiriades et al., 2000]
Graph problems	[Babb et al., 1996, Dandalis et al., 1999]
	[Ichikawa et al., 2000]
Maximum clique	[Wakabayashi and Kikuchi, 2004]
Minimum covering	[Plessl and Platzner, 2001]
N-Dame	[Abramson et al., 1998b]
Sorting	[Yeh et al., 1999]
String pattern matching	[Pryor et al., 1993, Sidhu et al., 1999b, Weinhardt and Luk, 1999]
	[Bobda and Lehmann, 2000, Baker and Prasanna, 2004]
Meta-heuristics	
Ant colony optimization	[Merkle and Middendorf, 2001b, 2002a, Diessel et al., 2002]
	[Janson et al., 2002, 2003, Scheuermann et al., 2003, 2004a,b]
	[Scheuermann and Middendorf, 2005, Scheuermann et al., 2005]
Genetic algorithms/ Evolutionary algorithms	[Graham and Nelson, 1995, 1996, Scott et al., 1995, 1997a,b]
	[Bland and Megson, 1996, 1998c,a,b, Megson and Bland, 1997]
	[Kajitani et al., 1998, Shackleford et al., 2000]
	[Aporntewan and Chongstitvatana, 2001, Lei et al., 2002]
Genetic programming	[Sidhu et al., 1999a]
Simulated annealing	[Abramson et al., 1998a]
Machine learning	
Neural networks	[Lysaght et al., 1994, Bade and Hutchings, 1994]
	[Eldredge and Hutchings, 1994, Alderighi et al., 1997]
	[Zhu et al., 1999, Zhu and Milne, 2000, Zhu and Sutton, 2003]
	[Maya et al., 2000, Sousa et al., 2003, Thoma et al., 2003]
Bayesian learning	[Pournara et al., 2005]
Cellular automata	[Marchal and Sanchez, 1994, Shackleford et al., 2002]
	[Cerdá et al., 2003, Kobori and Maruyama, 2003, Zipf et al., 2005]
Bio-informatics	
Genome/protein identification	[Lemoine and Merceron, 1995, Alex et al., 2004, Oliver et al., 2005]
Protein sequence alignment	[Dydel and Bala, 2004, Yamaguchi et al., 2004]
Biomolecular systems / chemical reactions	[McCaskill and Wagler, 2000, Osana et al., 2003, Court et al., 2004]
	[Keane et al., 2004, Yoshimi et al., 2004]
Arithmetics	
Additions	[Jebelean, 1995, Laurent et al., 1997, Mencer et al., 2001]
	[Lee and Burgess, 2003a,b]
Multiplications	[Louca et al., 1996, Ahlquist et al., 1999, Andrejas and Trost, 2000]
	[Courtney et al., 2000, Miomo et al., 2000, Mencer et al., 2001]
	[Visavakul et al., 2001, Wires et al., 2001]
	[Wirthlin and McMurtrey, 2001, Daly and Marnane, 2002]
	[Corsonello et al., 2003, Kim et al., 2003]
	[Myjak and Delgado-Frias, 2004, Sidahao et al., 2004]
Divisions	[Andrejas and Trost, 2000, Kim et al., 2003]
Matrix multiplications	[Middendorf et al., 1995, 1999, Amira et al., 2001, Jang et al., 2002]
	[deLorimier and DeHon, 2005, Dou et al., 2005]
	[Zhuo and Prasanna, 2005]

The spectrum of applications is manifold with a rapidly increasing number of contributions published in recent years. Even though FPGAs have been produced for more than 20 years, it took some time before they became widely accepted. Finally, the gate count breaking the 10K barrier considerably leveraged the research and industry activities in the area of reconfigurable computing in the early 1990s.

Most applications originate from the domain of standard logic devices like DSP (including audio, video, and image processing), communication systems, and cryptography. However, the high degree of flexibility, improved tool support, and comparatively low development cost make reconfigurable devices more and more attractive for such applications, which have predominantly been reserved to software engineering. Such applications include algorithms for optimization problems and machine learning. The list above distinguishes between optimization algorithms for specific optimization problems and generic meta-heuristics. The optimization problem, which has attracted most attention is the Boolean Satisfiability Problem (SAT). This might be explained by its practical relevance in the field of electronic engineering, where SAT solvers are needed for testing and debugging of electronic circuits, logic synthesis, pattern recognition etc.

The range of meta-heuristics implemented on reconfigurable architectures and models comprise Ant Colony Optimization, Genetic/Evolutionary Algorithms, Genetic Programming, and Simulated Annealing. Most designs are proposed for Genetic/Evolutionary Algorithms, which might be due to the great success of these meta-heuristics on a wide range of optimization problems. Furthermore, fast on-chip implementations of evolutionary techniques are specially interesting with respect to the development of *Evolvable Hardware* (see [Gordon and Bentley, 2002] for an introduction to Evolvable Hardware). The only known implementations of ACO on reconfigurable hardware target the abstract RMesh model [Merkle and Middendorf, 2001b, 2002a, Janson et al., 2002, 2003] as well as commercially available FPGAs as proposed by the author of this thesis [Diessel et al., 2002, Scheuermann et al., 2003, 2004a,b, Scheuermann and Middendorf, 2005, Scheuermann et al., 2005].

Many authors compare their reconfigurable circuits design with implementations in software on general-purpose processors. The main advantages reported concern the increase in speed and the reduction of energy consumption. For example [Telle et al., 2004] achieve a speedup factor of 540 for Elliptic Curve Cryptography (ECC) implemented on a Xilinx Virtex II pro FPGA running at 66 MHz in comparison to a software implementation on a dual-Xeon processor clocked at 2.6 GHz. A further recent analysis [Stitt et al., 2004] points out that migrating critical software loops into reconfigurable hardware allows to reduce the average energy consumption between 35% and 70% together with an average speedup between 3 and 7. A further overview of attainable speedup factors is

given in [Guo et al., 2004].

3.6 Trends and Emerging Directions

The main trends, which can be observed regarding the current research and developments in the field of reconfigurable architectures, include a move towards coarse-grained and embedded heterogeneous systems as well as software cores. With the continuous advances in VLSI technology it is possible to accommodate more and more logic onto the chip area coming along with an increasing demand for routing resources. To restrict the amount of required interconnect, the granularity of logic blocks is increased, e.g., the Altera Stratix II [Altera Corp., 2004] device uses logic blocks with seven input bits instead of a common composition of LUTs with four input bits. An increasing number of devices are offered with embedded blocks for special purposes, these systems come in variations with distinct combinations of fine-grained and coarse-grained logic, e.g., Xilinx Virtex 4 [Xilinx Inc., 2005d]. Software cores – often offered as so-called intellectual property cores (IP) – become increasingly attractive as they provide a very flexible and convenient way to integrate ready-to-use functions. Even though software cores require more chip area than physical hardware cores, this overhead becomes less hindering due to an increasing count of transistors. Several emerging directions regarding reconfigurable architectures can be observed in the rise of asynchronous architectures [Goldstein et al., 2000, Kagotani and Schmit, 2003, Teifel and Manohar, 2003, Wong et al., 2003] as well as low-power techniques [George et al., 1999, Lamoureux and Wilton, 2003, Rahman and Polavarapuv, 2004, Gayasen et al., 2004] and molecular micro-electronics [Williams and Kuekes, 2000, Butts et al., 2002, DeHon and Wilson, 2004].

Chapter 4

ACO on Reconfigurable Architectures

Ant Colony Optimization offers a meta-heuristic framework for ant-based algorithms, which have been applied successfully to many optimization problems. On the other hand reconfigurable architecture like FPGAs provide the capabilities to design fast circuits at low cost combined with a high degree of flexibility. In this section, it is therefore proposed to investigate the parallel implementation of ACO algorithms targeting field programmable technology. First of all the objectives and opportunities, but also the challenges and restrictions connected with this endeavor are explained. It is shown that fine-grained programmable architectures do not readily support certain data types and operations typically used by conventional ACO algorithms. After a brief overview of related work by other authors, alternative hardware-oriented variants of ACO algorithms, the Counter-based ACO (C-ACO) and the Population-based ACO (P-ACO), are described. Special algorithmic modifications and new techniques are examined for these two approaches. Sequential software simulations comparing C-ACO and P-ACO with standard ACO signify that both variants can be considered as competitive algorithms not only for the hardware implementation on FPGA but also as interesting software substitutes to standard ACO. Subsequently, the properties as well as the asymptotic runtime and resource requirements of both variants are compared with each other.

Note that the description of the henceforth introduced algorithms is restricted to optimization problems with solutions represented by permutations of numbers $\{0, \ldots, n-1\}$. However, appropriate adaptions also allow to apply the proposed algorithms to other non-permutation problems.

4.1 Objectives and Restrictions

Usually, ACO algorithms are implemented in software on sequential machines. Depending on the specific optimization problem, algorithm parameters, and CPU speed, processing times may often range between several minutes and even multiple days. However, if short computation times become crucial, there exist mainly two options to speed-up the execution. One possibility is to develop parallel variants of the algorithm to be executed on multi-processor machines (see Section 2.5.2 for an overview of such ACO algorithms). The other very promising approach is to directly map the ACO algorithm in hardware, thereby exploiting the parallelism and pipelining capabilities of the target architecture. When using reconfigurable architectures, one may further benefit from the advantages of reconfiguration at runtime like smaller circuit sizes, faster execution or reduced power consumption as outlined in the previous chapter.

As implementation platform FPGAs as the de facto only commercially available reconfigurable fabrics are considered. Due to their versatile programmability, FPGAs allow to implement different hardware variants of ACO that can be tested on the same chip. Certain properties of ACO make this meta-heuristic a very good candidate for the implementation in reconfigurable hardware:

Iterative structure: The algorithm mainly consists of a core of iteratively repeated instructions. Thus, most time of the algorithm execution is spent within this core. Mapping this core into hardware may therefore considerably accelerate the algorithm.

Rare conditional branches: Commonly, ACO algorithms contain only a very restricted number of conditional branches, which would otherwise afford complicated control structures. Programs with many conditional operations should rather be reserved to micro-controllers or CPUs.

Variable memory bandwidth requirements: Typically, ACO algorithms process data of various bandwidths. One advantage of reconfigurable devices like FPGAs is that the interface bandwidth for accesses to internal and external memory can be tailored to the specific requirements of the algorithm (though limitations by the available amount of I/O and routing resources have to be considered). Also very high bandwidths interfaces can be created, whereas the typical von-Neumann style computer only provides fixed bandwidths often causing a bottleneck, when data needs to be transferred sequentially through a narrow data path.

Array-based memory operations: The parallel implementation of ACO performs operations on data distributed over local and global memory re-

sources. FPGAs provide storage elements of different latencies and capacities, which suitably adapt to the need of the algorithm.

Concurrency: Many operations performed by ACO are predestined for an intensive parallelization. The hardware implementation allows to accelerate the execution by utilizing spatial (parallel functional units) as well as temporal (pipelining) concurrency.

However, some characteristics of ACO algorithms render a hardware realization on fine-grained reconfigurable device like FPGAs a difficult task:

- Usually, pheromones, heuristic values, and random numbers afford floating-point representations.
- Evaporation and the integration of heuristic information require a large number of multiplication operations. Only few FPGAs have dedicated multiplication blocks though these are restricted in size and number.
- The integration of weights α and β into the selection probabilities p_{ij} (cf. Equation 2.4) demands numerous exponentiation operations.

The realization of the required floating-point numbers, multiplication operations, and exponentiation operations in hardware is possible, but would afford long computation times and a high amount of chip resources on fine-grained programmable logic devices like FPGAs (as outlined in Section 3.2.4).

For these reasons it is proposed to implement alternative ACO approaches, which are specifically adapted to the characteristics of reconfigurable hardware. Two hardware-oriented variants of ACO, the Counter-based ACO and the Population-based ACO are introduced in sections 4.3 and 4.4. The goals of these alternative variants are to speed-up the execution and to economically use logic resources such that large instances of the optimization problem can be accommodated thereby preserving a good optimization performance in comparison to the standard ACO algorithm. To cope with the restrictions posed by the fine-grained hardware, the strategies followed are to replace floating-point numbers by integers, to avoid multiplications or to approximate them by shift operations on a bit level, and to avoid or to pre-calculate exponentiations.

4.2 Related Work

The utilization of reconfigurable computing to accelerate ACO has already been proposed by other authors [Merkle and Middendorf, 2001b, 2002a]. These approaches targeting the abstract computational model of the RMesh do not aim

to provide executable implementations on commercially available platforms. In the following, the proposed algorithm is referred to as *RMesh ACO*. The authors propose to map the pheromone matrix onto the grid of processing elements (PEs) of the RMesh such that every PE holds one pheromone value. A continuous flow of ants is pipelined top-down through the matrix, and every ant makes a decision in the respective row in which it is currently located. Afterwards all ants step one matrix row forward to proceed with the next decision. After the initial n decisions, the first complete solution is constructed, and in every following step, the construction of a further solution is completed.

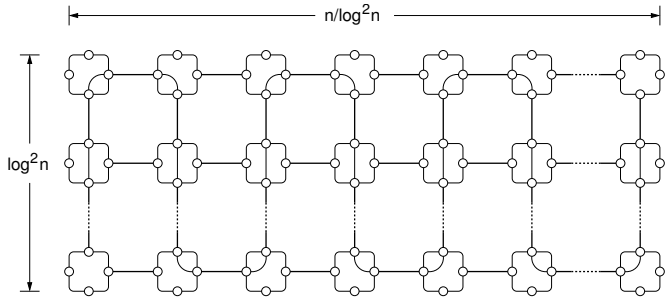


Figure 4.1: Alignment of a single row i of the pheromone matrix into a submesh [Merkle and Middendorf, 2002a].

All pheromone values in a row of the pheromone matrix are arranged in a snake-like form mapped to a $\log^2 n \times (n/log^2 n)$ submesh as visualized in Figure 4.1. Correspondingly, the entire pheromone matrix occupies an area of $n \cdot \log^2 n \times (n/log^2 n)$ PEs. The algorithm utilizes the dynamically reconfigurable bus system to speed up the selection process performed within every ant decision. Selection probabilities are calculated based on transformed pheromone and heuristic values:

$$\forall j \in S : f_{ij} := \begin{cases} h & : \quad \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} > t \\ l & : \quad \text{otherwise} \end{cases}$$

This means that the product range of the weighted pheromone and heuristic values is mapped to only two possible values h and l with $h > l > 0$; $t > 0$ defines a threshold. Items $j \in S$ with $f_{ij} = h$ ($f_{ij} = l$) receive a high (low) selection probability. PEs store these transformed values in the form of two bits:

$$h_{ij} := \begin{cases} 1 & : \quad f_{ij} = h \\ 0 & : \quad \text{otherwise} \end{cases} \qquad l_{ij} := \begin{cases} 1 & : \quad f_{ij} = l \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4.1}$$

An item is selected by drawing a random number r from the interval $[0, n_l \cdot l + n_h \cdot h)$ with n_l or n_h denoting the number of PEs in a row, which have the l-bit or h-bit

set to one respectively. If $r \in [(p-1) \cdot l, p \cdot l)$ then the p-th PE with the l-bit equal one is selected. If $r \in [n_l \cdot l + (p-1) \cdot h, n_l \cdot l + p \cdot h)$ then the p-th PE with the h-bit equal one is selected. Using the dynamically reconfigurable bus system, it is demonstrated that n_l and n_h can be calculated in $O(\log^* n)$ time.[1] The selected PE can also be identified in time $O(\log^* n)$. Hence the entire ant decision can be performed in $O(\log^* n)$.

The determination of the updating ants is also shown to require $O(\log^* n)$ time, and the update operation itself can be performed in constant time. To maintain a fast execution speed, it is required that the evaluation of every single ant decision also does not exceed $O(\log^* n)$. This condition is fulfilled, e.g., by TSP, sparse QAP, or SMTTP. Considering the pipelined flow of ants with parallel evaluation and pheromone update, it follows that new solutions can be constructed with an average period of $O(\log^* n)$, and a total number of z solutions can be computed in quasi-linear time $O((z+n) \cdot \log^* n)$ in comparison to $O(z \cdot n^2)$ required by the sequential software implementation of the standard ACO algorithm. In software simulations on QAP instances, the proposed algorithm with modified selection strategy shows a good optimization behavior only slightly worse than ACO with a standard selection procedure.

Based upon the work by the previously mentioned authors, [Janson et al., 2002, 2003] examine various aspects of scheduling multiple ACO algorithms concurrently running on an RMesh. Any of these ACO algorithms occupies a submesh of which the dimensions depend on the respective problem size n. During the algorithm runtime, however, this problem size is successively reduced whenever a pheromone value τ_{ij} exceeds a certain high threshold value (local convergence). A high pheromone value τ_{ij} would cause the ants to almost always select item j in row number i. Therefore, item j is written as a henceforth constant value at place number i of the solution vector ($\pi(i) := j$). Accordingly, row number i and column number j of the pheromone matrix are deleted by appropriately adjusting the size of the submesh. These size reduction steps are repeated whenever a local convergence occurs resulting in a gradually shrinking submesh size. As multiple ACO algorithms are executed in parallel, these repeated size reductions yield a growing amount of free but fragmented computing resources. This fragmentation is alleviated by applying heuristics for the compaction of the occupied area on the RMesh thereby releasing consecutive space for the execution of other ACO algorithms or for the repetition of the same ACO algorithm with different random seeds. The described procedure causes an increased throughput and an improved average solution quality in comparison to executions without dynamic size reduction.

[1] $\log^* n$ is the number of log-operations that must repeatedly be applied to n until the result is less or equal 1.

The described implementations of ACO have been developed for the RMesh model. The computational power of the RMesh, which largely relies on its dynamically reconfigurable bus system, has not yet been met by commercially available reconfigurable architectures. Therefore, the execution on FPGA must be expected to be effectively slower with respect to the attainable asymptotic runtime. However, the RMesh implementation can be regarded as an inspiring source for the ACO hardware realization on FPGA aiming to achieve a considerable speedup in comparison to the sequential software implementation.

4.3 Counter-based ACO (C-ACO)

In this section, the C-ACO algorithm is introduced as a hardware-oriented variant of ACO targeting FPGAs as an implementation platform. A comparison between C-ACO and standard ACO is given in terms of analytical as well as empirical examinations. Subsequently, alternative ways of integrating heuristic information are described. Two different types of parallelizing C-ACO are discussed combined with adapted variants for updating the pheromone matrix. The parallel hardware implementation also supports or even demands alternatives ways of comparing new solutions, sequencing ant decisions, and encoding the pheromone matrix. The respective algorithmic modifications are described and tested in experimental studies. Finally, the mapping of C-ACO onto FPGA is briefly sketched. The contents of this section is based on previous work by the author of this thesis [Scheuermann and Middendorf, 2005, Scheuermann et al., 2005].

4.3.1 Algorithm

In the following, the characteristics of C-ACO (listed in Algorithm 4.1) are explained.

Pheromone Representation: In contrast to standard ACO, pheromone values $\tau_{ij} \geq \tau_{min} > 0$ are represented by integer numbers, which demand less chip resources than floating-point numbers. During initialization all pheromone values τ_{ij} receive the same start value $\tau_{min} + \tau_{init}$ (line 3). The usage of limited integer values instead of floating-point numbers also means that pheromones can take fewer possible values. Inspiring source for this discretization of pheromone values is the RMesh ACO (cf. Section 4.2) restricting pheromones values to only two possible levels with no tremendous degradation in optimization performance.

Local Evaporation: Instead of global evaporation by multiplications with $1-\rho$, pheromones are locally evaporated during the selection process: When an

item j is selected in row i, the respective pheromone value is decremented by $\Delta^\rho = \min\{\tau_{ij} - \tau_{min}, 1\}$, i.e. pheromones behave like decremental hardware counters (lines 14 and 15). Each time an item is selected, the pheromone value is decreased, thereby reducing the attractiveness of the same decision for following ants. Consequently, the exploration of the search space is supported and ants are less likely to converge to a common suboptimal path. Furthermore, the immediate evaporation reduces the selection bias, such that an item selected early in a solution of one ant is more likely to be selected later in the solutions generated by succeeding ants. [Dorigo and Gambardella, 1997b] also investigate a similar form of on-line evaporation (local pheromone update) as outlined in Section 2.5.1. However, the local evaporation process in C-ACO, which results in a subtraction of a constant integer value, is better suited for an FPGA implementation.

Algorithm 4.1 Counter-based ACO (place-item encoding).

```
1: for i := 0 to n − 1 do
2:     for j := 0 to n − 1 do
3:         τ_ij := τ_min + τ_init                    /* initialize pheromone matrix */
4:     end for
5:     U_i := 0                                      /* initialize update counters */
6: end for
7: while stopping condition not met do               /* begin iterations */
8:     for a = 0 to m − 1 do                         /* construct m solution */
9:         S := {0, ..., n − 1}                      /* initialize selection set */
10:        for i := 0 to n − 1 do                    /* n ant decisions per solution */
11:            j := select(i)                        /* probabilistic selection */
12:            S := S \ {j}
13:            π^a(i) := j                           /* insert selected item into solution */
14:            Δ^ρ := min{τ_ij − τ_min, 1}
15:            τ_ij := τ_ij − Δ^ρ                    /* local evaporation */
16:            U_i := U_i + Δ^ρ                      /* increment update counter */
17:        end for
18:    end for
19:    a* := argmin_{a∈{0,...,m−1}} F(π^a); π* := π^{a*}   /* determine best solution */
20:    for i := 0 to n − 1 do
21:        τ_{iπ*(i)} := τ_{iπ*(i)} + U_i            /* pheromone update */
22:        U_i := 0                                  /* reset update counters */
23:    end for
24: end while                                        /* end iterations */
```

Update Counters: Per row of the pheromone matrix there exists an update counter U_i, which accumulates the total amount of pheromone evaporated in that row during an iteration (line 16). At the end of an iteration, when m solutions have been constructed, each update counter holds a value $0 \leq U_i \leq m$. Note that an update counter can be less than m. This happens when ants select such items for which the corresponding pheromone value is already equal to τ_{min}.

Pheromone Update: When the best solution π^* of the current iteration has been determined, the pheromone update is done according to this solution as follows: The pheromone value $\pi^*(i)$ in row i is incremented by the value of the update counter in the same row, i.e. $\tau_{i\pi^*(i)} := \tau_{i\pi^*(i)} + U_i$ (line 21), instead of adding value Δ as stated in Section 2.4.2. Thus, the total amount of pheromone in each row remains constant.

Hitherto the C-ACO algorithm has been described for optimization problems working on a place-item encoded pheromone matrix. Hence, the ants construct solutions while traversing top-down through the rows of the pheromone matrix. For optimization with item-item encoding the algorithm has to be modified as the ants would move arbitrarily between the rows. The index of the next row to be visited is equal to the last item selected. Further modifications are necessary to handle symmetric problem with item-item encoding, e.g., symmetric TSP: If a city j is selected in row i, then pheromone values τ_{ij} and τ_{ji} have to be decremented and the respective update counters U_i and U_j are incremented. Accordingly, the update process has to be adapted. In order to maintain a symmetric pheromone matrix, pheromone values τ_{ij} and τ_{ji} are always increased by the same amount. Since in every row i, two updates are performed, the pheromone values are only increased by at most $\lfloor U_i/2 \rfloor$. If in a row the amount of update is smaller than the update counter value, the remainder is transferred into the next iteration.

4.3.2 Comparison of Standard ACO with C-ACO

The standard ACO and the C-ACO algorithm share common techniques like the encoding of the pheromone matrix, the representation of solutions, the integration of heuristic information as well as the selection process. The main differences concern the limitation of pheromone values to range between a certain minimum τ_{min} and maximum $\tau_{min} + n' \cdot \tau_{init}$ with $n' = n$ for place-item and $n' = n - 1$ for item-item encoded problems. To some degree, bounding pheromone values to an interval resembles the MAX-MIN Ant System [Stützle and Hoos, 1997, Stützle, 1999, Stützle and Hoos, 2000]. Furthermore, pheromones are represented by integer values, which are locally decremented with every selection of an item. The

sequence of local evaporations by multiple ants can be considered as a substitute to global evaporation. Finally, pheromones are not updated by adding a constant or fitness-dependent value. In C-ACO, matrix entries associated with good solutions are updated by adding pheromones collected in update counters thereby maintaining a constant sum of pheromones in each matrix row.

The following sections further compare standard ACO and C-ACO. First the effect of pheromone intensification is examined analytically. This is followed by experimental studies comparing the optimization performance of both algorithms.

4.3.2.1 Analytical Approach

In this section, the pheromone update rule of C-ACO is motivated and its relation to the standard pheromone update rule is discussed. In the standard ACO pheromone update, each pheromone value $\tau_{i\pi^*(i)}$ that belongs to the best solution π^* is increased by a fixed value $\Delta > 0$. In the following, it is examined how much this individual portion of added pheromone affects the decisions of ants in the succeeding iterations and demonstrate that pheromone intensification in C-ACO and standard ACO have a similar stochastic effect. It is supposed that for standard ACO the pheromone values are normalized, such that the sum over all values in a row is $T = \sum_{j=0}^{n-1} \tau_{ij} = 1$. Therefore, the total amount of pheromone evaporated per iteration is equal to ρ. In order to keep T constant, the value of pheromone intensification is equal to the total amount of evaporated pheromone, i.e. $\Delta = \rho$ (if only one ant per iteration is allowed to update). This version of pheromone update for standard ACO has been used by several authors, e.g., [Blum et al., 2001, Guntsch and Middendorf, 2000].

Standard ACO Consider an amount of pheromone Δ, which has been added to τ_{ij} in an arbitrary iteration, and define an iteration counter t starting from this specific iteration $t := 0$. Assuming all items to be contained in the selection set, $p_1 := \Delta/T = \rho$ denotes the probability that in the following iteration $t = 1$, an ant selects item j in row i on account of value Δ. Since the pheromone is evaporated in every iteration, the probability that item j is selected in iteration $t = 2$ is $p_2 = (1 - \rho)\rho = (1 - p_1)p_1$, or more general: $p_t = (1 - p_1)^{t-1}p_1$. Let X_t be a random variable, which expresses the number of ants selecting item j in row i due to update value Δ in iteration t. Obviously, X_t is distributed binomially according to $B(m, p_t)$ with expected value:

$$E(X_t) = mp_t = m(1 - p_1)^{t-1}p_1. \tag{4.2}$$

C-ACO In C-ACO, the intensification received by τ_{ij} is $\Delta = U_i$. Let $p_1 = U_i/T$ (with $T = n \cdot (\tau_{min} + \tau_{init})$) denote the probability, that the first ant in iteration

$t = 1$ selects item j in row i on account of value Δ. With every selection the respective pheromone value τ_{ij} is exactly reduced by 1 such that random variables X_t are distributed hyper-geometrically $H(T, U_i, m)$. Accordingly, the expected value in iteration $t = 1$ is $E(X_1) = mU_i/T$, and in the succeeding iterations $t > 1$:

$$E(X_t) = \frac{m}{T}(U_i - \sum_{t'=1}^{t-1} E(X_{t'})). \tag{4.3}$$

Assuming that τ_{ij} received the maximum update $U_i = m$, then the following equation for the expected value can be proved by induction:

$$E(X_t) = m(1 - \frac{m}{T})^{t-1}\frac{U_i}{T} = m(1 - p_1)^{t-1}p_1, \tag{4.4}$$

which is equal to the corresponding value for the standard ACO (see Equation 4.2).

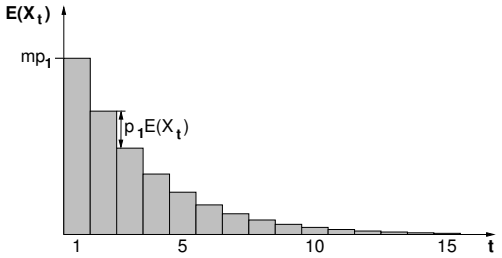


Figure 4.2: Development of expected value $E(X_t)$ for standard ACO and C-ACO. The shaded area is equal to m.

Hence, for standard ACO and C-ACO, an intensification on τ_{ij} causes an average of $m(1 - p_1)^{t-1}p_1$ selections of item j in row i in all iterations $t > 0$. Let random variable $Y_N = \sum_{t=1}^{N} X_t$ denote the frequency of selecting item j in the next N iterations. Then the expected value $E(Y_N) = \sum_{t=1}^{N} E(X_t)$ is determined by

$$E(Y_N) = mp_1 \sum_{t=1}^{N}(1 - p_1)^{t-1} \rightarrow m. \tag{4.5}$$

Hence, in both algorithms, item j is selected approximately m times (see Figure 4.2). Due to these similarities, it is expected that C-ACO shows an optimization behavior comparable to standard ACO.

4.3.2.2 Experimental Studies

In this section, the experimental studies are described, which are conducted to compare the standard ACO with the C-ACO algorithm. The results are obtained

from sequential implementations of both algorithms in software. These examinations allow to compare the optimization performance irrespective of any specific hardware platform.

Experimental Setup The experiment is conducted on a range of TSP and QAP instances. TSP instances are chosen from the TSPLIB benchmark library [Reinelt, 1991]:

symmetric instances	asymmetric instances
`gr48` (48 cities)	`ry48p` (48 cities)
`eil101` (101 cities)	`kro124p` (100 cities)
`d198` (198 cities)	`ftv170` (171 cities)

QAP instances are a selection from the QAPLIB benchmark library [Burkard et al., 1997]:

symmetric instances	asymmetric instances
`wil50` (size 50)	`lipa50a` (size 50)
`sko72` (size 72)	`lipa70a` (size 70)
`tai100a` (size 100)	`tai100b` (size 100)

For both algorithms, common parameter settings are $\alpha = 1$, and $m = 8$ ants per iteration. The results are computed without heuristic ($\beta = 0$), and with the standard heuristic ($\beta = 5$ for TSP and $\beta = 2$ for QAP) as explained in Section 2.4.2. Parameter q_0 modelling the average relation between exploitation and exploration is chosen from $q_0 \in \{0, 0.5, 0.9\}$ (cf. Section 2.3.2). Simulations are run with and without elitism (with elitism means that not only the best ant of the current iteration, but also the best-so-far ant is allowed to update).

Standard ACO is implemented such that $T = 1$ and $\Delta = \rho$ (see Section 4.3.2.1) with $\rho \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$. A minimum pheromone value τ_{min} is introduced to allow for a fair comparison with C-ACO, which is also equipped with a lower pheromone bound. Hence, items are selected with probability

$$p_{ij} = \frac{(\tau_{ij} + \tau_{min})^{\alpha} \eta_{ij}^{\beta}}{\sum_{l \in S} (\tau_{il} + \tau_{min})^{\alpha} \eta_{il}^{\beta}}. \tag{4.6}$$

The minimum pheromone value is determined by $\tau_{min} = \gamma_s \tau_{init} = \gamma_s / n'$ with $\gamma_s \in \{0, 0.001, 0.01, 0.1, 0.5, 1\}$ and $n' = n - 1$ (TSP) or $n' = n$ (QAP). For C-ACO, the initial pheromone value τ_{init} is varied with $\tau_{init} \in \{1, 2, 3, 4, 5, 8, 10, 50, 100, 1000\}$. The minimum pheromone value is set to $\tau_{min} = \lceil \gamma_c \tau_{init} \rceil$ with $\gamma_c \in \{0.001, 0.01, 0.1, 0.5, 1\}$. Duplicate pairs $(\tau_{min}, \tau_{init})$ are removed. Parameters γ_c and γ_s model the relation between the initial and the minimum pheromone value.

The ranges of parameters γ_c and γ_s are chosen to be the same, except that $\gamma_c = 0$ is not allowed since a pheromone value τ_{ij} dropping to its lower bound $\tau_{min} = 0$ would prohibit any further selection of item j in row number i.

Each run is terminated after $t^{max} = 50000 + 1000n$ iterations, which allows all algorithm runs to nearly converge. Note that using a minimum pheromone value $\tau_{min} > 0$, absolute convergence is hard to obtain. This is in contrast to ACO with lower bound $\tau_{min} = 0$, which earlier suppresses a further exploration of the search space.

Solution qualities are measured in relative iterations $t_i = \lfloor \frac{1}{7} i t^{max} \rfloor$, $i = 1, \ldots, 7$. The average solution qualities are calculated as the mean tour lengths (TSP) or mean assignment cost (QAP) over 10 runs with different random seeds. These average solution qualities are transformed into two different types of ranks:

Best ranks: Given a specific algorithm (standard ACO/C-ACO), problem class (TSP/QAP), problem instance, elitism strategy (with/without), and a fixed triple (β, q_0, t_i), the respective best parameter combinations ((ρ, γ_s) for standard ACO and (τ_{init}, γ_c) for C-ACO) are determined. The better algorithm receives rank 1, the other rank 2. Thereafter, the final ranks are calculated as the average over the two elitism strategies and all problem instances of a problem class.

Average ranks: Given a specific algorithm, problem class, problem instance, and a fixed triple (β, q_0, t_i), the solution qualities for all 122 variable parameter combinations (60 for standard ACO, and 62 for C-ACO) are ranked. The final ranks for an algorithm are calculated as the average over all variable parameter combinations and all problem instances of a problem class.

Results Based on Best Ranks In figures 4.3 and 4.4, the columns represent different degrees of exploitation. The bottom/top row shows the results received with/without heuristic. Regarding the TSPLIB instances (see Figure 4.3) the C-ACO algorithm outperforms standard ACO in all but one case ($\beta = 0$, $q_0 = 0.5$). With an overall average rank of 1.30 C-ACO performs considerably better than standard ACO, which achieves a mean rank of only 1.70. The advantage of C-ACO over standard ACO grows with an increasing degree of exploitation. Also all subsequent test results based on best as well as average ranks show that it is advantageous to run C-ACO with a higher and standard ACO with a lower rate of exploitation. Concerning the test runs on QAPLIB instances (see Figure 4.4), C-ACO receives better ranks than standard ACO with only one exception ($\beta = 0$, $q_0 = 0.0$) in relative iteration t_1. Calculating the average ranks for both algorithms the lead of C-ACO (mean rank 1.09) against standard ACO (mean rank 1.91) is even more noticeable than in the experiments for TSP.

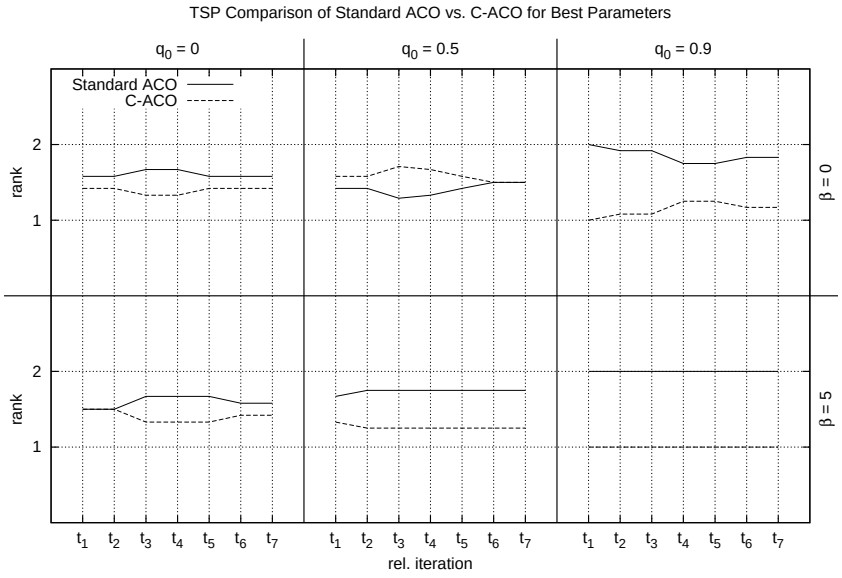


Figure 4.3: Comparison Standard ACO vs. C-ACO on TSP. Ranks for best parameter combinations, averaged over all TSP instances.

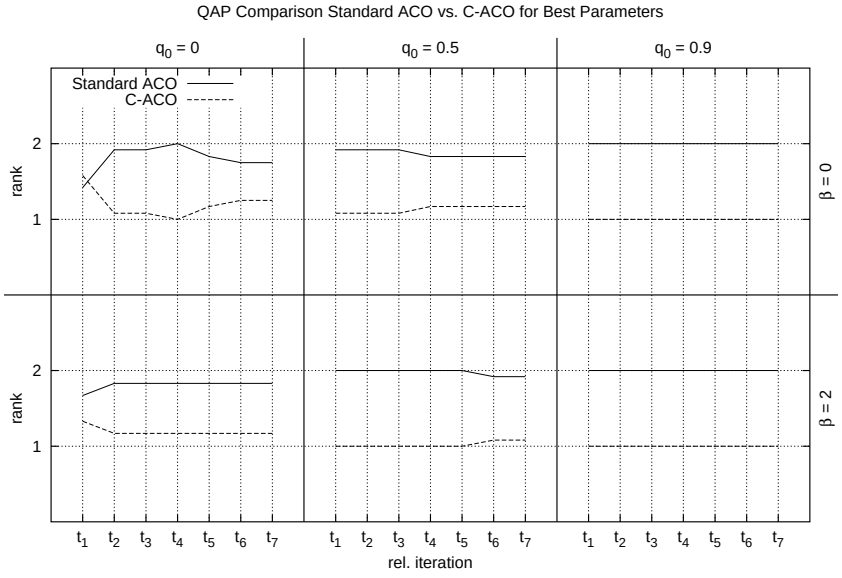


Figure 4.4: Comparison Standard ACO vs. C-ACO on QAP. Ranks for best parameter combinations, averaged over all QAP instances.

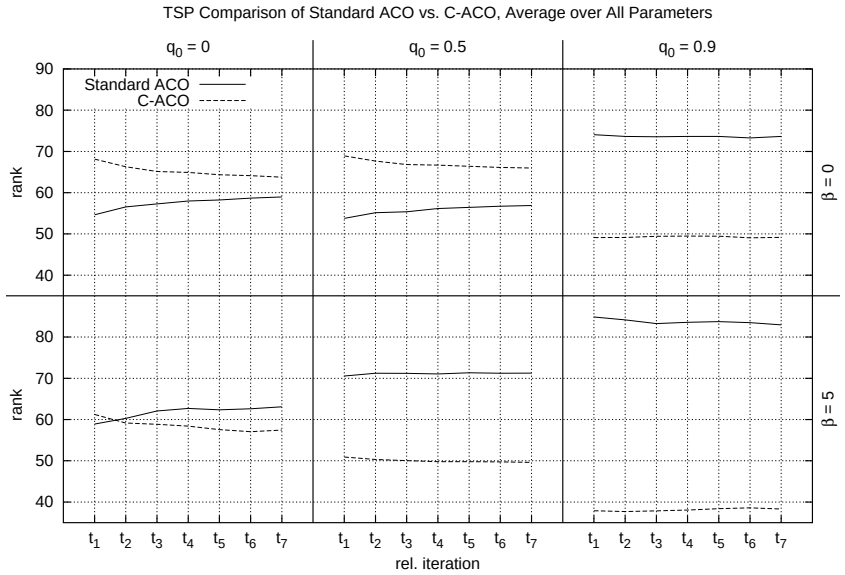


Figure 4.5: Comparison Standard ACO vs. C-ACO on TSP. Ranks averaged over all input parameters and TSP instances.

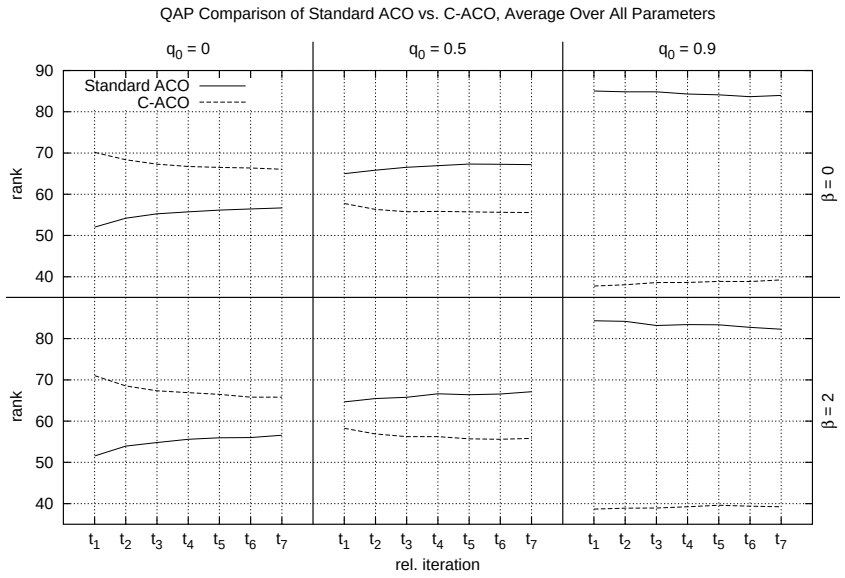


Figure 4.6: Comparison Standard ACO vs. C-ACO on QAP. Ranks averaged over all input parameters and QAP instances.

Results Based on Average Ranks With respect to the TSPLIB instances (see Figure 4.5), standard ACO performs better than C-ACO for $\beta = 0$ and $q_0 \in \{0, 0.5\}$. In runs with heuristic guidance ($\beta = 5$), C-ACO performs consistently better than standard ACO with only one exception (t_1, $q_0 = 0$). Overall the average ranks for TSP are 67.25 (standard ACO) and 54.69 (C-ACO). Figure 4.6 demonstrates that on the selection of QAPLIB instances, standard ACO performs better than C-ACO for $q_0 = 0$. In all remaining cases, however, the ranks for C-ACO are better than for standard ACO. In contrast to TSP, the ranks with heuristic ($\beta = 2$) are very similar to the ranks obtained without heuristic ($\beta = 0$). For QAP, the overall average ranks are 68.43 (standard ACO) and 54.16 (C-ACO).

Overall, the simulation results on both benchmark libraries indicate that on average C-ACO shows a competitive or even superior optimization behavior.

4.3.3 Heuristic Extensions

Optimization performance usually benefits from the integration of heuristic information. However, with respect to the intended hardware realization, including heuristic information into the calculation of selection probabilities (see Equation 2.4) requires multiplication and exponentiation operations with floating-point numbers. To save computation time and FPGA resources, an alternative procedure is proposed, which consists of the following two steps:

1. Heuristic values are weighted by β and then scaled to integer values. In the case of static heuristic information (cf. Section 2.3.2), this can be precomputed on an external processor.

2. The transformed values are combined with pheromone values in a multiplicative or additive way.

Two alternative types of heuristics, the η-heuristics or τ-heuristic, are examined.

4.3.3.1 Algorithmic Modifications

η-Heuristics The so-called η-heuristics integrate heuristic information into the ant decision process by multiplying transformed heuristic values η'_{ij} with pheromone values τ_{ij}. Three variants of η-heuristics are considered (see Figure 4.7):

REALVAL : Heuristic values are processed unchanged: $\eta'_{ij} = \eta^{\beta}_{ij}$.

INTVAL : Heuristic values are transformed into integer numbers $\eta'_{min} \leq \eta'_{ij} \leq \eta'_{max}$, where η'_{ij} is determined by $\eta'_{ij} = \lfloor f(\eta^{\beta}_{ij}) \rfloor \in I\!N$ with

$$f(\eta^{\beta}_{ij}) = \frac{\eta'_{max} - \eta'_{min}}{\eta^{\beta}_{max} - \eta^{\beta}_{min}}(\eta^{\beta}_{ij} - \eta^{\beta}_{min}) + \eta'_{min} \tag{4.7}$$

where η_{min} and η_{max} denote the minimum, respectively the maximum of all values η_{kl} in the heuristic matrix (with $k \neq l$ for item-item encoded optimization problems). Multiplications by integers instead of floating-point values save FPGA resources and computation time.

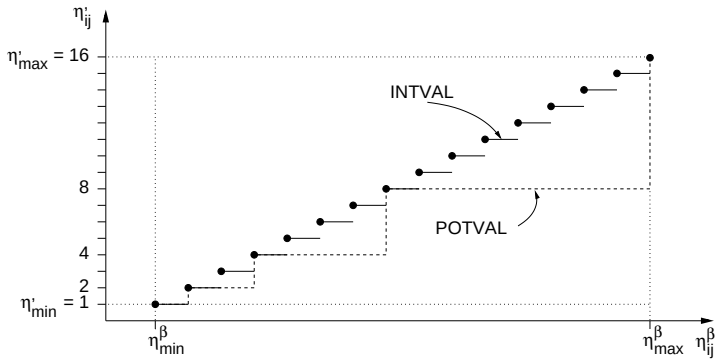


Figure 4.7: Example of scaling standard heuristic information to an interval between $\eta'_{min} = 1$ and $\eta'_{max} = 16$.

POTVAL : Heuristic information is transformed to an interval of integer numbers $\eta'_{min} \leq \eta'_{ij} \leq \eta'_{max}$, where $\eta'_{ij} = 2^k$ with $k = \lfloor \log_2 f(\eta^{\beta}_{ij}) \rfloor$ and f being the same scaling function as in Equation 4.7. Multiplications by numbers $\eta'_{ij} = 2^k$ in hardware can be realized by shifting the respective bit representation of the pheromone value by k digits. The implementation in shift registers demands very little chip resources and allows for a expedited execution.

τ-Heuristic In the τ-heuristic, values η^{β}_{ij} are transformed into integers η'_{ij} in the same way as the η-INTVAL heuristic. The resulting values are then included into the pheromone matrix as lower bound $\tau'_{ij} = \tau_{min} + \eta'_{ij}$. Pheromone values are not allowed to fall below this bound, i.e. $\tau_{ij} \geq \tau'_{ij}$. Initial pheromone values are calculated as $\tau_{ij} := \tau'_{ij} + \tau_{init}$, where $\tau_{init} = \lceil v\bar{\eta} \rceil$ with v a parameter and $\bar{\eta}$ is the average over all values η'_{kl} (with $k \neq l$ for item-item encoding). As opposed to Algorithm 4.1, pheromone values of selected items are decremented by $\lceil w\tau_{init} \rceil$ with w being a parameter. Selection probabilities are computed as

$p_{ij} = \tau_{ij} / \sum_{l \in S} \tau_{il}$. Thus, no multiplications with heuristic values are required at all.

4.3.3.2 Experimental Results

To compare these heuristics, they are tested on the `kro124p` TSPLIB instance using parameter values: $q_0 = 0.3$, $\alpha = 1$, $\beta = 5$, $m = 8$, $\tau_{min} = 1$, $\eta'_{min} = 1$, and $\eta'_{max} = 2^k$ with $k \in \{0, \ldots, 25\}$. The η-heuristics are run with $\tau_{init} \in \{1, 2, 3, 4, 5, 8, 10, 50, 100\}$, the τ-heuristic with $v \in \{1.0, 10.0, 100.0\}$ and $w \in \{0.00001, 0.1, 1.0\}$.

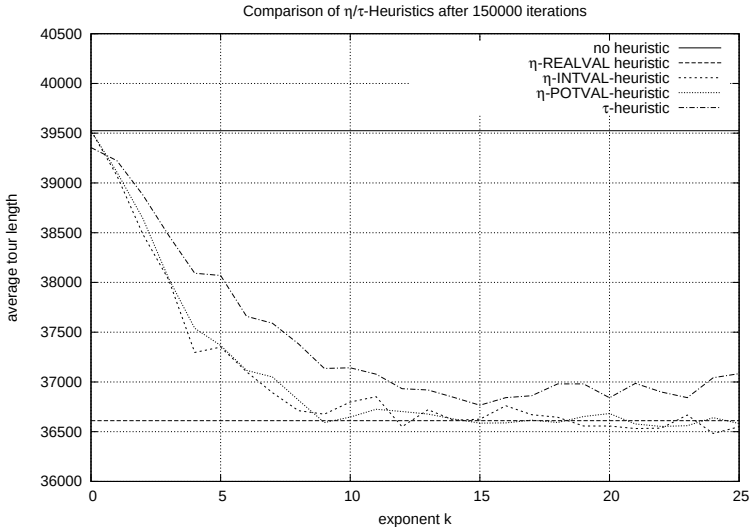


Figure 4.8: Comparison of the η- and τ-heuristics.

Figure 4.8 shows the average solution qualities (20 repetitions) after 150000 iterations. For every value of k, the corresponding solution quality is determined as the minimum average tour length over the input parameter combinations. The chart shows two horizontal lines indicating the best tour length reached by the standard heuristic (REALVAL) on a level of 36610.6 and the best tour length obtained without heuristic on a level of 39524.6, i.e. switching off heuristic support deteriorates solution quality by about 8% with respect to the results gathered during this experiment. With an increasing exponent k the alternative heuristics achieve solution qualities comparable to the standard heuristic. Starting from a value of $\eta'_{max} \geq 2^9$, the INTVAL/POTVAL heuristics reach solution qualities, which are at most 0.66% worse than the best quality of the standard heuristic. This means that multiplications by floating-point numbers can reasonably

be substituted for multiplications with integer numbers with a size of at most $k = 9$ bits. Both heuristics, INTVAL and POTVAL, perform equally well even though POTVAL works on a very restricted range of possible heuristic values. The POTVAL heuristic even allows to efficiently replace large multiplication circuits by significantly smaller and faster shift registers. The τ-heuristic performs worse than the INTVAL and POTVAL heuristics but reaches a minimum tour length only 0.43% above the best tour found by the standard heuristic. In this case, one would have to provide pheromone counters with a lower bound of size $k = 15$ bits. Altogether, the results show that POTVAL represents a good choice in terms of size and execution speed.

4.3.4 Parallel Execution

So far all considered ACO algorithms have been executed sequentially. However, the goal is to speed-up the execution by a parallel implementation of C-ACO in hardware. This section outlines the differences between sequential and parallel execution as well as the experimental studies, which compare these two operation modes.

4.3.4.1 Terminology

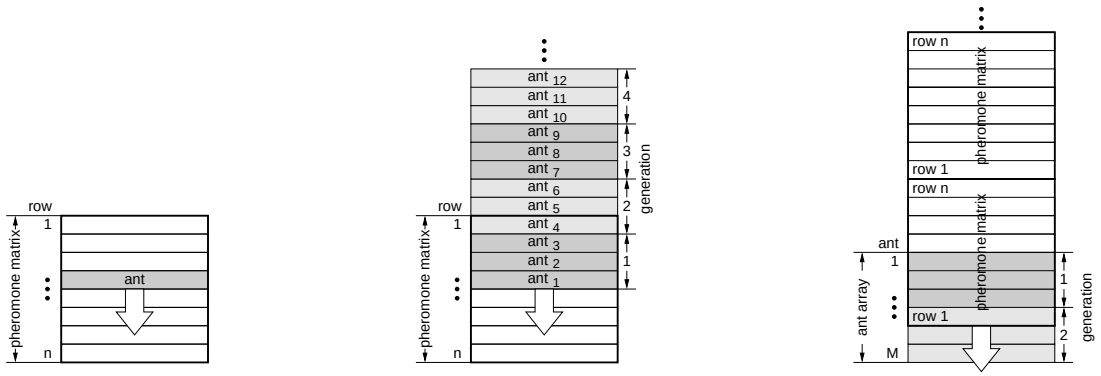
First of all, various terms are introduced, which are used to characterize the algorithmic concepts described in the subsequent sections. Given a certain instant during the algorithm run, all ants being currently involved in solution construction are referred to as *active ants*. The duration needed to allow all active ants to finish their current decisions is denoted as an *algorithm step* (AS). Usually, ants in ACO are grouped into *generations*, and the number of ants within a generation is called the *generation size* m. All ants in a generation directly compete with each other, i.e. only the best (or several best) of the m solutions constructed by this generation is allowed to update the pheromone matrix.

4.3.4.2 Execution Modes

In the following, the characteristics of the *sequential* and the *parallel execution mode* are described. The parallel execution mode refers to the aspects of implementing C-ACO in hardware. Other approaches to parallel software implementations of ACO, as introduced in Section 2.5.2, are not considered.

Sequential Execution Mode Typically, when implementing a sequential ACO algorithm in software, every ant makes one decision per row of the pheromone matrix until it has created a complete solution. The m solutions of a generation

are constructed one after another. This means that at any time of the algorithm run, only one ant is actually active (see Figure 4.9a). Accordingly, the number of algorithm steps required in the sequential case can be computed as $AS = t \cdot m \cdot n$ with t denoting the number of completed generations. The best (or several best) of the m solutions constructed by a generation may update the pheromone matrix. This update is finished before the next generation starts constructing new solutions.



(a) sequential (b) parallel, PA (c) parallel, PP

Figure 4.9: Process of solution construction in the sequential and parallel case after 4 algorithm steps. In this example: $n = 8$, $m = 3$, and $M = 6$.

Parallel Execution Mode An implementation in hardware aims to speed up the execution of C-ACO by parallelizing certain operations and by exploiting the pipelining capabilities of hardware algorithms. Two variants of parallel execution are considered, which can be distinguished by their type of pipelining: *piped ants* and *piped pheromone*.

Piped ants (PA): The pheromone matrix is mapped statically onto the chip area. A continuous flow of ants is piped top-down through this circuitry in a systolic fashion as visualized in Figure 4.9b. This flow of ants is sub-divided into generations of m consecutive ants. As in the sequential case, an ant makes a decision for its current row of the pheromone matrix. However, pipelining of ants allows not just one but up to n ants to be active at a time.

Piped pheromone (PP): An array of M ants is accommodated statically onto the chip area. The pheromone matrix is piped perpetually top-down through this ant array (see Figure 4.9c), which is sub-divided into M/m generations

of m consecutive ants. An ant makes a decision for every row of the pheromone matrix passing by. This type of pipelining allows up to M ants to be active at a time. To allow for a fair comparison between these two types of pipelining, it is assumed that henceforth the size of the ant array M is equal to the problem size n.

After the initial n algorithm steps, the systolic pipelining of ants or pheromones, respectively, yields one complete solution with every following algorithm step. Therefore, the number of algorithm steps in the parallel execution mode can be described as $AS = t \cdot m + n - 1$.

4.3.4.3 Spectrum of Parallelization

The hardware implementation of C-ACO allows to speed-up the execution by parallelizing a wide range of tasks, which are traditionally executed in a sequential manner. From a top-level point of view, these tasks include the construction, the evaluation, and the comparison of new solutions, the update of the pheromone matrix, and the check on the fulfillment of certain stopping condition.

Solution Construction Every solution construction by an ant a consists of n decisions D_{ai}, one decision per row i of the pheromone matrix with $i \in \{1, \ldots, n\}$ (see Figure 4.10). Considering an individual ant, all these decisions have to be computed sequentially as every decision directly depends on its preceding decisions. For each decision, the ant has to compute the following sequence of steps:

Calculation of selection probabilities: The selection probabilities p_{ij} are computed for all items in selection set S:

$$\forall j \in S : \; p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta'_{ij}}{\sum_{l \in S} \tau_{il}^{\alpha} \cdot \eta'_{il}}, \tag{4.8}$$

with η'_{ij} representing one of the four possible alternative heuristics[2] proposed in Section 4.3.3. In order to create the probability distribution, it is not necessary to calculate the denominator in Equation 4.8 and to perform the division. It is sufficient to calculate the prefix sum

$$\forall j \in S : \; pr_{ij} = \sum_{l \in S, l \leq j} \tau_{il}^{\alpha} \cdot \eta'_{il} \tag{4.9}$$

over the numerators of the as yet unselected items. A hardware implementation on FPGA requires $O(\log n)$ time to compute this prefix sum.

[2] In the case of the τ-heuristic, no multiplications are afforded as heuristic information is integrated into the pheromone matrix.

Generation of random number: An integer random number r_i is drawn from the interval $r_i \in \{0, \ldots, B_i - 1\}$, where the upper bound B_i is determined by the maximum prefix sum $B_i = \max_{l \in S} pr_{il}$ in row i. Random numbers on FPGAs can be generated by an appropriate choice of a parallel pseudo random number generator [Ackermann et al., 2001], which can produce a random number in $O(1)$ time.

Selection: Random number r_i is compared with the prefix sums calculated in row i. If $pr_{i,j-1} \leq r_i < pr_{ij}$, then item $j \in S$ is selected and removed from S. The parallel implementation in hardware allows the selection to be computed in $O(1)$ time.

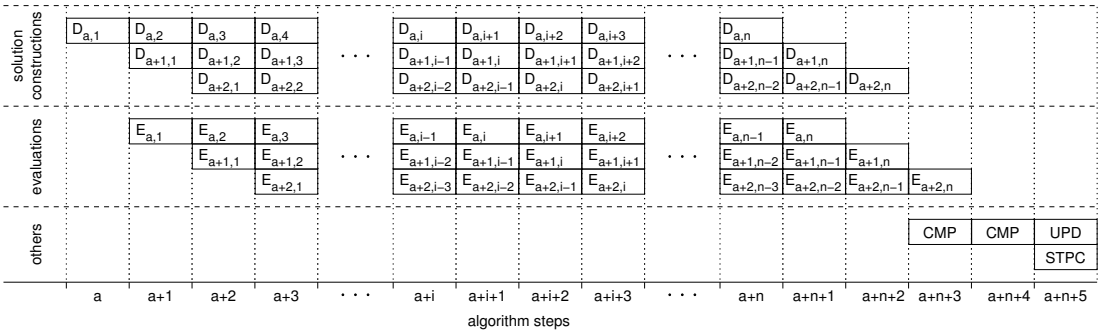


Figure 4.10: Example of a schedule of tasks associated with the computations for one generation of $m = 3$ ants starting from ant number a. Ants are piped systolically through a pheromone matrix consisting of n rows. Matrix rows are updated all at once. Key: D = decision, E = evaluation, CMP = comparison, UPD = update, and STPC = stopping condition.

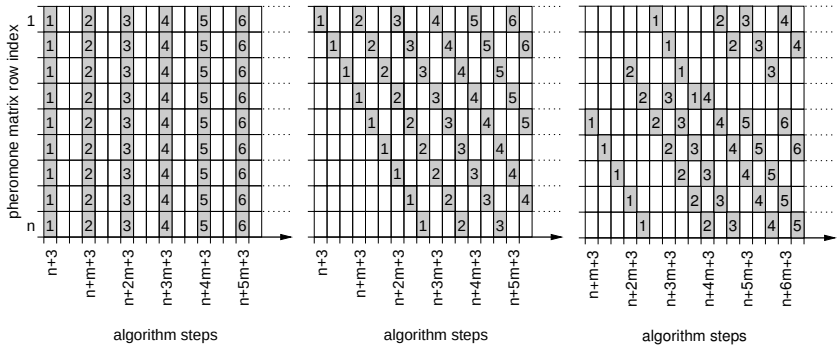
The time to compute a decision amounts to $O(\log n)$, such that a complete solution is constructed in $O(n \cdot \log n)$. As ants are piped through the pheromone matrix (or vice versa), multiple solutions can be constructed concurrently. With respect to the number of simultaneously active ants, the hardware implementation attains a pipelining degree of n, and solution constructions can be started with a period of $O(\log n)$.

Evaluation Every solution constructed by an ant is assigned a solution quality. If the optimization problem permits, this evaluation can be split into a sequence of n evaluations of individual ant decisions. This online evaluation (cf. Section 2.3.2) means that it is possible to calculate how much an individual decision D_{ai} contributes to the final solution quality. In Figure 4.10, the corresponding

process is denoted by E_{ai}, which can be started as soon as the associated decision D_{ai} has finished. In order to maintain a rapid flow of solution constructions and evaluations with a period of $O(\log n)$, it is essential that every evaluation E_{ai} of a single decision can also be computed in at most $O(\log n)$ time.

Consider TSP as an example: If in decision D_{ai} city j is selected, the respective evaluation simply consists of adding distance d_{ij} to the length of the path along all as yet visited cities. Obviously, this operation demands only $O(1)$ time. The same is true for many scheduling problems. With respect to QAP, the sequential evaluation of a solution in software usually requires a computation time of $O(n^2)$, whereas in hardware, this process can be divided into a sequence of n evaluation steps (one per assignment) each requiring $O(\log n)$ time.

Comparison The qualities of new solutions are compared with each other to determine the best solution of the current generation. The systolic construction of new solutions also allows comparisons to be done on the fly: The latest solution is compared with the best of the previously constructed solutions of the same generation. After $m - 1$ comparisons the generational-best solution is known. Every comparison takes $O(1)$ time and does therefore not affect the critical path.



(a) ATONCE (b) PIPED (c) STATIONARY

Figure 4.11: Update operations as they are propagated through the pheromone matrix for all three update modes. Update operations are represented by shaded boxes with numbers. Equal numbers indicate that the corresponding update operations belong to the same update. In this example, the initial six updates are shown with $n = 9$, $m = 3$, and $M = 9$.

Pheromone Matrix Update The pheromone matrix is modified according to an *update solution*, which might be the best solution of a generation or the hitherto overall best solution found (elite solution). With respect to C-ACO, those

entries in the pheromone matrix, which are associated with the update solution, are incremented by the value stored in the update counters. The hardware implementation allows to parallelize the requested operations. Depending on the respective pipelining type (cf. Section 4.3.4.2), three different update modes can be distinguished:

ATONCE: In the case that ants are piped through the pheromone matrix, the circuitry for each matrix cell memorizes the decision made by each passing ant, i.e. whether the cell is selected or not. After an update solution has been determined, the most intuitive approach is to update the rows of the pheromone matrix all at once (see Figure 4.11a). This is done by broadcasting to every matrix cell the index of the ant, which constructed the update solution. This index is used as an address to restore the decision made for the update solution. This at-once-update can be executed very efficiently in $O(1)$ time. However, the matrix is updated while ants are engaged in solution construction. With respect to a solution being currently constructed, this means that one part of this solution may possibly be produced based on older pheromone information than another part of the very same solution. It must be examined how much impact this online pheromone update has on optimization performance.

PIPED: This update mode also targets the parallel implementation with piped ants. In contrast to ATONCE, the index of the updating ant is not broadcast to the whole pheromone matrix, but piped top-down through the matrix rows within the flow of ants constructing solutions (see Figure 4.11b). As before, this index is used as an address to restore the decision made for the update solution. As opposed to ATONCE, update steps do not interfere with solution construction. Like in the sequential execution mode, an individual solution is entirely constructed based on pheromone information of the same age. The operations within a single row require $O(1)$ time and can be executed in parallel to solution construction.

STATIONARY: When pheromone matrices are piped through a statically allocated ant array, decisions are stored within the ant circuitry. This means that each ant can memorize the solution it constructed. After the update solution of a generation has been determined, the solution index is broadcast to the ants thereby identifying the ant, which constructed and stored this solution. Immediately, this ant starts updating the pheromone values. An update operation is performed with every matrix row received by the stationary positioned update ant. Figure 4.11c shows that the propagation of update operations very much resembles the PIPED update process. However, updates do not start consistently from matrix row number 1. The

first row to be updated rather depends on the position of the update ant within the ant array. Consequently, updates can interfere with solution constructions. One part of a solution may be created based on older pheromone information than the other part. This interference can be avoided if the update is delayed until row number 1 of the pheromone matrix is passing the update ant. As before, the update operations within a single row require $O(1)$ time.

To save computation time, all three update modes are executed in parallel to ant decisions. However, read-write-conflicts must be avoided while accessing pheromone values.

Stopping Conditions Stopping conditions are also checked on the fly and do not influence the critical path. The most common stopping condition is to terminate the optimization after a predefined maximum number of generations executed. In hardware this stopping condition can easily be realized by a generation counter.

4.3.4.4 Experimental Results

In experimental studies, the sequential and the parallel implementation of C-ACO are compared with each other. For both variants, also in the parallel case, the results are obtained from simulations in software. The parallel variants comprise both types of pipelining (PA and PP). The parallel PA variant is tested with the ATONCE and the PIPED update mode, whereas the algorithm with PP is combined with the STATIONARY update mode. The latter one is also run with delayed update, such that every matrix update starts from row number 1. As problem instance, QAP instance `tai100a` with a problem size of $n = 100$ is used with constant parameters $q_0 = 0.3$, $\alpha = 1$, $\beta = 0$, $\tau_{min} = 1$, $\tau_{init} = 10$, and including elitism. The parallel variant with PP is executed with an ant array of size $M = n$ and chooses the generation size from the range $m \in \{1, 2, 4, 5, 10, 20, 25, 50, 100\}$ such that m is a divider of M. All other combinations are started with $m \in \{1, \ldots, n\}$. For every algorithm run, the best assignment cost is recorded, of which the average is calculated over 20 repetitions with different random seeds.

For a varying generation size m, Figure 4.12 shows the average assignment cost of the QAP obtained after 12 million solutions have been generated. Note that the comparison based on solutions generated does not account for differences in execution speed. In particular, the potential speedup of the parallel over the sequential implementation is disregarded at this stage. This experiment

solely aims to examine the impact of different execution and update strategies on optimization performance.

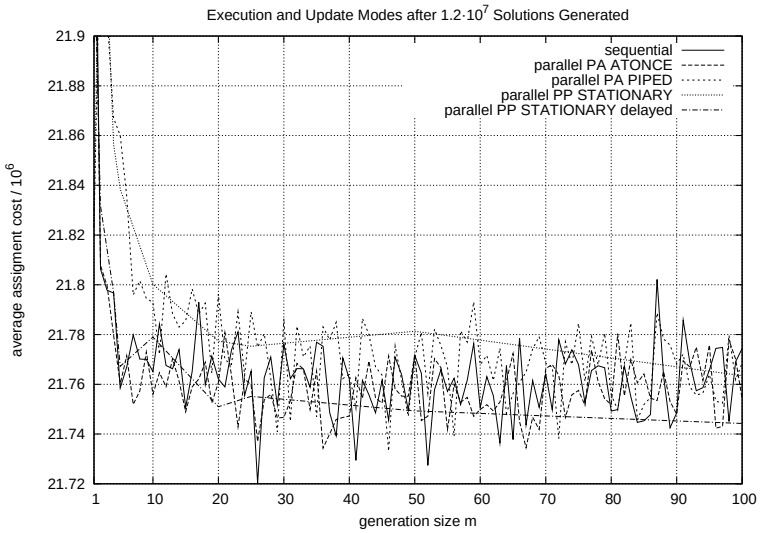


Figure 4.12: Comparison of execution and update modes for varying generation size m after 12 million solutions generated.

Considering the sequential, the parallel PA ATONCE, and the delayed PP STATIONARY execution and update modes, solution qualities improve with a growing number of ants per generation. However, starting from a generation size of approximately $m \geq 8$, the solution qualities do not change much. On average, these three execution and update modes perform equally well. Seemingly, the concurrent pheromone updates by PA ATONCE, which interfere with the continuous parallel solution construction, has no measurable impact on optimization performance.

The qualities for the parallel PA PIPED ad PP STATIONARY variants improve with an increasing generation size till about $m = 20$. These two execution and update modes perform slightly worse than the sequential, parallel PA ATONCE, and the delayed PP STATIONARY variants. Obviously, the PP STATIONARY variant benefits from the introduction of delayed updates.

Figure 4.13 shows the development of the average solution qualities during the complete optimization run executed with the generation size, which yields the respective best average assignment cost after 12 million solutions generated. The curves show no significant differences between the five alternative execution and update modes. The results suggest that concurrent pheromone update during solution construction of active ants does not notably affect optimization performance. Furthermore, both types of pipelining, piped ants and piped pheromone,

are equally well suited. However, piped ants should preferably be combined with ATONCE update and piped pheromone works best with delayed STATIONARY update. All subsequent examinations are based on piped ants and ATONCE update mode.

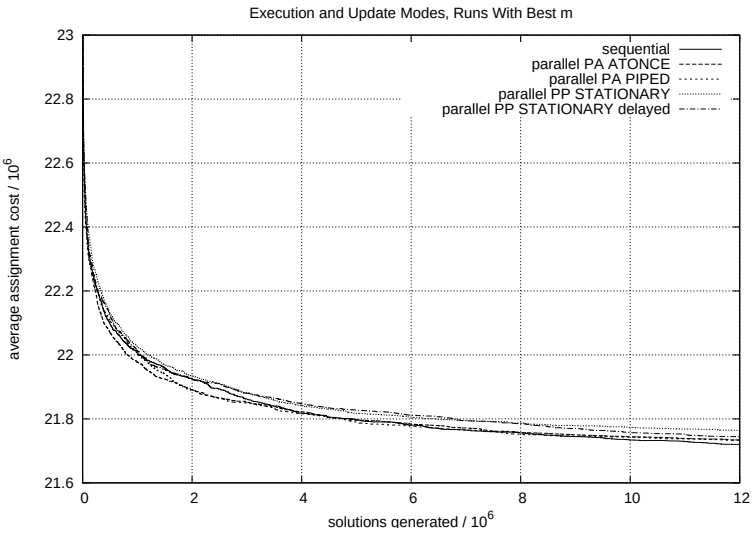


Figure 4.13: Comparison of execution and update modes during complete optimization runs. Average assignment cost for respective best generation size m.

4.3.5 Comparison Modes

Commonly in ACO, multiple recently constructed solutions are compared with each other to determine the best solution, which is then allowed to update the pheromone matrix. In practice, the methods applied to identify updating solutions and the frequency of performing matrix updates may differ. As one possible way of classification, it is proposed to distinguish between the *generational* and the *non-generational* comparison mode.

4.3.5.1 Generational Comparison Mode

The most widely used approach is to wait for all m ants of a generation to complete their solution constructions. Thereafter, these m new solutions are compared to determine the best solution, which is used for updating the pheromone matrix. This generational update mode has also been employed in all previous examinations.

4.3.5.2 Non-generational Comparison Mode

In the parallel hardware implementation, the pipelining of ants through the pheromone matrix produces new solutions with a period of $O(\log n)$. This steady flow of new solutions suggests to compare solutions online instead of waiting for a sequence of solutions to complete. This means that each time a solution has been constructed, it is immediately compared with others. If a best solution is found, this solution is as before allowed to perform a matrix update. Since this approach differs from the standard generational way of comparing solutions, it is referred to as the non-generational comparison mode.

A kind of non-generational comparison is also proposed in [Maniezzo, 1999] (cf. Section 2.5.1). A further variant of non-generational comparison for RMesh ACO (cf. Section 4.2) is presented in [Merkle and Middendorf, 2002a]: A new solution is allowed to update the pheromone matrix if it is better than the $m' - 1$ best solutions generated by the $m - 1$ preceding ants, where m' and m with $m' \leq m$ are external algorithm parameters. However, the algorithm allows updates to the pheromone matrix while other ants are involved in solution constructions. Therefore, it is possible that the $m - 1$ preceding ants have worked on older pheromone information and might have worse chances to find good solutions. Hence, the authors propose a modification of the described comparison method: Whenever an ant has created a new solution, it has to wait until the succeeding $\lfloor (m-1)/2 \rfloor$ ants have also finished their solution construction. This ant is then allowed to update the pheromone matrix, if its solution is better than the $m' - 1$ best solutions of the $\lceil (m-1)/2 \rceil$ preceding ants and the $\lfloor (m-1)/2 \rfloor$ succeeding ants.

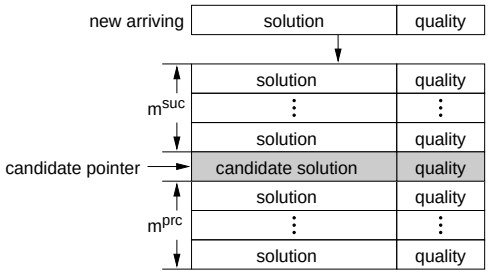


Figure 4.14: Comparison queue filled completely with m^{prc} preceding solutions, m^{suc} succeeding solutions, and a candidate solution. New arriving solutions are inserted at the top.

The comparison method introduced in this thesis can be considered as a generalization of the two previously described approaches. Whenever a new solution has been constructed, it is inserted into a so-called *comparison queue*, which pro-

vides storage for multiple solution vectors and their associated solution qualities (see Figure 4.14). The queue maintains a pointer to a candidate solution, which is allowed to update the pheromone matrix if it is the winner in a comparison with its m^{prc} preceding and m^{suc} succeeding solutions. Two types of comparison are distinguished:

BEST: The candidate solution is allowed to update the pheromone matrix if it is at least as good as the best of the m^{prc} preceding and m^{suc} succeeding solutions.

AVERAGE: The candidate solution is allowed to update the pheromone matrix if it is at least as good as the average over all m^{prc} preceding and m^{suc} succeeding solutions.

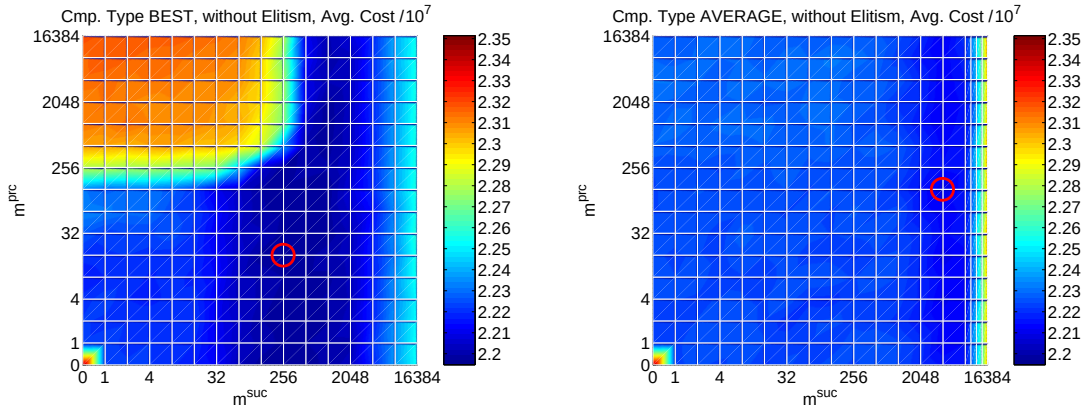
In the very special case of $m^{prc} = m^{suc} = 0$, the candidate solution always updates the pheromone matrix. All matrix updates are performed in parallel with solution constructions by other ants. After the initialization of the algorithm, the first comparison takes place after $m^{prc} + m^{suc} + 1$ solutions have been created, i.e. as soon as the comparison queue is filled entirely. The queue is organized according to a FIFO-policy, i.e. after every comparison (and matrix update if applicable), the oldest solution is removed from the queue before the next arriving solution is inserted.

4.3.5.3 Experimental Results

Experimental studies are conducted to support the design of an appropriate comparison procedure for the parallel implementation of C-ACO. These studies are sub-divided into two stages: First, aspects related to the new non-generational comparison mode are examined followed by a competition between the generational and the non-generational comparison mode.

Properties of the Non-generational Update Mode The first series of experiments examines the effect of the two possible comparison types (BEST and AVERAGE) and determines a good choice of parameters m^{suc} and m^{prc}. All experiments are run on the QAPLIB instance `tai100a` with $n = 100$ and the following common parameter values: $q_0 = 0.3$, $\alpha = 1$, $\beta = 0$, $\tau_{min} = 1$, and $\tau_{init} = 10$. The number of successors $m^{suc} = 2^k$ and the number of predecessors $m^{prc} = 2^k$ are scaled logarithmically with $k \in \{0, \ldots, 14\}$. Additionally, the case of a comparison queue without successors ($m^{suc} = 0$) or without predecessors ($m^{prc} = 0$) is considered. Both comparison types, are tested with and without elitism. For every algorithm run, the best assignment cost is recorded, of which the average is calculated over 20 repetitions with different random seeds. Figures

4.15 and 4.16 show the results after 1.2 million solutions per simulation run have been generated with the attained average assignment cost represented by colors. The axes for m^{suc} and m^{prc} are scaled logarithmically. To be able to also include the results for $m^{suc} = 0$ (and $m^{prc} = 0$), the respective solution qualities are plotted at the actual position of $m^{suc} = 0.5$ (and $m^{prc} = 0.5$).



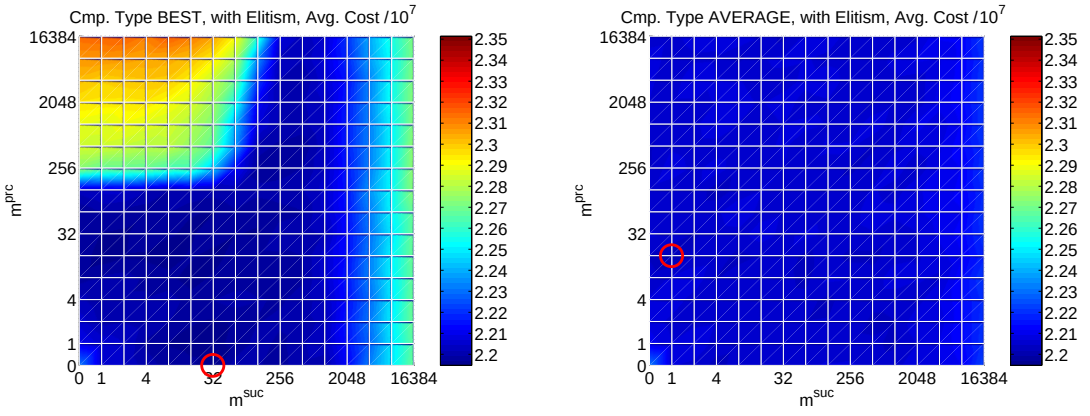
(a) Comparison type: BEST

(b) Comparison type: AVERAGE

Figure 4.15: Average assignment cost after 1.2 million solutions generated for a varying number of succeeding and preceding ants. QAPLIB instance `tai100a`, without elitism.

Figure 4.15 shows the results obtained with no consideration of an elitism update strategy. Applying comparison type BEST (see Figure 4.15a), it is unfavorable to choose a high value of predecessors m^{prc}, because the candidate solution has to compete against a large number of preceding solutions. This leads to very infrequent updates of the pheromone matrix and too many solutions are generated on the very same distribution of pheromone information. Very high values of succeeding solutions m^{suc} also worsen the attainable solution quality as the pheromone matrix is updated with a comparatively old candidate solution, which was generated m^{suc} algorithm steps ago. However, the incline of assignment cost regarded for high values of m^{suc} is less strong than for equally high values of m^{prc}. In general it is unprofitable to choose a very high number of successors and/or predecessors as it takes an increasingly long time to fill the comparison queue completely, such that the overall first matrix update is performed very late. This is especially true for short simulation runs. In the case of $m^{suc} = m^{prc} = 0$, candidate solutions (also very bad ones) are always allowed to update the pheromone matrix. This lack of competition with other solutions results in the worst solution quality measured. With respect to the parameter combinations tested,

the overall best choice (indicated by the red circle) is situated at $m^{suc} = 256$ and $m^{prc} = 16$ with an average solution quality of 21969330. Starting from about 300000 solution constructions, good solution qualities are always located in the interval $m^{suc} \in [128, 1024]$ in combination with $m^{prc} \in [0, 256]$. It can be observed that in very early phases of the algorithm run, this favorable region is located closer to the origin with $m^{suc}, m^{prc} \in [0, 32]$ (except for the origin itself).



(a) Comparison type: BEST

(b) Comparison type: AVERAGE

Figure 4.16: Average assignment cost after 1.2 million solutions generated for a varying number of succeeding and preceding ants. QAPLIB instance `tai100a`, with elitism.

Considering the results for comparison type AVERAGE (see Figure 4.15b), the choice of a very high parameter value m^{prc} has an less significant impact on solution quality than observed for comparison type BEST. The candidate solution quality is compared with the average quality of the other solutions in the queue. Therefore, a high number of predecessors still allows for relatively frequent matrix updates, as opposed to comparison type BEST. As far as the number of successors is concerned, comparison type AVERAGE shows a behavior very similar to its counterpart BEST. However, the incline in the average assignment cost is very rapid. Therefore, three additional test points are inserted in the interval $m^{suc} \in [8192, 16384]$. The deterioration in solution quality is also due to the updates based on comparatively old candidate solutions. The origin with $m^{suc} = m^{prc} = 0$ is as before related with the worst average solution quality. With respect to the parameter combinations considered, the overall best setting is situated at $m^{suc} = 4096$ and $m^{prc} = 128$ with an average solution quality of 22097890. A region of good solution qualities can be identified for $m^{suc} \in [2048, 8192]$ irrespective of the choice of the number of predecessors. As above,

in very early phases of the algorithm run, this favorable region is situated closer to the origin.

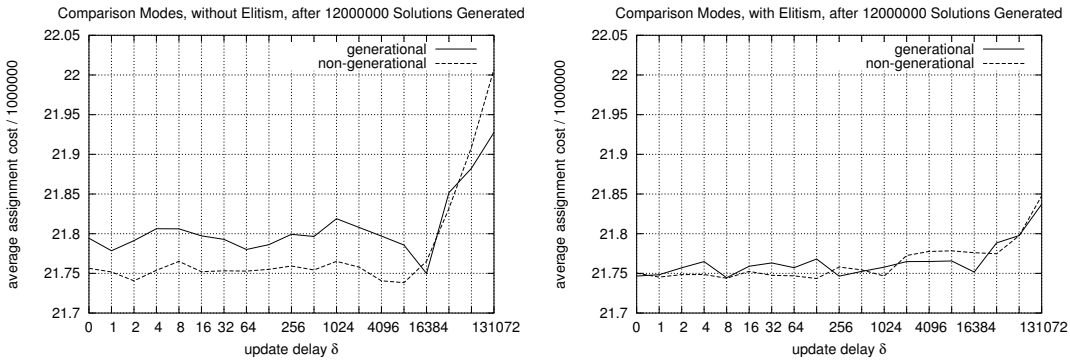
The results obtained from the corresponding simulation runs with an additional pheromone update by the elite solution is visualized in Figure 4.16. In the case of comparison type BEST (see Figure 4.16a), the minimum average assignment cost of 21947490 is measured for $m^{suc} = 32$ and $m^{prc} = 0$. For comparison type AVERAGE, the minimum is located at $m^{suc} = 1$ and $m^{prc} = 16$ with a solution quality of 22017570 (see Figure 4.16b). It is noteworthy that for comparison type AVERAGE with elitism, the solution qualities are very insensitively to changes of m^{suc} and m^{prc}. The deterioration in solution quality for a high number of successors is measurable, though less immense than in Figure 4.15b. For both comparison types, BEST and AVERAGE, the favorable regions of good combinations of successors and predecessors are located closer to the origin than observed in the simulations without elitism. In contrast to the algorithm runs without elitism, the favorable regions constantly stay at about the same location. The solution qualities for $m^{suc} = m^{prc} = 0$ are better than in the experiments without elitism due to the updates with the potentially good elite solution.

Overall it can be summarized that the solution qualities obtained for comparison type BEST are better than for AVERAGE and that, as expected, the inclusion of elitism further improves optimization performance. In accordance with the results of Merkle and Middendorf [2002a], it is beneficial to compare the candidate solution not only with its predecessors (as done in Maniezzo [1999]) but also to compare it with succeeding solutions. The presented results show that it might even be better to completely abandon a comparison with preceding solutions (i.e. $m^{prc} = 0$), thereby reducing the amount of resources needed for the implementation of the comparison queue.

Generational versus Non-generational Comparison The next experiment aims to examine the differences between the generational and the non-generational comparison mode. As outlined in Section 4.3.4.3, the comparison of solutions is embedded in a sequence of tasks including solution construction, evaluation, comparison and pheromone matrix update. The attainable execution speed strongly depends on how these tasks are synchronized. The synchronization is determined by the execution time of these tasks and with which period they can be started. To efficiently exploit the pipelining capabilities, it is essential that the periods are in the same order of magnitude. However, depending on the respective execution times, it may happen that the final task of updating the pheromone matrix is delayed by some time interval, which is referred to as the *update delay* δ. Such an update delay may occur for instance, when solution constructions are followed by some local optimization or repair mechanisms, or when the sequence of evaluating

ant decisions is preceded/succeeded by some initial or concluding calculations, or when comparison is not done on the fly but at once after an entire generation. In the following, it is abstracted from the a specific reason for the occurrence of update delays. It is simply assumed that after every solution construction all subsequent tasks are delayed by δ algorithm steps.

The experiment is also run on QAPLIB instance `tai100a` with $n = 100$ and the following common parameter values: $q_0 = 0.3$, $\alpha = 1$, $\beta = 0$, $\tau_{min} = 1$, and $\tau_{init} = 10$. The update delay, applied to both comparison modes, is scaled logarithmically within the range between $0 \leq \delta \leq 2^{17}$. Experiments are run with and without elitism. In algorithm runs with elitism, the generation size $m = 36$ for the generational comparison mode is a favorable choice with regard to the experimental results in Section 4.3.4.4. As for the non-generational comparison mode, the preferable settings are retrieved from the previous experiment, i.e. comparison type BEST is combined with $m^{prc} = 0$ and $m^{suc} = 32$. Algorithm runs without elitism use a generation size of $m = 98$ (determined in preliminary experiments) for the generational comparison mode, whereas comparison type BEST is combined with $m^{prc} = 16$ and $m^{suc} = 256$ with respect to the non-generational comparison mode. The best assignment cost is recorded, whereof the average is calculated over 40 repetitions with different random seeds.



(a) without elitism (b) with elitism

Figure 4.17: Average assignment cost after 12 million solutions generated for a varying update delay δ. QAPLIB instance `tai100a`.

Depending on the specific update delay, the average assignment cost after 12 million solutions generated is plotted in Figure 4.17. Regarding the results without elitism (cf. Figure 4.17a), initially, both comparison types react very insensitive to a change of the update delay δ. Also common to both comparison modes is that for very high values of δ, solution qualities degrade considerably.

Obviously, a moderate update delay does not notably affect optimization performance, whereas a high delay may cause too many solutions to be constructed based on obsolete pheromone information. It is noteworthy that the generational comparison mode can even profit from a delayed pheromone update with update delays around $\delta = 16384$. This behavior can be observed during the complete algorithm run. However, this phenomenon cannot be recorded for the non-generational comparison mode. Almost consistently, the non-generational comparison mode performs better than the generational counterpart. The results obtained from the experiments with elitism also show a decline in optimization performance as the update delay takes very high values (see Figure 4.17b). However, the deterioration is less immense than in the experiments without elitism. The differences between the solution qualities of both comparison modes are not significant.

In the following, the influence of an update delay is further examined by observing the development of the *pheromone entropy*. The pheromone entropy in row i shall be defined as $H_i^\tau = -\frac{1}{n} \cdot \sum_{j=0}^{j=n-1} \log p_{ij}$, with p_{ij} denoting the relative frequency of pheromone value τ_{ij} in the i-th row of the pheromone matrix. The average pheromone entropy $H^\tau = \frac{1}{n} \cdot \sum_{i=0}^{i=n-1} H_i^\tau$ measures the uniformity/diversity of the distribution of pheromone values within the pheromone matrix. The term entropy is introduced an explained in [Shannon, 1948].

Figure 4.18 shows the average pheromone entropy associated with the algorithm runs in Figure 4.17a with generational comparison mode, without elitism and for a selection of update delays between $\delta = 0$ and $\delta = 131072$. All executions start with an average pheromone entropy of $H^\tau = 0$ as all pheromone values are initialized to $\tau_{ij} = \tau_{min} + \tau_{init}$. In the case of immediate updates ($\delta = 0$) the overall highest diversity of pheromone values is attained after 1276 algorithm steps. This is followed by a decline of pheromone entropy until a level of about 0.08 is reached, which corresponds to a converged pheromone matrix, i.e. in every matrix row, all freely movable pheromones are concentrated on one entry while the others remain on the minimum pheromone value $\tau_{ij} = \tau_{min}$. With an increasing update delay $\delta > 0$ more and more pheromones are accumulated in the update counters before the first update takes place. Correspondingly, during the first 2048 algorithm steps, the maximum attainable pheromone entropy becomes increasingly restricted. The curves for update delays $\delta \geq 2048$ show that all disposable pheromones have entirely been transferred into the update counters prior to the first update. This means that temporarily, all pheromone entries fall to their minimum values τ_{min} and the average entropy decreases to 0. Starting with the first update, algorithm runs with $\delta \geq 1024$ are subject to a second rise in entropy. Regarding executions with $\delta \geq 8192$, it can be observed that pheromone matrices do not converge in a steady but rather wave-like form with a wave length in the proximity of δ algorithm steps.

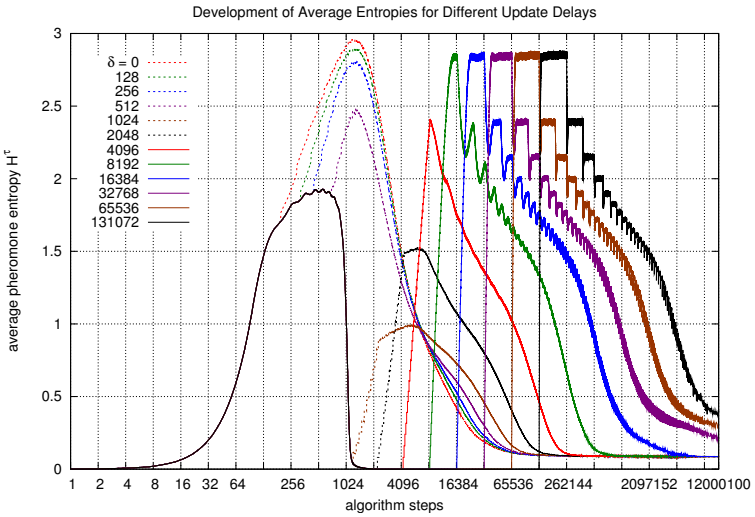


Figure 4.18: Development of average pheromone entropy for generational comparison mode, without elitism, and for different update delays δ.

Seemingly, entropy momentarily falls whenever the algorithm has been able to find a new good solution, but the following updates based on comparatively old solutions disperse the distribution of pheromone values leading to an intermittent increase in pheromone entropy. Presumably, update delays cause a dispersion of pheromone values with two different effects. On the one hand, the diversity of pheromone values is maintained for a longer time, which may potentially avoid getting trapped in unfavorable local optima. On the other hand, the perpetual update with old solutions slows down the convergence of pheromone values and hinders the rapid improvement of solution qualities. Figure 4.18 shows that only algorithm runs with $\delta \leq 16384$ can finally converge (all others remain on an almost constant level for about the last 3000000 algorithm steps). It is also an update delay of $\delta = 16384$ which yields the best solution qualities in Figure 4.17a. Given a maximum number of 12 million solutions generated, apparently, this update delay offers the right balance between prevention of local optima and fast convergence speed.

This experiment demonstrates that the preference for one of the two comparison modes strongly depends on the specific update delay. An appropriately chosen update delay $\delta \leq 16384$ does not critically affect or can, under some circumstances, even improve optimization performance. This is an important result, as the designer may choose from a wider range of circuit instantiations. For a given hardware algorithm, usually, there exist alternative instantiations, for which one has to consider trade-offs between specific design objectives, e.g.,

size and speed: A small-sized but relatively slow module can very often be substituted for a larger but faster circuit and vice-versa. Especially, in the case of scarce resources, one may prefer a small but slower module if the induced update delay is acceptable.

All subsequent experiments are made for $\delta = 0$ and including elitism. For this parameter setting, both comparison modes perform equally well. It is decided to continue the following examinations for the generational variant, as in practice this comparison mode is widest spread.

4.3.6 Decision Sequencing

As a constructive meta-heuristic, ACO creates new solutions by a sequence of local ant decisions. Considering optimization problems with a place-item encoding of the pheromone matrix, it is common practice to let all ants make the first decision for row number 0 and to always step one row further with every subsequent decision. This decision sequence $DS = (0, \ldots, n-1)$ is equal to all ants and does not change during the algorithm runtime. Too much importance is given to the first decisions. At the beginning of the solution construction, an ant can choose from a wider range of available items than at later stages of the construction process. An item j may be favorable at the beginning for it has a high pheromone value τ_{ij} for a small row index i. Therefore, it is very likely to be selected at an early stage of the construction process and to be removed from selection set S, even if a later placement would actually be more favorable.

[Merkle et al., 2002] observe that during the process of solution construction, the strong interdependencies between ant decisions lead to a so-called *selection bias*. This means that at later stages of the construction process, the distribution of pheromone values in a row of the pheromone matrix does not properly reflect the distribution of selection probabilities. Hence, the order in which an ant makes decisions plays an important role and can influence the degree of selection bias. In the following, three different types of decision sequences are distinguished.

4.3.6.1 Fixed Decision Sequence

The *fixed decision sequence* (FDS) refers to the standard decision sequence as introduced before. All ants make the first decision in the first row and then proceed top-down through the pheromone matrix. A variant to this kind of sequencing for QAP is presented in [Dorigo et al., 1996]. As outlined in Section 2.4.2.2, the authors suggest to sort the facilities in non-increasing order of their flow potentials. This means that the at the beginning of the algorithm, the rows of the pheromone matrix are permuted such that facilities with high flow potentials

are assigned earlier in the process of solution construction. Subsequently, this variant is called the *sorted fixed decision sequence* (S-FDS).

4.3.6.2 Random Decision Sequence

Using the *random decision sequence* (RDS), an ant steps through the pheromone matrix following a randomly generated permutation of the row indices $i \in \{1, \ldots, n\}$. For every solution to be constructed, a new random sequence is created. After multiple solution constructions according to RDS, on average the decisions in all matrix rows become equally important. A drawback of this approach is that ants proceed arbitrarily through the rows of the pheromone matrix, which in turn prohibits pipelining. Therefore, RDS is not suited for a hardware implementation.

Applying ACO to the Single Machine Total Weighted Deviation Problem (SMTWDP), [Merkle and Middendorf, 2001c] compare FDS with RDS. It is shown that RDS yields better solutions than FDS, even though the optimization problem considered only allows to integrate heuristic information when scheduling jobs in a forward order. Further improvements are obtained by using two types of ants simultaneously: One group of ants complies to FDS (with heuristic), the other proceeds according RDS (without heuristic).

4.3.6.3 Cyclic Decision Sequence

The *cyclic decision sequence* (CDS) allows for a parallel implementation in hardware but diverts from the principle of a continuous flow of ants being piped top-down through the pheromone matrix as described in Section 4.3.4.2. The idea is to evenly distribute a generation of m ants over n rows of the pheromone matrix. Here it is supposed that ant a is placed in row number a with $a \in \{0, \ldots, m-1\}$. After every algorithm step, all ants are shifted cyclically one row forward. This means that for each ant a the decision sequence can be described as $DS_a = (r_0^{(a)}, \ldots, r_{n-1}^{(a)})$ where the i-th decision of ant a is made in row $r_i^{(a)} = (a + i) \bmod n$ with $i \in \{0, \ldots, n-1\}$. CDS intends to combine the advantages of the fixed and the random decision sequence: Pipelining of ants is maintained, and on average the decisions in all matrix rows are equally important. One further advantage of the new decision sequence is that it allows an easy use of heuristic information for many types of problems. For RDS the use of heuristic information in such problems is more difficult (see [Merkle et al., 2002]). Using this type of decision sequencing, it requires $AS = m \cdot t$ algorithm steps to complete t generations. Note that the parallel implementation of the non-generational update mode cannot be combined with CDS. The continuous process of comparison and pheromone update demands FDS instead.

4.3.6.4 Experimental Results

Software simulations are conducted to compare the fixed decision sequence (FDS, S-FDS) and the cyclic decision sequence with regard to the parallel hardware implementation of C-ACO. As the random decision sequence is not suited for systolic algorithms, it is tested for the sequential implementation. Experiments are run on the QAP instance `tai100a` with problem size $n = 100$ with the following constant parameters: $q_0 = 0.3$, $\alpha = 1$, $\beta = 0$, $\tau_{min} = 1$, $\tau_{init} = 10$, and including elitism. The number of ants per generation is varied with $m \in \{1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. Note that for CDS, only with $m = n$ the maximum possible pipelining degree can be achieved. For every algorithm run, the best assignment cost is recorded, of which the average is calculated over 20 repetitions with different random seeds.

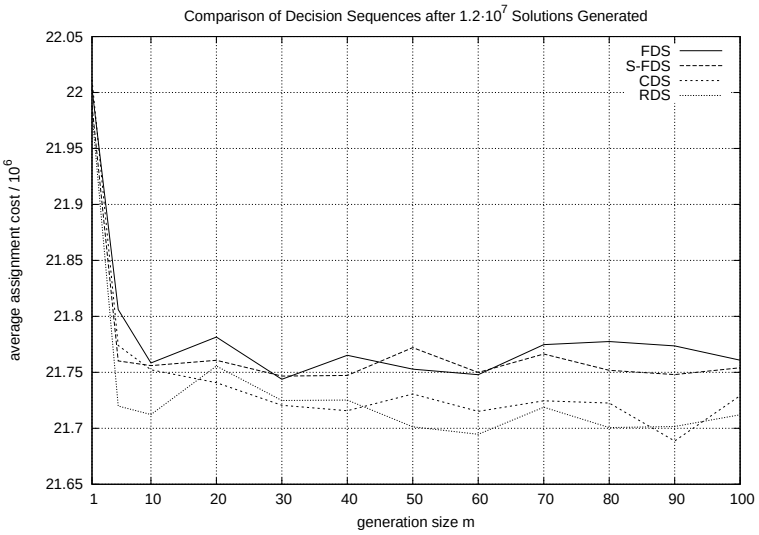


Figure 4.19: Comparison of decision sequences after 12 million solutions generated.

For all decision sequences under test, Figure 4.19 visualizes the average assignment cost for a varying number of m ants per generation. As in the previous experiment, the curves show the solution qualities obtained after 12 million solutions generated. The differences between the results for the fixed decision sequences, FDS and S-FDS, are only marginal. The results obtained for CDS and RDS are, as expected, better than for the fixed decision sequences. The optimization performance for both, CDS and RDS, is approximately the same.

Figure 4.20 visualizes the development of the average solution qualities during the complete optimization executed with the generation size, which per-

formed best after 12 million solutions generated. The results show that both CDS and RDS outperform the fixed decision sequences. FDS and S-FDS differ only marginally and the curves for CDS and RDS are almost congruent. The experimental studies clearly demonstrate that sequencing ant decisions in a cyclic order can compete with the random decision sequence. CDS should therefore be preferred for the parallel implementation in hardware, as it allows for a pipelined flow of ants through the pheromone matrix, but also for the sequential implementation in software, CDS represents an attractive substitute to RDS.

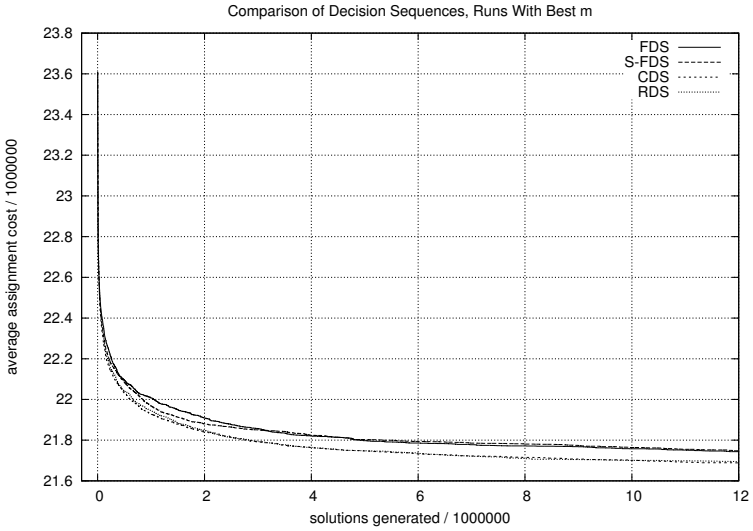


Figure 4.20: Comparison of decision sequences during complete run with best m.

4.3.7 Matrix Encoding

Depending on the optimization problem, the pheromone matrix may be encoded in different ways as outlined in Section 2.4.2. Most common are the place-item encoding (PIE) and the item-item encoding (IIE). Generally, PIE is preferred if it is essential to model to which place in the solution vector an item should be assigned to. IIE is more suited to express which item is selected next after the previously selected item. Therefore, IIE is commonly chosen for TSP, which is used as an example for the subsequent explanations.

If an ant, currently located in row number i, randomly selects city j, then the ant moves to row number j to make its next decision according to the IIE encoding. Hence, ants can move arbitrarily between the rows of the pheromone matrix.

This is a major drawback with respect to the parallel hardware implementation as IIE prevents a continuous pipelining of ants. Alternatively, the pheromones could be place-item encoded, though this is commonly not done for TSP as it is not meaningful to which place in the solution vector a city is assigned to. In fact there exist n different solution vectors, which all map to the same tour, because shifting the sequence of cities does not change the tour.

4.3.7.1 Place-Item-Item Encoding

A new matrix encoding is proposed, which unifies the advantages of PIE and IIE. Therefore, this approach is named *place-item-item* encoding (PIIE). The pheromone matrix is represented in the same manner as IIE. However, ants make decisions while they are row-wise moving through the matrix as if it was place-item encoded. Considering TSP, in every row i, an ant selects a successor j for city i. For its next decision, the ant does not move to row j but selects a successor for city $i + 1$ as shown in Figure 4.21.

(a) item-item encoding (b) place-item-item encoding

Figure 4.21: The same tour along cities (A,D,E,B,C) constructed by using IIE (a) and PIIE (b). Numbers on arcs indicate the order in which cities have been selected.

Starting from single cites, PIIE produces many small paths, which are linked together until at last a complete tour is constructed. During the construction process the creation of cycles must be avoided (only the last decision may produce a Hamiltonian cycle comprising all n cities). Using IIE, a tour is constructed by repeatedly appending cities to the as yet constructed path. Therefore, it is sufficient to exclude selected cities from selection set S to prevent the creation of cycles.

For PIIE, it is proposed to complement the selection set by two *tabu-tables*. One of them is called the *tabu-to-table* $\mathcal{T}_{to}$. For an ant located in city i, $\mathcal{T}_{to}(i)$ returns the city to which the ant must not move to in order to avoid the creation of a cycle, the ant may only select a city from $S \setminus \{\mathcal{T}_{to}(i)\}$. Consider the example

shown in Figure 4.22: The current location i of the ant is always the last city on a path starting from city $h = \mathcal{T}_{to}(i)$. It is sufficient to declare city h as a tabu, since all successors of h have already been excluded from S. It is supposed that the ant randomly selects city j. A selected city is always the first city of another path. If these two paths are linked together, the tabu-to-table has to be updated for city k, which is the last city of the appended path. In order to determine this city, a further tabu-table is maintained. Given a city j, the *tabu-from-table* provides city $k = \mathcal{T}_{from}(j)$ from which the ant is not allowed to come from. Algorithm 4.2 lists all steps required to update both tabu-tables when an ant decides to select city j in row number i. The update of the tabu-tables can be computed very efficiently in $O(1)$ time.

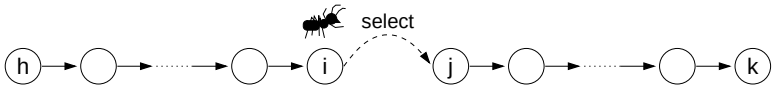


Figure 4.22: An ant located in city i may not return to one of the preceding cities of its path. City j is selected, which is the first city on a different path.

Algorithm 4.2 Update of tabu-tables.

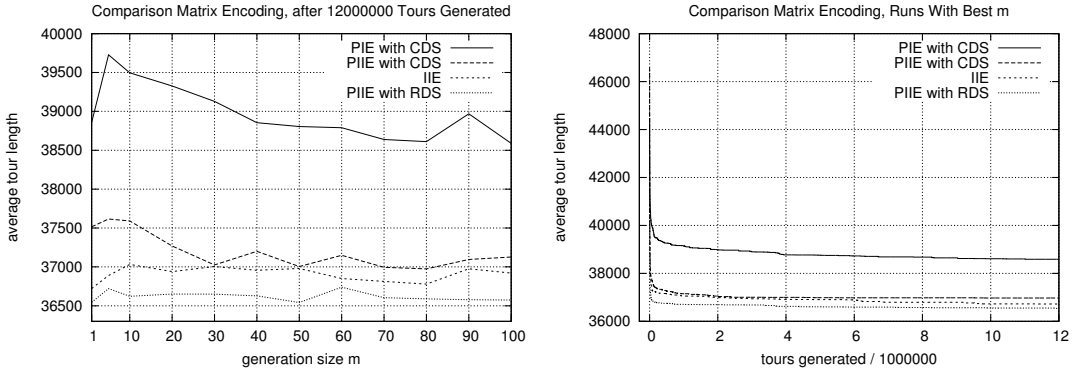
1: $h := \mathcal{T}_{to}(i)$
2: $k := \mathcal{T}_{from}(j)$
3: $\mathcal{T}_{to}(k) := h$
4: $\mathcal{T}_{from}(h) := k$

4.3.7.2 Experimental Results

In experimental studies, the three different types of matrix encoding, PIE, IIE, and PIIE, are compared with each other. All simulations are performed with C-ACO on the TSPLIB instance `kro124p` with $n = 100$ cities and with the following common parameters: $q_0 = 0.3$, $\alpha = 1$, $\beta = 5$, $\tau_{min} = 1$, $\tau_{init} = 10$, and including elitism.

PIE and PIIE are tested in combination with CDS and RDS, whereas implicitly IIE defines its own decision sequence and is therefore compared independently with its opponents. Algorithms with CDS are run in parallel execution mode while RDS and IIE can only be executed sequentially. The number of ants per generation is varied with $m \in \{1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. As before, the parallel execution of CDS can only benefit from the maximum degree of pipelining with the generation size set to $m = n$. For every algorithm run, the

best tour length is recorded, of which the average is calculated over 20 repetitions with different random seeds.



(a) After 12 million tours generated.

(b) Complete run for best m.

Figure 4.23: Comparison of different types of matrix encoding.

Figure 4.23a shows the best tour lengths obtained for a varying number of ants per generation after 12 million tours constructed. It can be seen that IIE and PIIE far outperform the PIE variant. The solution qualities for IIE are somewhat better than for PIIE with CDS. For both, IIE and PIIE, solutions are encoded in the same fashion, the only difference lies in the order in which the ants step through the pheromone matrix. Presumably, the alternating sequence of ant decisions inhibited in IIE can reduce the selection bias and gives this type of matrix encoding a slight advantage over PIIE with CDS. However, IIE is not suitable for a hardware implementation as the ants move arbitrarily between the rows of the pheromone matrix. Using PIIE with RDS in place of CDS consistently achieves better results than IIE, since ants make their decisions in an arbitrary order, thereby reducing the degree of biased selections. Whereas PIIE benefits from sequencing ant decisions in a random order, it is noteworthy that in combination with RDS the performance of PIE deteriorates drastically. The respective average tour length is lifted from an average value of 38982.9 (PIE with CDS) to a level around 145899 (not visualized).

Choosing the generation size, which achieves the best solution qualities after 12 million tours generated, Figure 4.23b shows the tour lengths during the whole algorithm run. The results for IIE and PIIE with CDS differ only slightly. Both of them far outperform the PIE variant, whereas PIIE with RDS is throughout the best choice. The experiments conducted suggest that PIIE with CDS is an appropriate alternative type of encoding the pheromone matrix, which supports pipelining of ants such that traditionally item-item encoded optimization prob-

lems are also suited for a parallel implementation in hardware. Furthermore, PIIE in combination with randomly sequenced ant decisions may offer an attractive substitute for IIE with regard to sequential ACO algorithms.

4.3.8 Mapping C-ACO onto FPGA

This section sketches the mapping of C-ACO onto an FPGA assuming a statically allocated pheromone matrix with piped ants (PA). Thus, the index of ant a and its selection set are propagated top-down through the cells of the matrix circuitry. Such an individual cell is depicted in Figure 4.24.

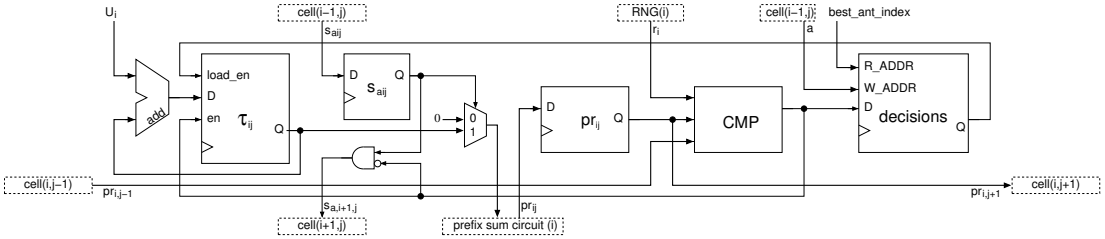


Figure 4.24: Circuit of cell (i, j) of the pheromone matrix. For reasons of clarity, signals for clock, reset and control are omitted.

The current selection set of ant a in row i is denoted by $S_{ai} = \{s_{a,i,0}, \ldots, s_{a,i,n-1}\}$ with $s_{aij} = 1$ if j has not yet been selected by ant a located in row i, else $s_{aij} = 0$. Pheromone value τ_{ij} is stored in a loadable decremental counter with minimum value τ_{min}. In this design, heuristic knowledge is disregarded, but it can be extended by one of the alternative heuristics introduced in in Section 4.3.3. Pheromones are weighted by $\alpha := 1$ (common choice in standard ACO, see Section 2.2) to avoid exponentiations. Therefore, the calculation of selection probabilities (cf. Equation 2.4) can be simplified to $p_{ij} = \tau_{ij} / \sum_{h \in S} \tau_{ih}$. In order to select an item, it is not necessary to calculate the denominator in this equation and to perform the division. It is sufficient to calculate the prefix sum over the numerators of the yet unselected items (as explained in Section 4.3.4.3). Hence, in every row of the pheromone matrix, the design contains a circuit to calculate the prefix sum $pr_{ij} = \sum_{l=0}^{j} s_{ail}\tau_{il}$. Which item is selected next is decided probabilistically by a random number r_i (see, e.g., [Ackermann et al., 2001] for a random number generator on FPGA). All decisions d_{aij} in cell (i, j) are stored in a decision memory with $d_{aij} = 1$ if $pr_{i,j-1} \leq r_i < pr_{ij}$ (CMP), else $d_{aij} = 0$. After a generation of ants have constructed their solutions and the best solution has been determined, the index a^* of the best ant `best_ant_index` is broadcast to all cells (assuming ATONCE update). The received index is used as an address to read

the decision d_{a^*ij} made by this ant. If $d_{a^*ij} = 1$ then the load enable signal of the pheromone counter is asserted and the contents U_i of the update counter is added to pheromone value τ_{ij}.

4.4 Population-based ACO (P-ACO)

In the previous section, the C-ACO algorithm is presented as a hardware-oriented variant of ACO. As explained, this variant exhibits many favorable properties supporting an efficient FPGA implementation. The Population-based ACO (P-ACO) algorithm used in this section is introduced in [Guntsch and Middendorf, 2002b]. Originally designed for the sequential implementation in software and targeting dynamic optimization problems [Guntsch and Middendorf, 2002a], P-ACO also offers certain features making it very attractive for a realization on FPGAs. Similar to C-ACO, the population-based variant allows to represent pheromone values by integer numbers, which are confined to an interval between a specific minimum and maximum. It is also common that both approaches avoid an explicit evaporation of pheromone values as known from standard ACO. C-ACO substitutes global evaporation for a local evaporation step with every selection of an item, whereas P-ACO applies a concept of positive and negative pheromone updates, which correspond to updates of a so-called population. Giving birth to the name of this algorithm, the population keeps and maintains a set of previously generated good solutions.

In the following, the P-ACO algorithm is briefly introduced. Several modifications to the original algorithm are explained, which allow to accelerate the execution and facilitate the mapping into hardware. The P-ACO algorithm is actually implemented and tested on FPGA. Furthermore, a new kind of heuristic is presented as a straightforward extension to the existing hardware implementation. The following contents concerning P-ACO is based on previous work by the author of this thesis [Diessel et al., 2002, Scheuermann et al., 2003, 2004a,b].

4.4.1 Introduction

The main idea of the P-ACO algorithm proposed by [Guntsch and Middendorf, 2002b] is to manage a *population* $P = \{\pi_0, \ldots, \pi_{l-1}\}$ as a memory of l good solutions constructed in the preceding iterations with $0 \leq l \leq k$. *Population size* k denotes the maximum number of solutions the population can hold.

From this population the corresponding pheromone matrix can be derived as illustrated in Figure 4.25. In this example, it is assumed that the pheromone matrix is encoded in a place-item fashion and that the population comprises the maximum number of k solutions. Within this population, each solution π_h is

shown as a column, and $\pi_h(i)$ describes the item stored at place number i of solution h. Initially, every cell in the pheromone matrix is assigned a minimum pheromone value $\tau_{min} > 0$, which represents a lower bound. Afterwards for every item $\pi_h(i) = j$ in the population, the respective pheromone value τ_{ij} is incremented by a constant amount of pheromone $\Delta_P > 0$. Hence, an individual pheromone value τ_{ij} can be calculated as follows:

$$\forall\ i, j \in \{0, \ldots, n-1\} : \tau_{ij} = \tau_{min} + \zeta_{ij} \cdot \Delta_P, \tag{4.10}$$

with ζ_{ij} denoting the number of occurrences of item j at the i-th place of all solutions in the population, i.e. $\zeta_{ij} = |\{\pi \in P : \pi(i) = j\}|$. It follows that for each entry in the pheromone matrix, there exist only $k + 1$ possible pheromone values between τ_{min} and $\tau_{max} = \tau_{min} + k \cdot \Delta_P$.

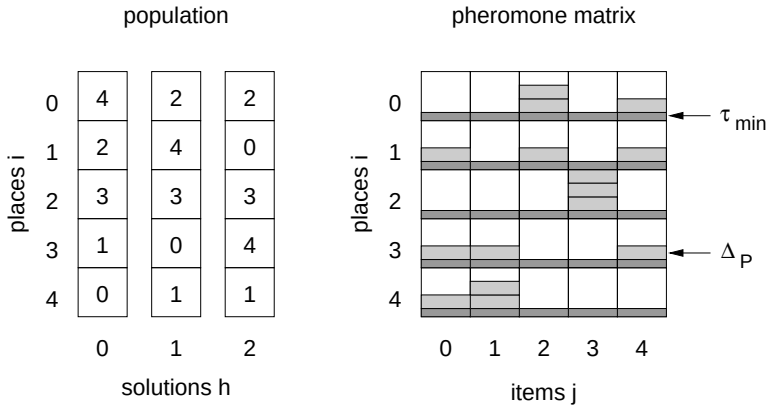


Figure 4.25: Example of a population containing a maximum number of $k = 3$ solutions and the corresponding pheromone matrix.

As in standard ACO, ant decisions executed by the P-ACO algorithm are based upon the information stored in the pheromone matrix and optional heuristic knowledge. The difference to standard ACO lies in the pheromone update procedure, which completely abandons the concept of global evaporation. The update of the pheromone matrix consists of two steps tightly coupled to a synchronous *population update*:

Positive update: Whenever the best solution π^+ of the current iteration has been determined, this solution is inserted into the population $P := P \cup \{\pi^+\}$. This process is reflected by an equivalent update of all pheromone values associated with π^+:

$$\forall\ i \in \{0, \ldots, n-1\} : \tau_{i\pi^+(i)} := \tau_{i\pi^+(i)} + \Delta_P. \tag{4.11}$$

Negative update: If after a positive update the population contains more than k solutions, then the oldest solution π^- is removed from the population $P := P \setminus \{\pi^-\}$. This removal corresponds to an equivalent pheromone update:

$$\forall\, i \in \{0, \ldots, n-1\} : \tau_{i\pi^-(i)} := \tau_{i\pi^-(i)} - \Delta_P. \tag{4.12}$$

Initially, the population is empty ($P = \emptyset$) and is subsequently filled up during the first iterations until it reaches its maximum capacity k. Afterwards the oldest solution is always removed whenever a new solution enters the population. This means that the population is organized as a FIFO-queue of size k. Other schemes for deciding which solutions should enter/leave the population are discussed in [Guntsch and Middendorf, 2002a]. As the population behaves like a queue, it is henceforth proposed to use a so-called *queue matrix* $Q = [q_{ih}]$ as an alternative notation with $q_{ih} = \pi_h(i)$. Therefore, the number of occurrences of item j in the i-th row of the queue matrix can also be described as $\zeta_{ij} = |\{h : q_{ih} = j\}|$.

Empirical studies show that P-ACO is a competitive approach in comparison to other ACO algorithms and that for population size k small values $1 \leq k \leq 8$ are sufficient for all test instances regarded in [Guntsch and Middendorf, 2002b]. Therefore, k can be considered to be a small problem-independent constant.

4.4.2 Modifications

The modifications proposed in this section allow to speed up the execution of the P-ACO algorithm and to profitably support the implementation on FPGA. The first modification concerns the data types used to represent pheromone values. Focusing on the usual implementation in software, [Guntsch and Middendorf, 2002b] only require the minimum pheromone value and the update values to be positive real numbers, i.e. $\tau_{min} \in \mathbb{R}^+$ and $\Delta_P \in \mathbb{R}^+$. In their experimental studies, the authors further demand that the sum over all pheromone values in a row is equal to 1. However, scaling τ_{min} and Δ_P with a constant ratio Δ_P/τ_{min} produces an identical probability distribution as per Equation 2.4 and therefore has no influence on the decision process. With respect to an expedited and resource-efficient implementation on FPGA, it is proposed to set $\tau_{min} := 1$ and $\Delta_P := \lceil f_P \cdot n/k \rceil$ such that both parameters are integer values. Experimental studies suggest that on average $f_P = 10$ is a favorable choice.

A further modification avails of the fact that in P-ACO the pheromone matrix represents redundant information for it can completely be derived from the information stored inside the population. Consequently, a method is proposed, which solely relies on population data, but produces ant decisions equivalent to those made by means of pheromone information. At this stage it is supposed that all ant decisions are only based upon pheromone information, thus the selection

probabilities can be simply expressed as:

$$\forall j \in S : p_{ij} = \frac{\tau_{ij}}{\sum_{l \in S} \tau_{il}}. \tag{4.13}$$

The integration of heuristic knowledge is discussed later in Section 4.4.5. Also note that, as practised in conjunction with C-ACO, exponent α is set to 1 so as to avoid exponentiations. If the probability distribution in Equation 4.13 were to be built based upon the information stored in the pheromone matrix, every single ant decision would afford $O(n)$ time on a sequential processor, or $O(\log n)$ time when choosing a parallel implementation (cf. Section 4.3).

With a closer look on the distribution of pheromone values in an arbitrary row i of the pheromone matrix, there exist at most k values different from τ_{min} and every single entry has one of at most $k+1$ possible pheromone values. This information, however, is implicitly stored inside the population as pointed up by Equation 4.10. It is an important aspect of P-ACO, which makes it interesting for a hardware based implementation, that it transfers less and only the most important information from one iteration of the algorithm to the next. This observation inspires an alternative ant decision procedure, which processes data only provided by the population as depicted in Figure 4.26.

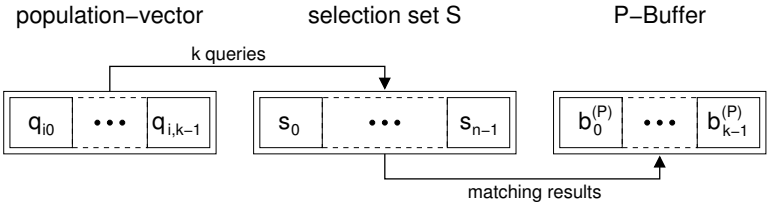


Figure 4.26: Process of building distribution of selection probabilities. All components are shown with their maximum number of items they can store.

Consider a decision in row number i, then *population-vector* $(q_{i0}, \ldots, q_{i,k-1})$ denotes the $i-th$ row of queue matrix Q, and $\{s_0, \ldots, s_{N-1}\}$ describes the selection set of the $N \leq n$ yet available items. One by one, all k items in the population-vector are used to query the selection set whether the respective items are still contained in S. A successful query $(\exists j \in \{0, \ldots, N-1\} : q_{ih} = s_j)$ is called a match producing a *matching result*. This matching result can be either the matched item s_j itself or the address $a_j = a(s_j)$ at which this item is stored within the selection set. Here it is assumed that matching results are represented by matched items, which are collected in a *match buffer*. As the matched items are associated with items from the population, this match buffer be more precisely referred to as the population-buffer (*P-Buffer*). After all queries are finished, the

P-Buffer keeps $M_P \leq k$ matched items, each associated with the weight Δ_P. Since item j may occur multiple times within the population vector, the P-Buffer finally holds ζ_{ij} copies of this item if j is still contained in S. Filling the P-Buffer implicitly produces a distribution of selection probabilities:

$$\forall j \in S : p_{ij} = \frac{\tau_{ij}}{\sum_{l \in S} \tau_{il}} = \frac{\tau_{min} + \zeta_{ij} \cdot \Delta_P}{\sum_{l \in S}(\tau_{min} + \zeta_{il} \cdot \Delta_P)} = \frac{1 + \zeta_{ij} \cdot \Delta_P}{N + M_P \cdot \Delta_P} \tag{4.14}$$

A selection is made by drawing an integer random number rn from the range $[0, \ldots, N + M_P \cdot \Delta_P - 1]$. The selected item $j^* \in S$ is determined as follows:

$$j^* := \begin{cases} s_{rn} & : \quad rn < N \\ b_j^{(P)} & : \quad else \end{cases} \tag{4.15}$$

with $j = \lfloor \frac{rn-N}{\Delta_P} \rfloor$. This means that item j^* is either chosen from selection set S or from the P-Buffer. Building up the probability distribution in Equation (4.14) requires $O(k)$ steps. Drawing a random number and selecting an item can be accomplished in $O(1)$ time. Hence, a complete ant decision can be computed in $O(k)$ time, compared to $O(n)$ for ant decisions based on pheromone information. As stated before, population size k can be considered as a small problem-independent constant, so that the proposed modification allows to compute an ant decision in asymptotically constant time.

4.4.3 Implementation of P-ACO on FPGA

This section first gives a general top-level overview of the P-ACO hardware implementation. Thereafter the implementation of the main modules is explained in greater detail. Note that in this section, heuristic information is disregarded, but the inclusion of a suitable heuristic is discussed in Section 4.4.5. The P-ACO implementation described uses SMTTP as an application example.

4.4.3.1 Overview

The processing flow executed by the P-ACO is presented in Figure 4.27. The algorithm starts by initializing selection set $S := \{0, ..., n-1\}$ and problem-specific evaluation data (the processing times p_j and due dates d_j). The population is initialized to an empty set $P := \emptyset$. Afterwards m ants iteratively construct solutions until some stopping condition is fulfilled. Here m solutions are generated and evaluated in parallel. After comparing the results of m evaluations, the best of these m solutions is used to update the current population.

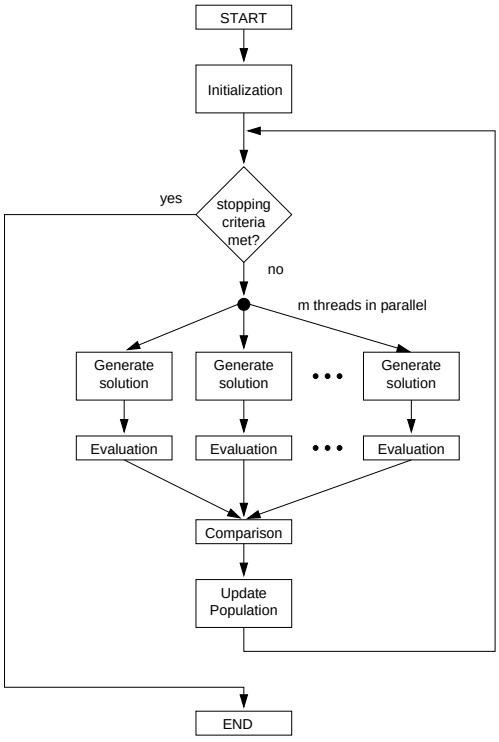


Figure 4.27: Parallel P-ACO processing flow.

At high level, the mapping of the P-ACO algorithm into the corresponding FPGA design consists of four main modules: Population, Generator, Buffer, and Evaluation Module (see Figure 4.28). Note that, for the sake of clarity, the Control circuitry and the control signals are omitted. The Population Module provides the logic to store and control the queue matrix $Q = [q_{ij}]$. Here it is assumed that the P-ACO algorithm is implemented with an additional elitism strategy. The elitist solution is also a member of the population, and within the queue matrix it is placed in column $j = 0$, whereas the FIFO-queue is situated in the adjacent columns $j \in \{1, ..., k-1\}$. For SMTTP, each item $q_{ij} \in \{0, ..., n-1\}$ is the number of a job to be scheduled. The Population Module is responsible for broadcasting items q_{ih} ($h \in \{0, ..., k-1\}$) in the i-th row of the population matrix to the Generator Module. Furthermore, at the end of the current iteration it receives the best solution from the Evaluation Module, which is then inserted into the queue. The Generator Module holds m solution generators working concurrently, one Solution Generator per ant. The solutions are transferred from there to m parallel solution buffers and evaluation blocks inside the Buffer and the Evaluation Module respectively. Simply providing temporary storage until the next population update, the Buffer Module is not further described. It is also

possible to have less than m solution generators, solution buffers, and Evaluation Blocks, which is discussed in greater detail in Section 5.1. The evaluation results of these m solutions are collected in a Comparison Block, which determines the best solution of the current iteration. This best solution also becomes the new elitist solution, if it is better than the current elitist solution.

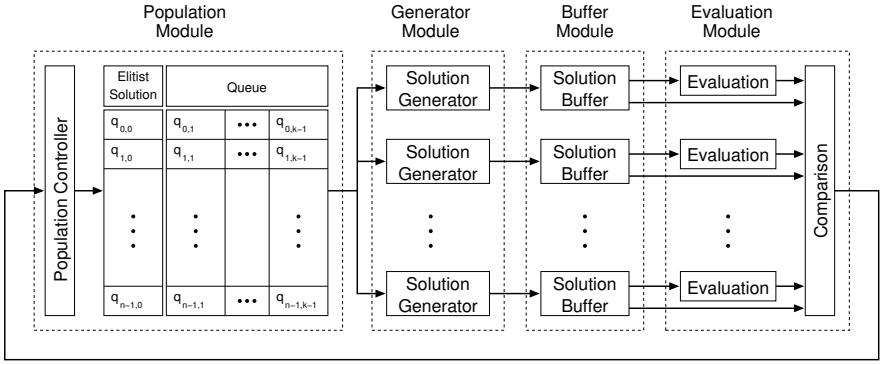


Figure 4.28: P-ACO design with four main modules.

4.4.3.2 Generator Module

The Generator Module contains m identical solution generators, each of them simulates the behavior of an artificial ant constructing a solution using the P-ACO algorithm.

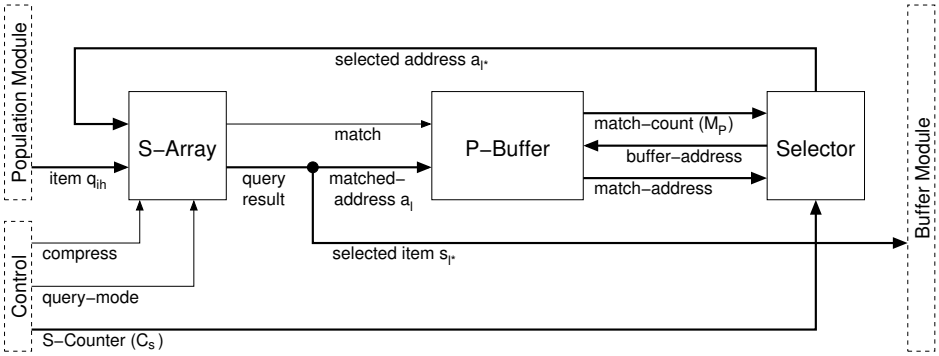


Figure 4.29: Solution Generator.

The Solution Generator (see Figure 4.29) consists of the three blocks S-Array, P-Buffer, and Selector. The S-Array stores and maintains selection set $S \subseteq$

$\{0, ..., n-1\}$ of items $s_i \in S$ with $i \in \{0, ..., C_s\}$. $C_s = |S|-1 = N-1$ is the current value of a decremental counter called S-Counter delivering the maximum index of all items $s_i \in S$. This S-Counter is implemented in external logic in the Control circuit. The current S-Counter value is available to all solution generators. The S-Array can be queried for its contents. When it receives a broadcast item q_{ih} from the Population Module, it compares this item with all remaining items in S. If there exists an item $s_l \in S$ with $s_l = q_{ih}$ (i.e. item s_l has been matched) then the S-Array returns the matching result in terms of address $a_l = l$, which is the address of s_l within the S-Array. In this circuit design, matching results are represented by match addresses needed to identify the selected item. The P-Buffer stores these match addresses a_l into a register. The Selector builds-up a probability distribution and randomly selects an item $s_{l^*} \in S$ which is then sent to the Solution Buffer.

Algorithm 4.3 Processing flow of Solution Generator.

```
1: /* initialization */
2: for j := 0 to n − 1 do
3:     s_j := j
4: end for
5:
6: /* construct solution */
7: for i := 0 to n − 1 do
8:     M_P := 0
9:     /* filling P-Buffer */
10:    for h : 0 to k − 1 do
11:        if ∃s_l ∈ S : s_l = q_ih then
12:            b^(P)_{M_P} := a_l
13:            M_P := M_P + 1
14:        end if
15:    end for
16:
17:    /* selection */
18:    randomly select address a_{l*} = l*
19:    search item j* = s_{l*} ∈ S
20:    S := S \ {j*}
21: end for
```

The processing flow within a Solution Generator is described by Algorithm 4.3. The Solution Generator starts off by initializing the selection set S with numbers $s_j := j$ for all $j \in \{0, ..., n-1\}$. All items q_{ih} with $h \in \{0, ..., k-1\}$ in row i are received coming from the population and forwarded to the S-Array. Whenever

an item s_l in the S-Array has been matched, i.e. $s_l = q_{ih}$, then the corresponding match address a_l is stored into the P-Buffer, and the Match Counter M_P is incremented by 1. Note that if an item matches multiple times its address is also transferred multiple times, and M_P is increased accordingly. After the last repetition of the inner loop, the P-Buffer contains the addresses of all items which appear in row i of the population as well as in selection set S. From these matches the selection probabilities p_{ij} are derived as described by Equation 4.14. According to this probability distribution the ant (Solution Generator) makes a decision for the i-th place in the solution: It randomly selects an address $a_{l^*} = l^*$, which is sent to the S-Array to query for the selected item $j^* = s_{l^*}$ stored at address a_{l^*}. The selected item s_{l^*} is then transferred to the respective Solution Buffer. Afterwards s_{l^*} is removed from selection set S and the process continues making decisions for the remaining places of the solution.

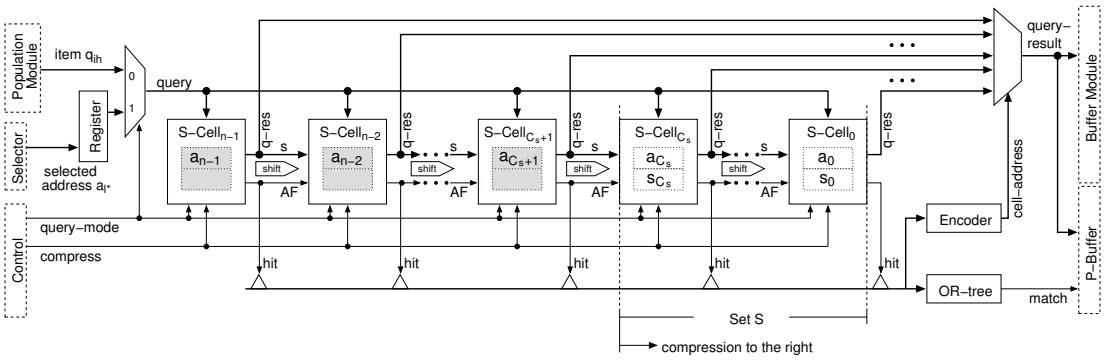


Figure 4.30: S-Array. Items $s_l \in S$ are stored at addresses $a_l = l$ with $a_l \leq C_s$. Shaded cells with addresses $a_l > C_s$ are deactivated.

The organization of an S-Array within the Generator Module is shown in Figure 4.30. Incoming queries are broadcast via a fanout bus to n parallel S-cells. Each S-Cell is connected to its successor to realize a right-shift of items during array compression (removal of the selected item from set S), which is done to guarantee that the items in S are always stored in S-cells with addresses $a_l \leq C_s$ (active S-cells). Only active S-Cells contain valid items in S as indicated by the Active Flag (`AF`). An OR-tree generates the `match` detection bit for the entire array. An n-to-$\lceil \log_2 n \rceil$ encoder is required to produce the address of the S-Cell, which asserted the `hit` signal. This `cell-address` represents the select signal of the multiplexer connecting the respective `q-res` bus to the `query-result` output.

An S-Cell performs the basic operations during solution generation, its architecture is shown in Figure 4.31. It contains two comparators, a hard-coded address a, and a register storing item s. It is controlled via two flags (Active Flag

AF and Match Flag MF) and two signals (query-mode and compress) encoding the current phase. The operation of the S-cells and the S-Array consists of three phases: the Broadcast Phase, the Selection Phase and the Compression Phase.

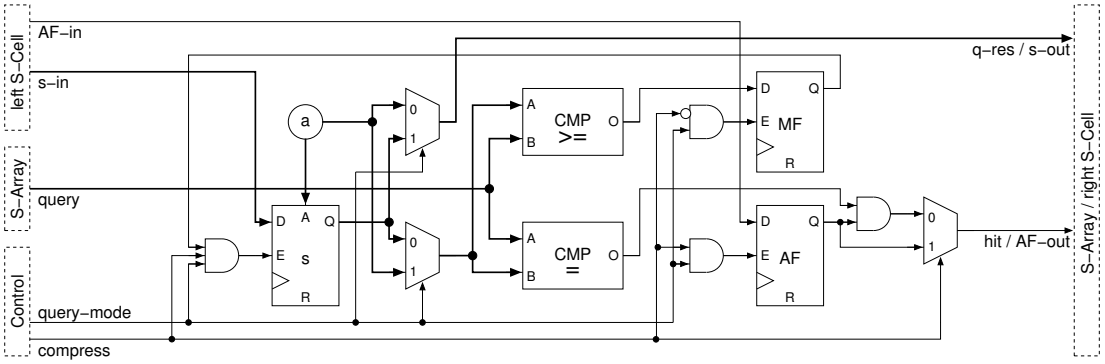


Figure 4.31: S-Cell architecture.

During the Broadcast Phase (query-mode=0, compress=0) the k items in population row i are broadcast to all S-cells (i.e. query $= q_{ih}$). The eq-comparator in the S-Cell compares the broadcast item with its own stored item s. If they are equal the S-Cell emits its address a as query result q-res and asserts a hit signal, if the respective S-Cell is active (AF=1).

After the Selector has randomly selected item s_{l^*} the S-Array enters the Selection Phase (query-mode=1, compress=0). The selected address a_{l^*} is broadcast to the S-cells (i.e. query $= a_{l^*}$). Each S-Cell uses its eq-comparator to compare the broadcast address to its own address a. S-Cell l^* forwards its stored item s_{l^*} as query result q-res to the Solution Buffer. Each S-Cell l also uses its own ge-comparator to check whether it is a predecessor of the selected S-Cell with respect to the shift chain, i.e. $a_l \geq a_{l^*}$. All predecessor S-cells set their Match Flags to MF:=1, all others set MF:=0. These flags are further processed during the Compression Phase.

In final phase, the Compression Phase (query-mode=1, compress=1), each S-Cell sends its Active Flag (AF) as AF-out and its stored item s as s-out to its immediate successor. An S-Cell latches the corresponding data AF-in and s-in arriving from its immediate predecessor, if its own Match Flag was set (MF=1) in the Selection Phase. A logical zero is loaded into the AF flip-flop of S-Cell number $n-1$. The overall effect is that the selected item s_{l^*} is overwritten, all items to the left of the selected cell (i.e. $a_l > l^*$) are shifted one cell to the right. After an iteration is complete, the original values of the data registers in the S-Array are re-initialized by loading the hard-coded S-Cell address values a into the s-registers.

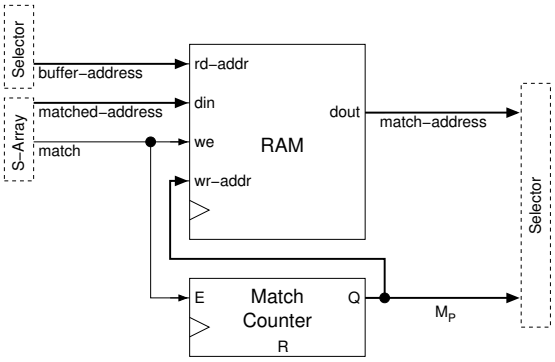


Figure 4.32: P-Buffer with match addresses stored in RAM.

The P-Buffer circuit stores incoming match addresses into a memory if the match signal is asserted. The value M_P of the Match Counter represents the write address for the next match address to be stored. After the last broadcast M_P indicates the total number of matches. The architecture of the P-Buffer is shown in Figure 4.32.

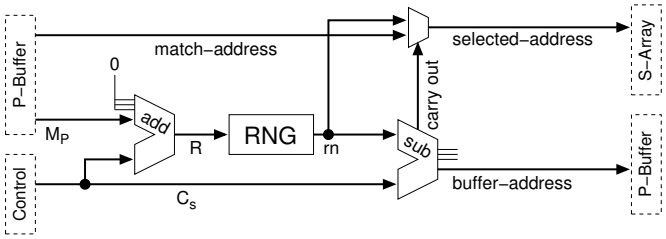


Figure 4.33: Selector. In this example: $\Delta_P = 8$, shift distance $y = 3$ bits.

The Selector (see Figure 4.33) receives the number of matches M_P from the P-Buffer. This circuit is designed to realize the selection of an item according to Equation 4.14. The upper bound $R = C_s + M_P \cdot \Delta_P$ for a pseudo-random number generator is calculated, where $\Delta_P = 2^y$ is chosen to be a power of two in order to replace multiplications and divisions, which are not readily supported on current FPGA technology, by simple left and right shifts respectively. A random number rn is drawn uniformly from the interval $[0, R]$ using the parallel pseudo-random number generator presented by [Ackermann et al., 2001]. Afterwards rn is compared with C_s to determine whether an item from the S-Array or a buffer-address in the P-Buffer has been chosen. If $rn \leq C_s$, then $a_{l^*} = a_{rn}$ is the address of the selected item $j^* = s_{l^*} \in S$. Otherwise, $a_{l^*} = b_l^{(P)}$, with $l = (rn - C_s) \gg y$, i.e. the P-Buffer is queried for the actual match address a_{l^*}. Recall that the P-

Buffer contains the addresses $\{b_0^{(P)}, \ldots, b_{M_P-1}^{(P)}\}$. After a_{l^*} has been determined, the corresponding item s_{l^*} is queried from the S-Array and sent to the Solution Buffer.

4.4.3.3 Evaluation Module

The Evaluation Module holding m Evaluation Blocks evaluates and compares the solutions produced by the solution generators. An Evaluation Block is required for each distinct optimization problem to be solved using the P-ACO algorithm. Such an Evaluation Block takes problem specific evaluation parameters (e.g. job processing times p_i and due dates d_i for the SMTTP) and calculates the objective function value of a solution arriving from a Solution Generator.

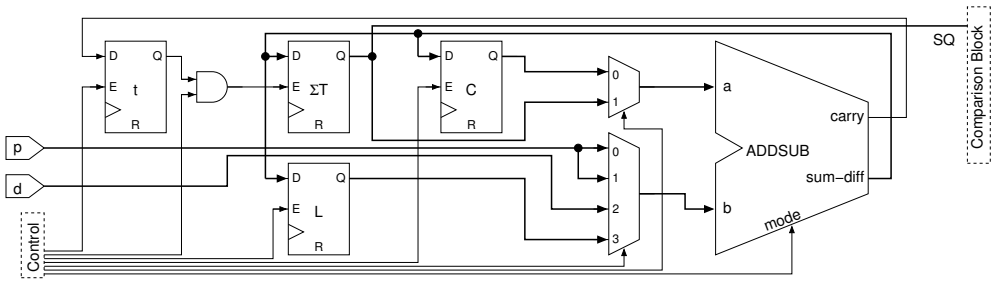


Figure 4.34: Calculation of the objective function for SMTTP. Reset and clock wires are omitted.

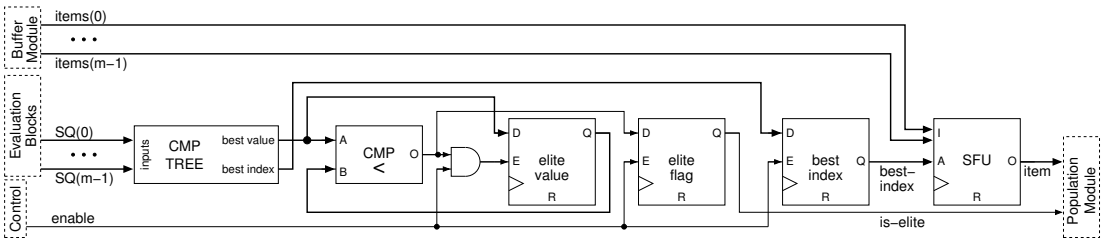


Figure 4.35: Comparison Block with reset and clock wires omitted.

The data path design for calculating the objective function for SMTTP is depicted in Figure 4.34. Dedicated registers are used to store the intermediate total tardiness $\sum T$ and the job completion time C. If a job is determined to be tardy ($t = 1$), then its lateness L is added to the tardiness sum $\sum T$. The only processing component used is an adder/subtractor, which can be implemented with

small delays by the dedicated carry-chain present in current FPGA architectures. The final solution quality `SQ` is sent to the Comparison Block.

The Comparison Block determines the best solution of the iteration and the new elite solution (if found) to be stored into the population. The design of the Comparison Block for SMTTP is shown in Figure 4.35. A comparator tree reduces the logic delay of the Comparison Block. The index (`best-index`) of the Solution Generator, which constructed the best solution, is used by the Solution Forwarding Unit (SFU) to multiplex the buffered solutions. The items of the best solutions is sequentially forwarded to the Population Module. If the best solution of the current iteration is also better than the elite solution, then an elite flag `is-elite` is asserted.

4.4.3.4 Population Module

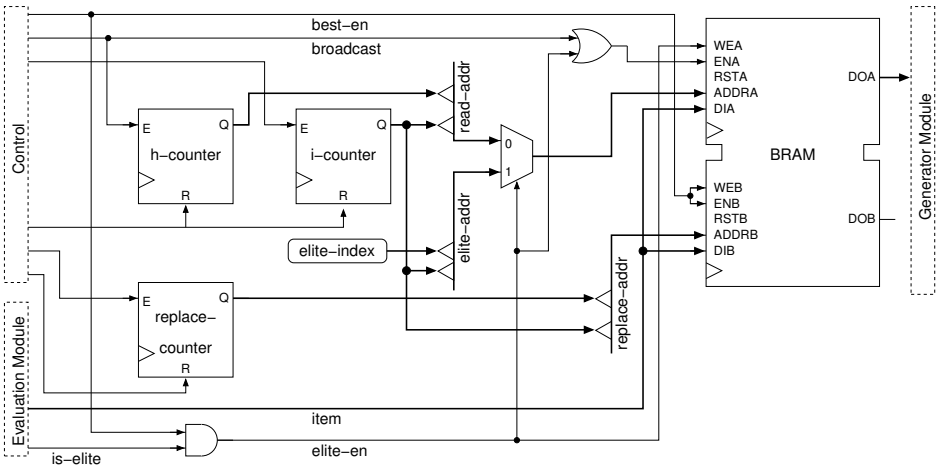


Figure 4.36: Population Module.

The Population Module (see Figure 4.36) provides the storage for the queue matrix. This storage makes use of internal block RAM (BRAM) available on many FPGA architectures. The BRAM is supposed to provide two ports A and B for concurrent input and output, although output port B is actually not used. Apart from the BRAM the remaining logic realizes the Population Controller. The Population Module can be switched between two modes: the broadcast mode and the update mode. When in broadcast mode, the concatenation of the h-counter and the i-counter builds up the read address of queue item q_{ih}, which is broadcast to the solution generators. Running in update mode, the Population Module sequentially receives the n items of the best solution of the

current iteration. This best solution replaces the oldest solution in the population whose index is determined by the replace-counter. Together with the i-counter this index creates the write address for the incoming items written via port B. If this solution is also a new elitist solution (`is-elite` asserted) then the old elitist solution is concurrently overwritten by use of port A.

4.4.3.5 Generic Framework for P-ACO on FPGA

So far the P-ACO design described targets SMTTP as an application. However, the design exhibits many generic portions and can therefore be adapted to other permutation problems by performing a few modifications. Subsequently, these modifications are sub-divided into three groups referring to changes evoked by the evaluation function, the encoding style, and symmetric problems.

Evaluation Function Most obviously, with every changing optimization problem the evaluation logic realizing the objective function may vary. In how far the modifications affect the timing of the execution is discussed in Section 4.4.4.2.

Encoding Style Using SMTTP as an example, parts of the circuitry are specific for place-item encoding. If item-item encoded optimization problems are to be implemented then the required changes concern the communication between the Population Module and the solution generators. With respect to place-item encoded optimization problems, the algorithm can be implemented such that all solution generators synchronously receive the same items from the population, i.e. at any time all solution generators access the same address of the BRAM storing the population. When working on item-item encoded problems, however, the solution generators may make ant decisions in different rows of the population and demand an individual access to population data. Hence, a two port BRAM storage would not be sufficient for more than two solution generators working in parallel.

Therefore two options lend themselves to overcome this bottleneck. If the amount of on-chip RAM blocks permits, then one could maintain multiple copies of the population such that every two solution generators share a common two-port BRAM. Another possibility would be to store population data in distributed memory as typically provided by flip-flops or look-up tables. All rows of the population $(q_0, \ldots, q_{n-1})$ should be addressable in parallel, and all solution generators share this distributed memory via an $n \times m$ crossbar switch as depicted in Figure 4.37. Using a crossbar even offers an interesting way of exploiting the partial reconfiguration capabilities of FPGAs (see [Young et al., 2003] for an example of a runtime reconfigurable crossbar switch).

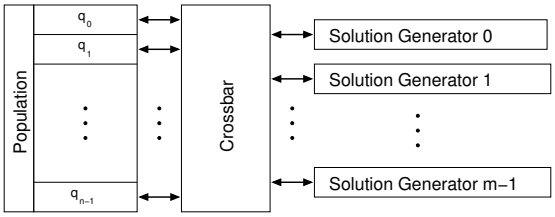


Figure 4.37: Population access via crossbar switch.

Item-item encoded problems further demand a slight modification to the Solution Generator: Whereas in place-item encoded problems, rows in the population are broadcast top-down one after the other, item-item encoding demands the selected item s_{l^*} also to be sent to the Population Module as the address of the next row to be processed (see Figure 4.29). Correspondingly, the Population Module has to be changed as well. Furthermore, prior to every first decision, a row i needs to be determined from which the ant starts from. Memorizing row i, it must be ensured that item i is selected with the final decision of the same construction process. It is also necessary, that the Buffer Module stores the selected items received in the proper order according to the item-item encoding.

Symmetry If symmetric problems are considered, i.e. such problems commonly using a symmetric pheromone matrix, this symmetry needs to be reproduced by the population. This can be easily accomplished by maintaining for every solution $\pi \in P$ an additional symmetric solution $\pi' \in P$ with $\pi'(\pi(i)) = i$. Accordingly, the population and P-Buffer sizes increase to $2k$, and every item in the population is associated with only half of its usual weight $\Delta_P/2$.

4.4.4 Experimental Results

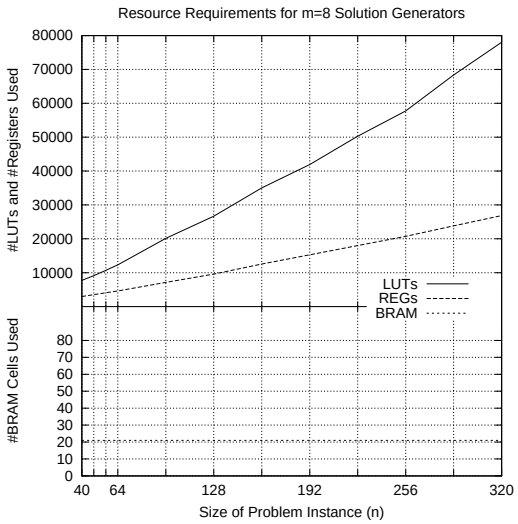
The hardware implementation of P-ACO is compared with its software counterpart by utilizing contemporary FPGA technologies. For the hardware implementation, the design is encoded in register-transfer level VHDL targeting the Xilinx Virtex-II Pro Platform FPGA XC2VP125. The design is synthesized by XST 5.2sp1 on normal optimization effort for high level logic synthesis, and Xilinx ISE 5.2sp1 Place and Route for implementation on the FPGA. In software, P-ACO is programmed in C++ to be executed on a AMD Athlon uni-processor machine clocked at 1540 MHz to measure timing performance of the software implementation. The experiments restrict to comparing the implementation on a single FPGA with a software-based solution on a single processor. Considering an implementation on a multi-FPGA-board or a parallel variant of the software

implementation is an interesting aspect for future research.

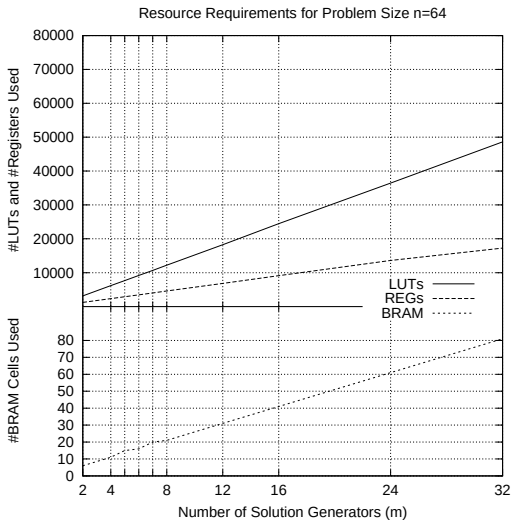
The optimization problem regarded is SMTTP, where the problem size, i.e., the number of jobs, is scaled within the range from $40 \leq n \leq 320$. Job processing times are chosen randomly from the $p_i \in \{1, \ldots, 64\}$. The number of solution generators (which equals the number of ants per iteration) ranges from $m = 2$ to $m = 32$. The population size is set to $k = 4$ including one elitist solution. For the hardware implementation, the utilization of FPGA resources, circuit delays, and operational frequencies are recorded. For every problem instance of the software counterpart, the execution time per iteration is recorded as an average over 100000 iterations of the P-ACO algorithm.

4.4.4.1 Resource Requirements

The resource requirements are measured by counting the number of look-up tables (LUTs), slice registers (REGs), and internal block RAM cells (BRAM) used by the design implementations. In Figure 4.38a, resource requirements are depicted for a fixed number of solution generators $m = 8$ and variable problem size n. The resource requirements shown in Figure 4.38b are for a fixed problem size $n = 64$ and a variable number of solution generators.



(a) Varying problem sizes n, fixed number of solution generators ($m = 8$)



(b) Varying number of solution generators m, fixed problem size ($n = 64$)

Figure 4.38: FPGA resource utilization – number of BRAM cells, number of LUTs, number of slice registers – for varying problem sizes and number of implemented solution generators.

As the problem size increases (see Figure 4.38a), the number of S-cells, i.e. the width of an S-Array, grows proportionally. The height of the S-Array, grows logarithmically, because each S-Cell stores one item represented by $\log n$ bits. Since the S-arrays occupy the largest portion of the total P-ACO circuitry, the overall consumption of LUTs and REGs grows according to $O(m \cdot n \cdot \log n)$.

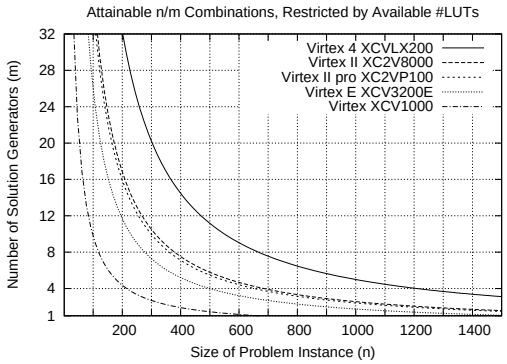
Table 4.1: Regression results for # LUTs, # REGs and # BRAM. For every regression function, the Bravais-Pearon correlation index r is given.

Fixed	Resource	Regression Function f	r
$m = 8$	LUTs	$28.2257 \cdot n \cdot \log n + 1531.71$	0.999079113
	REGs	$9.74471 \cdot n \cdot \log n + 929.175$	0.999861506
	BRAM	21	undefined
$n = 64$	LUTs	$1516.17 \cdot m + 97.2749$	0.999987262
	REGs	$542.244 \cdot m + 244.764$	0.998796397
	BRAM	$2.4802 \cdot m + 1.5297$	0.999374327

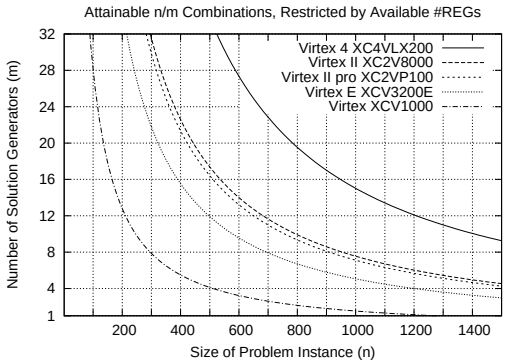
This theoretical assumption is confirmed by the regression results (see Table 4.1) which correspond to the values shown in Figure 4.38. As indicated by the Bravais-Pearon index r very close to 1, all resource requirements show a very strong correlation to m and n respectively. The number of BRAM cells used remains on a constant level of 21 when varying problem size n for fixed m. For an increasing number of solution generators (see Figure 4.38b), all resource requirements grow linearly. Given a fixed problem size of $n = 64$, additional 1516 LUTs, 542 REGs, and 2.5 BRAM cells approximately are needed, if the P-ACO circuit shall be extended by a further Solution Generator (see Table 4.1).

Figure 4.39 provides an overview of the maximum attainable combinations of m and n for the largest members of the Xilinx Virtex, Virtex E, Virtex II, Virtex II pro, and Virtex 4 FPGA families. Note that in practice, it is inefficient to use very large numbers of ants (solution generators). Therefore, in the diagrams the upper limit in the range of solution generators is set to $m = 32$. The left figure shows all combinations, if the implementations were only restricted by the number of available LUTs; whereas the right figure shows all implementations, if they were only restricted by the number of available REGs. On each of the FPGA devices considered, the number of LUTs is equal to the number of REGs. However, in the design implementation the number of configured LUTs is about 170% higher than the number of REGs used. Hence the curves in the right figure are considerably higher than the respective curves on the left hand side. These figures suggest that even on nowadays available FPGA devices, it is possible to accommodate P-ACO circuits for SMTTP realizing combinations of m and n

with practical relevance.



(a) Restrictions set by the number of available LUTs



(b) Restrictions set by the number of available REGs

Figure 4.39: Attainable combinations of n and m for different types of FPGAs.

4.4.4.2 Time Requirements

The generation of a solution for SMTTP consists of n decision steps (see Figure 4.40). For each decision, k queue items are broadcast and matched. The time requirement for this operation is denoted by t_{bm}. Afterwards item s_{l^*} is selected and evaluated, which takes t_s and t_e time. The time to compress the S-Array is denoted by t_{cps}. Due to the generic implementation of the solution generators, for a given problem size n, the total time for any decision t_d is constant and does not depend on the optimization problem under consideration. On the other hand, the time to evaluate an item does strongly depend on the optimization objective. Thus the time for generating a complete solution is $t_g = n \cdot t_d$ if $t_{cps} \geq t_e$, otherwise $t_g = \max\{n \cdot t_d + t_e, n \cdot t_e + t_d\} - t_{cps}$. The experiments are conducted for SMTTP so that evaluation does not retard the beginning of the next decision, i.e. $t_g = n \cdot t_d + t_e - t_{cps}$. In every iteration, m solutions are generated in parallel. Afterwards these solutions are compared by their solution qualities and the population is updated, which requires t_{cmp} and t_u time respectively. Thus the time to finish a complete iteration is $t_{it} = t_g + t_{cmp} + t_u$. The time to re-initialize the solution generators can be disregarded.

Table 4.2 shows the maximum global clock frequencies f_{gl} for a fixed number of solution generators $m = 8$ and varying problem sizes n. It further gives an overview of the required clock tics: number of clock tics for generating a solution $c_g = t_g \cdot f_{gl}$, clock tics for comparison $c_{cmp} = t_{cmp} \cdot f_{gl}$, clock tics per population update $c_u = t_u \cdot f_{gl}$, and clock tics per iteration $c_{it} = t_{it} \cdot f_{gl} = c_g +$

$c_{cmp} + c_u$. Accordingly, the maximum frequency per iteration can be calculated: $f_{it} = f_{gl}/c_{it}$.

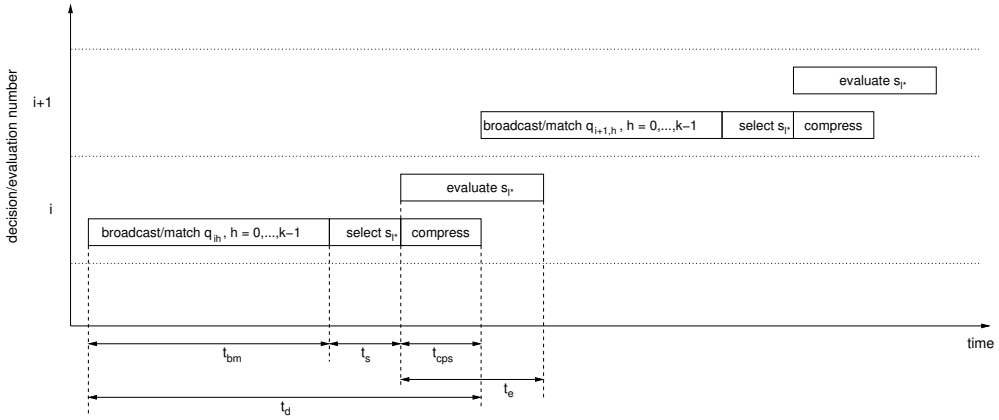


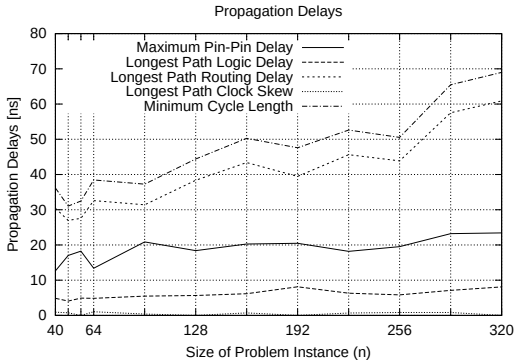
Figure 4.40: Schedule of two decisions and evaluations. Here only one solution generator is considered.

Table 4.2: Operating frequencies for the P-ACO implementation in hardware with $m = 8$ solution generators on the Virtex-II Pro X2VP125-7 device.

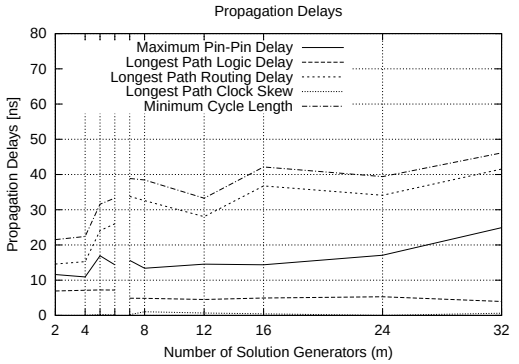
	Clock Tics				Max f_{gl} (MHz)	Max f_{it} (kHz)
n	c_g	c_{cmp}	c_u	c_{it}		
40	403	2	40	445	27.651	61.935
48	483	2	48	533	32.219	60.448
56	563	2	56	621	30.806	49.607
64	643	2	64	709	25.997	36.667
96	963	2	96	1061	26.832	25.289
128	1283	2	128	1413	22.522	15.939
160	1603	2	160	1765	19.899	11.274
192	1923	2	192	2117	21.011	9.925
224	2243	2	224	2469	19.001	7.695
256	2563	2	256	2821	19.789	7.015
288	2883	2	288	3173	15.273	4.813
320	3203	2	320	3525	14.494	4.112

Figure 4.41 presents the timing results measured. The gap at $m = 7$ is present as the endpoints of the critical path change. For $m \leq 7$, the address counter to the S-Array output is the critical path; whereas for $m \geq 8$, a control signal is the source of the critical path. The increase in the critical path length can be

attributed to the increased logic levels as the design grows in size; the growth in critical path length for increasing numbers of solution generators implemented is likely to be due to the increased difficulty of solving the placement and routing problems for large circuit sizes. In all instances, routing delay is the major component of the total critical path delay. Clock skew is a negligible problem for larger circuits due to the dedicated clock routing resources of the FPGA. As the number of solution generators is increased, propagation delays grow only slightly.

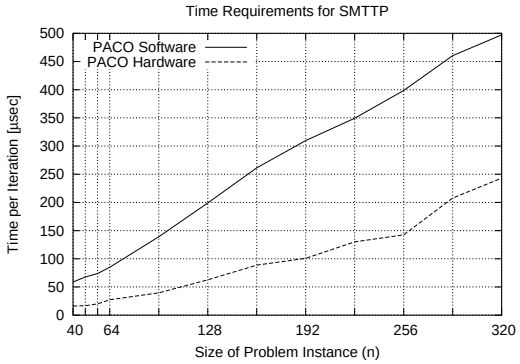


(a) Number of implemented solution generators $m = 8$, varying problem size n

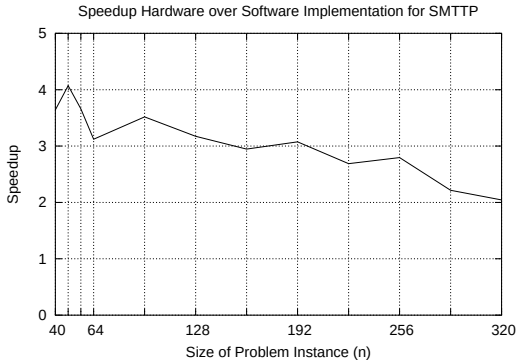


(b) Problems of size n =64, varying number of implemented solution generators m

Figure 4.41: Delay components contributing to critical path for different problem sizes and number of implemented solution generators.

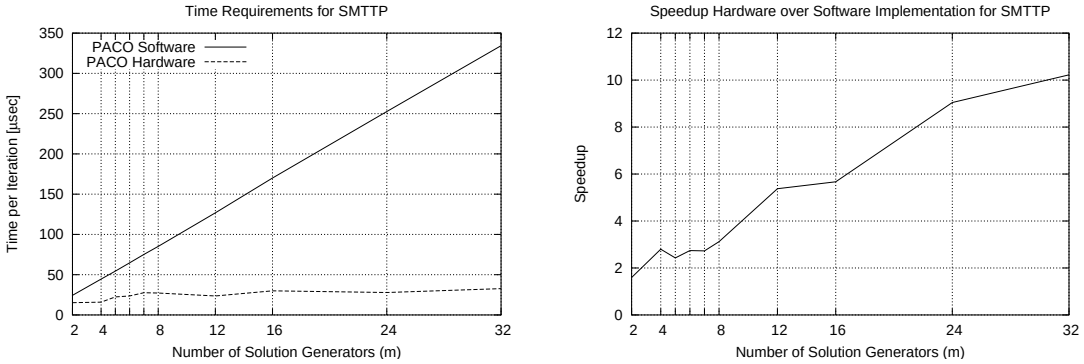


(a) Time requirements per iteration.



(b) Speedup of hardware over software implementation.

Figure 4.42: Comparison of hardware and software implementation of P-ACO for fixed $m = 8$.



(a) Time requirements per iteration.

(b) Speedup of hardware over software implementation

Figure 4.43: Comparison of hardware and software implementation of P-ACO for fixed $n = 64$.

In figures 4.42 and 4.43, the software and the hardware realizations of the P-ACO algorithm are compared with each other. Since the software and hardware versions produce the same solutions, their performances are compared by means of the time required per iteration. Note that, except for the first k iterations, the execution time for every iteration is constant. The figures on the left compare the time per iteration, whereas the figures on the right depict the respective speedup values, where the speedup is defined by:

$$\text{Speedup} = \frac{\text{Time per iteration in software}}{\text{Time per iteration in hardware}}$$

In Figure 4.42 the execution time per iteration for both, the software and the hardware implementations, grow consistently as the problem size n is increased. However, for every problem size, the hardware version runs faster than its software counterpart. Speedup values range from a minimum of 2.04 for a problem size of $n = 320$ to maximum value of 4.07 for $n = 48$.

Figure 4.43 shows that the time per iteration in software grows with a linear trend as the number of solution generators is increased; whereas for the hardware implementation, the time requirements remain on an almost constant level of approximately 28 microseconds. The corresponding speedup values range from 1.59 for $m = 2$ solution generators to 10.22 at $m = 32$. The speedup grows almost linearly in m, although the task of placing and routing the design becomes considerably more difficult for larger m. From the P-ACO perspective this means that within certain constraints (e.g. chip size), the degree of parallelism (in terms of concurrently working ants) can be easily increased with only a marginal effect on execution speed.

4.4.5 Heuristic Extension

In this section, the *Time-Scattered Heuristic* (TSH) is introduced as a new heuristic concept for the P-ACO algorithm that sustains the small runtimes of the hardware implementation. The construction and the usage of the new heuristic concept is demonstrated, and the proposed approach is tested on various instances of TSP. The performance of TSH is compared with the standard heuristic approach and candidate lists. Parts of the contents of this section are based upon previous work by the author published in [Scheuermann et al., 2004a].

4.4.5.1 Time-Scattered Heuristic

The new heuristic approach aims to supply the P-ACO algorithm with heuristic information thereby maintaining a constant asymptotic runtime per ant decision. However, the integration of heuristic information into the P-ACO hardware algorithm poses two problems: Heuristic values exist for all items of the set S, not just an $O(k)$ size subset. It should therefore be avoided to calculate the selection probabilities based on products $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$ for all items $j \in S$, which would result in a runtime of $O(n)$ per ant decision. Furthermore, a large number of arithmetic operations on floating-point numbers would be afforded requiring enormous amounts of logic resources on the FPGA.

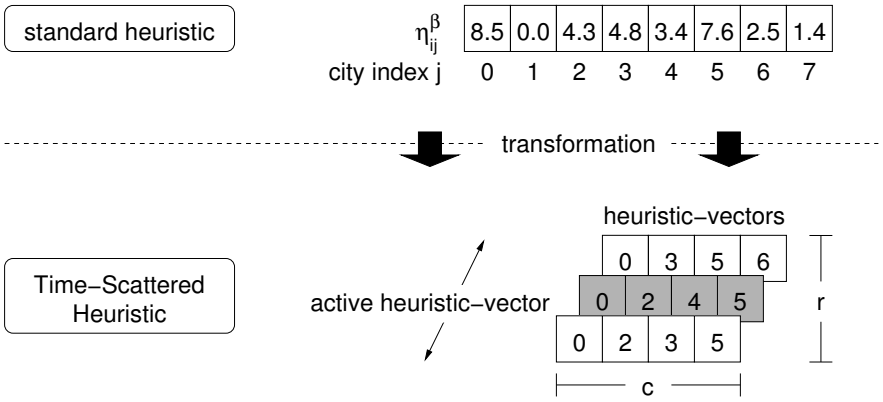


Figure 4.44: Example of transformation process from standard heuristic values into a sequence of heuristic-vectors used by TSH.

The basic idea of TSH is to transform the standard heuristic information into a time-variant sequence of integer vectors, which allow for a sufficiently exact approximation. Figure 4.44 illustrates this transformation process by choosing TSP as an example with $n = 8$ cities. The upper part of the figure visualizes for an arbitrary row i, the weighted standard heuristic values $\eta_{ij}^{\beta} \in I\!R$ for moving

from city i to city j. These heuristic values are transformed into a sequence of $r = 3$ integer *heuristic-vectors* of size $c = 4$ as shown in the lower part of the figure. Each entry in such a heuristic-vector is a city index j. The higher the heuristic value of a city the higher the frequency of the respective city index in the heuristic-vectors. For example, cities 0 and 5 possess the highest heuristic values and therefore appear most often in the heuristic-vectors, whereas cities 1 and 7 with the lowest heuristic values do not appear at all. It is required that the distribution of the original standard heuristic is approximately equal to the distribution of city indices in the heuristic-vectors:

$$\frac{\eta_{ij}^{\beta}}{\sum_{l=0}^{n-1} \eta_{il}^{\beta}} \approx \frac{F_{ij}}{r \cdot c}, \tag{4.16}$$

with F_{ij} denoting the absolute frequency of city j over all heuristic-vectors for row i. City indices are distributed as evenly as possible over all heuristic-vectors, the order of the city indices within an individual heuristic-vector is arbitrary. In order to constrain the runtime of the algorithm, the length of the heuristic-vectors $c << n$ should be kept small. However, to achieve a sufficiently good approximation of the standard heuristic information, the city indices are scattered over multiple heuristic-vectors, but only one at a time is actually used for constructing solutions. This designated vector is called the *active heuristic-vector*. During the runtime of the P-ACO algorithm heuristic-vectors are repeatedly activated and deactivated, which is referred to as a shift of the active heuristic-vector.

For a given row i, Algorithm 4.4 lists the steps executed to transform the standard heuristic information into a sequence of r heuristic-vectors. In the following these vectors are assumed to be arranged in a so-called $r \times c$ *heuristic-matrix* $[h_{uv}^{(i)}]$ with $h_{uv}^{(i)} \in \{0, \ldots, n-1\}$. It is supposed that the heuristic-matrix is calculated and stored on the FPGA before a run of the P-ACO algorithm starts. Therefore, heuristics which require an dynamic computation of the heuristic values cannot be handled by this method (see Section 2.3.2). The transformation starts off by weighting the standard heuristic values with $\beta > 0$. In accordance with standard ACO, this parameter determines the impact of the heuristic on an ant decision. Afterwards δ_i is calculated as the average over all weighted heuristic values. This value is then used to compute the absolute frequency F_{ij} of item j in the heuristic-matrix for row number i. Finally, all items with $F_{ij} > 0$ are inserted into this matrix according to their absolute frequencies. Scanning all items $j \in \{0, \ldots, n-1\}$ and traversing the heuristic-matrix in a column-wise order, it is achieved that item numbers are evenly distributed over all heuristic-vectors.

As mentioned before, during optimization only one of the r heuristic-vectors $h_u^{(i)}$ with $0 \leq u < r$ is actually used by the ants to construct solutions. This vector is referred to as the active heuristic-vector $h_{u^*}^{(i)}$ and all other $r - 1$ vectors are

inactive. After a specific number p_s of solution generations, the active heuristic-vector is shifted, i.e. it is replaced by some other $h_u^{(i)}$ with $u \in \{0, \ldots, r-1\} \backslash \{u^*\}$. Parameter $p_s \geq 1$ is called the shift period. Two different policies of shifting the active heuristic-vector are considered:

Random shift: The next active heuristic-vector u^* is chosen randomly from $\{0, \ldots, r-1\}$ with uniform distribution.

Cyclic shift: The active heuristic-vector rotates through the sequence of available heuristic-vectors, i.e. $u^* := (u^* + 1) \bmod r$.

Algorithm 4.4 Calculation of heuristic-matrix for row i

```
 1: /* initialization */
 2: for j := 0 to n − 1 do
 3:     η̂_ij := η_ij^β
 4:     F_ij := 0
 5: end for
 6: δ_i = 1/(r·c) Σ_{j=0}^{n−1} η̂_ij
 7: /* compute absolute frequencies of items in heuristic-vectors */
 8: for l := 0 to r · c − 1 do
 9:     determine j* with η̂_ij* = max_{j=0,...,n−1} η̂_ij
10:     η̂_ij* := η̂_ij* − δ_i
11:     F_ij* := F_ij* + 1
12: end for
13: /* fill heuristic-matrix */
14: j := 0
15: for v := 0 to c − 1 do                          /* column-wise traversal */
16:     for u := 0 to r − 1 do
17:         found := false
18:         while found = false do
19:             if F_ij = 0 then
20:                 j := j + 1
21:             else
22:                 F_ij := F_ij − 1
23:                 found := true
24:             end if
25:         end while
26:         h^i_{u,v} := j                  /* insert item j into heuristic-matrix */
27:     end for
28: end for
```

In the hardware P-ACO algorithm, the necessary calculations to transform row i of the queue matrix Q into the respective pheromone values τ_{ij} are done according to Equation 4.10. Likewise it is possible to transform any heuristic-vector $h_{u^*}^{(i)}$ into the corresponding heuristic value η'_{ij} describing the heuristical impact of item j when making a decision in row i:

$$\forall\ j \in \{0, \ldots, n-1\} : \eta'_{ij} := \eta_{min} + \gamma_{ij} \cdot \Delta_H, \tag{4.17}$$

where $\eta_{min} > 0$ denotes a base heuristic value assigned to every item j and γ_{ij} describes the number of occurrences of item j in the active heuristic-vector, i.e. $\gamma_{ij} = |\{v : h_{u^*v}^{(i)} = j, v = 0, ..., c-1\}|$ and $\Delta_H > 0$ is the weight assigned to items in the heuristic-vectors.

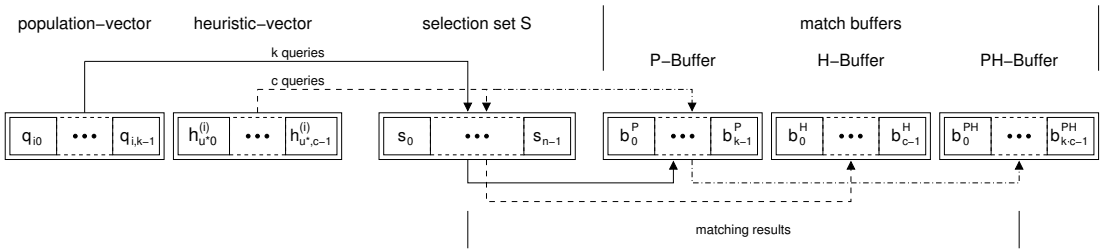


Figure 4.45: Items in the current population row i and the respective heuristic-vector $h_{u^*}^{(i)}$ are transmitted. Matching results are stored into match buffers. All components are shown with their maximum number of items they can store.

The proposed method for heuristic guidance can be integrated into the P-ACO hardware implementation by extending the match buffer concept introduced in Section 4.4.3. Let population-vector $(q_{i0}, \ldots, q_{i,k-1})$ be the currently processed row in the population, and $h_{u^*}^{(i)} = (h_{u^*0}^{(i)}, \ldots, h_{u^*,c-1}^{(i)})$ the current heuristic-vector (see Figure 4.45). All $k + c$ items in both vectors are sent to the selection set S and matching results are then stored into three different types of match buffers. For simplicity it is assumed that as in Section 4.4.2 matching results are matched items. The P-Buffer stores $M_P \leq k$ items, which are in the population-vector as well as in set S. The respective $M_H \leq c$ items which occur in the heuristic-vector and in selection set S are copied into a separate location called the *H-Buffer*. Furthermore, since the pheromone and heuristic values are multiplied to compute selection probabilities, an additional buffer – called *PH-Buffer* – is required. The PH-Buffer stores the items contained in the heuristic-vector as well as in the population-vector and in set S. Therefore, items in the heuristic-vector are sent to selection set S and in parallel to the P-Buffer. Let Δ_P be the weight associated with items in the P-Buffer, and Δ_H the weight of items in the H-Buffer. Then

$\Delta_{PH} = \Delta_P \cdot \Delta_H$ is the weight for an item j which is stored $\phi_{ij} = \zeta_{ij} \cdot \gamma_{ij}$ times in the PH-Buffer. After all queries the PH-Buffer contains $M_{PH} \leq k \cdot c$ items.

Using this extended buffer concept, ants make random decisions according to the following probability distribution p_{ij}:

$$\forall j \in S: \; p_{ij} = \frac{\tau_{ij} \cdot \eta'_{ij}}{\sum_{l \in S} \tau_{il} \cdot \eta'_{il}} = \frac{1 + \zeta_{ij} \cdot \Delta_P + \gamma_{ij} \cdot \Delta_H + \phi_{ij} \cdot \Delta_{PH}}{N + M_P \cdot \Delta_P + M_H \cdot \Delta_H + M_{PH} \cdot \Delta_{PH}} \tag{4.18}$$

Note that the initial values are set to $\tau_{min} = \eta_{min} = 1$. Exponentiations by weight β are calculated offline prior to building the heuristic-matrix. Pheromone and heuristic values are weighted by choosing the Δ parameters accordingly. The required arithmetic operations better suit the resources provided by the FPGA architecture. To perform an ant decision, an integer random number rn is drawn from the range $[0, N + M_P \cdot \Delta_P + M_H \cdot \Delta_H + M_{PH} \cdot \Delta_{PH} - 1]$. This random number determines the selected item j^*:

$$j^* := \begin{cases} s_r & : \; rn < o_P \\ b_{j_P}^{(P)} & : \; o_P \leq rn < o_H \\ b_{j_H}^{(H)} & : \; o_H \leq rn < o_{PH} \\ b_{j_{PH}}^{(PH)} & : \; else \end{cases} \tag{4.19}$$

Here $o_P = N$, $o_H = o_P + M_P \cdot \Delta_P$, and $o_{PH} = o_H + M_H \cdot \Delta_H$ denote the offsets associated with the P/H/PH-buffers respectively. In the case that an item is selected from one of the buffers then $j_x = \lfloor \frac{rn - o_x}{\Delta_x} \rfloor$ with $x \in \{P, H, PH\}$ retrieves the index of the selected buffer item. An efficient parallel implementation of this new heuristic concept on an FPGA allows to make a decision in $O(k + c)$ time, where k and c can be regarded as small constants. Also in the case of exploitation, according to the pseudo-random proportional transition rule (see Section 2.3.2), the search for the maximum over the yet available items in set S can be computed in $O(k+c)$ time. In comparison, a sequential processor that uses the ACO approach requires $O(n)$ time per ant to make a decision. In addition, the functional parallelism embodied in processing m ants in parallel on an FPGA allows m solutions to be formed in $O((k + c) \cdot n)$ time, compared to $O(m \cdot n^2)$ time on a sequential processor using the standard ACO approach.

Figure 4.46 illustrates the integration of TSH into the existing implementation of P-ACO on FPGA. The circuit is extended by an additional Heuristic Module comprising for all rows $i \in \{0, \ldots, n-1\}$, the heuristic-matrices, which are as the population supposed to be stored in internal block RAM. Heuristic-vectors are shown in different layers with the active heuristic-vectors on top. The TSH-Controller is responsible for shifting the heuristic-vectors after a predefined number of solution constructions according to a certain shift policy. To allow for

an easy integration into the existing hardware implementation, it is very useful that heuristic data is treated in the same fashion as population data. The solution generators need to be extended by two further types of match buffers to also process the heuristic data received. The Population, Buffer, and Evaluation Module, however, remain unchanged.

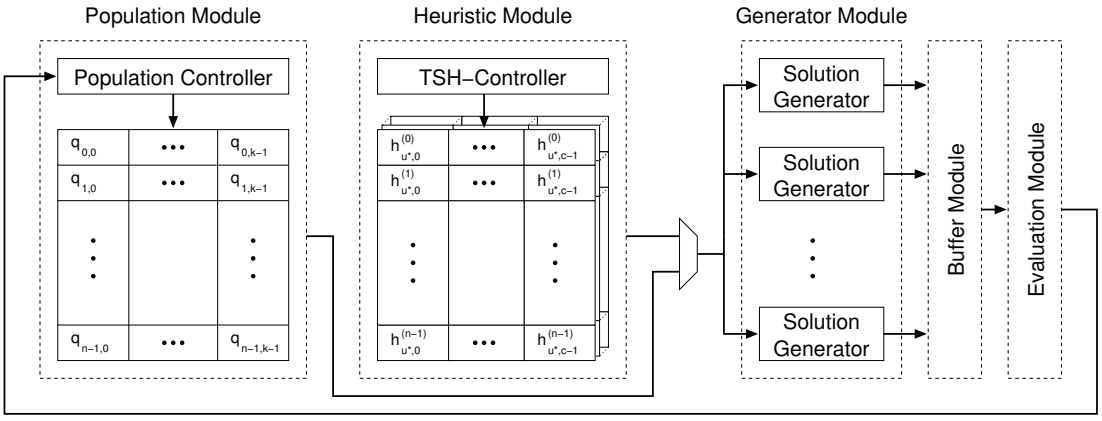


Figure 4.46: Integration of TSH into the P-ACO architecture on FPGA.

4.4.5.2 Experimental Results

Various experiments are carried out to study the performance of the proposed TSH approach. The experiments are run on Pentium 4 CPUs clocked at 1.5 GHz. TSH is not implemented on an FPGA but simulated in software with the goal to gauge the effect of algorithm-specific parameters on optimization performance.

In the first experiment, the influence of parameters r and c, which determine the size of the heuristic-matrix, is studied (see Figure 4.47). The P-ACO algorithm with Time-Scattered Heuristic (P-ACO-TSH) is applied to the symmetric TSP instance `eil101` (101 cities) from the TSPLIB benchmark library. The lower part of the figure shows the deviation of the distribution of items in the heuristic-matrix (obtained by applying Algorithm 4.4) from the distribution of the standard heuristic values η_{ij}^{β}. This deviation is determined by calculating the average quadratic distance between these two distributions. The results show that deviation decreases rapidly with the size of the heuristic-matrix ar. The upper part of Figure 4.47 shows the tour length for different aspect ratios $R = c/r$ (average over 20 runs). The result for P-ACO with no heuristic guidance (P-ACO-NH) is also shown. Suitable parameter settings for P-ACO have been determined in a range of preliminary experiments. The number of ants per iteration is chosen as $m = 8$ together with a population size of $k = 4$ solutions including one elitist

solution, minimum values $\tau_{min} = \eta_{min} = 1$, update values $\Delta_P = f_P \cdot n/(k+1)$ and $\Delta_H = f_H \cdot n/c$ with $f_P = f_H = 10$, weight $\beta = 4$, shift period $p_s = 8$ with random shift policy, and a runtime limit of 1000 seconds for every combination of r and c. For small values of ar and aspect ratios $R \leq 1$ (i.e. $r \geq c$), P-ACO-TSH performs worse than P-ACO-NH. Obviously, in these cases the approximation of the original heuristic information is not good enough and misleads the decisions of the ants. However, as the mean quadratic deviation decreases, the algorithm benefits from the additional heuristic guidance. The solution qualities for $R \leq 1$ and $c \geq 3$ are better than those obtained for P-ACO-NH. Runs with high aspect ratios $R > 1$ perform better for small matrix sizes, since the ants are provided with more heuristic knowledge during individual decisions than in runs with low aspect ratios. For combinations with $R > 1$ solution qualities become worse with increasing matrix size ar and are even worse than P-ACO-NH for $ar > 2^{18}$. This behavior is due to a higher execution time per ant decision which is proportional to c. The overall best solution quality is obtained for parameters $r = 48$ and $c = 3$, i.e. $ar = 144$ and $R = 0.0625$.

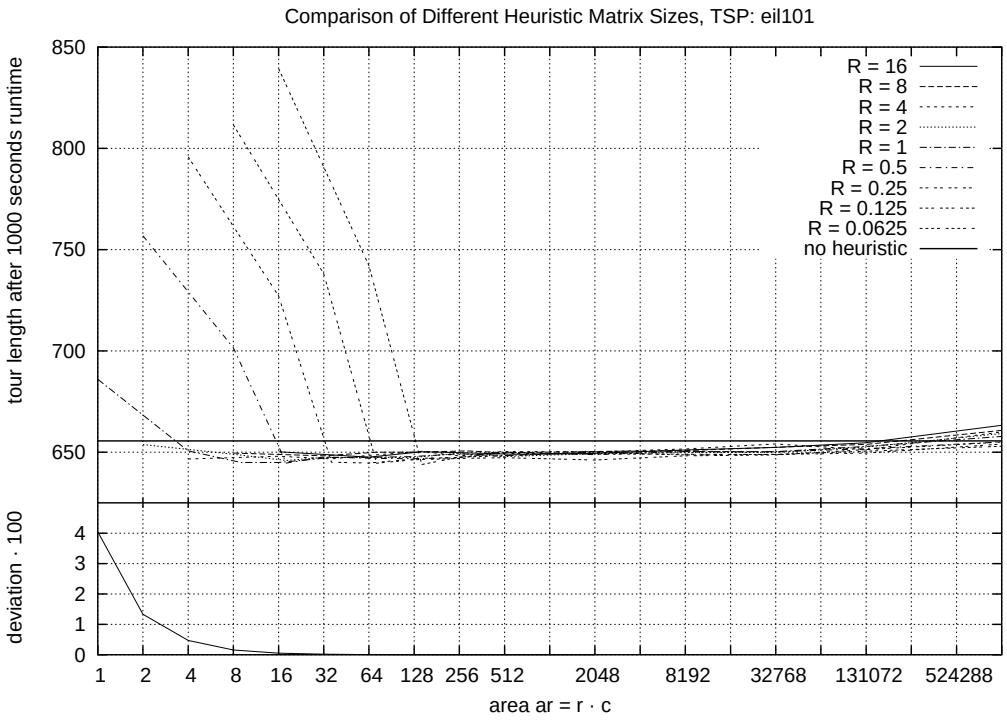


Figure 4.47: Comparison of r and c combinations with different aspect ratios $R = c/r$ and heuristic matrix sizes $ar = r \cdot c$.

The next experiment examines the shift policy (either random or cyclic) and

the shift period, which is scaled from $p_s = 1$ (shift after any solution generation) to $p_s = 2^{20}$ (no shift during the complete run). With constant values for $r = 48$ and $c = 3$, the tour lengths are calculated as the average over 20 repetitions with different random seeds and termination after 100000 iterations, all other parameter settings remain as in the previous experiment. Solution qualities for both policies worsen with increasing shift periods (see Figure 4.48). Apparently, shifting the heuristic-vectors very often better approximates the original heuristic information and provides the ants with more accurate heuristic knowledge. In the majority of shift periods considered, shifting randomly performs better than the cyclic shift policy, which causes the heuristic-vectors to always rotate in the same order. Presumably, this fixed order causes some kind of disadvantageous correlation between active heuristic-vector and the current ant decision. The computational results suggest the choice of random policy with shift periods from the interval $p_s \in [4, 128]$, where the best solution quality is obtained for $p_s = 4$.

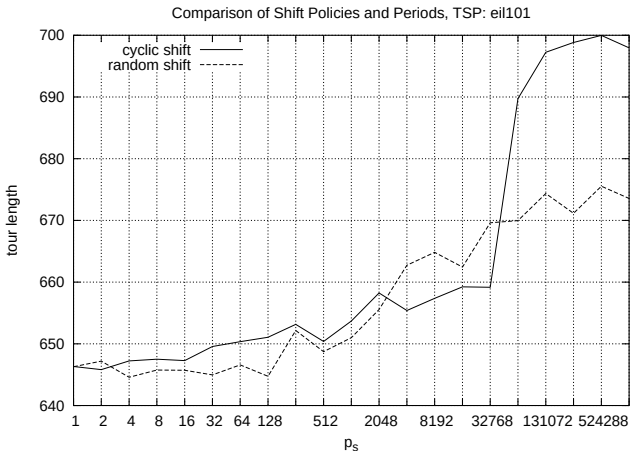


Figure 4.48: Comparison of shift policies cyclic vs. random for different shift periods p_s.

Another experiment is performed to determine the influence of weight β for P-ACO-TSH. Therefore, β is scaled from 1 to 1024 (see Figure 4.49), and heuristic-vectors are shifted after every $p_s = 8$ solution generations with random shift policy. All other parameter settings are chosen as in the previous experiment. For every β, the respective mean entropies of the heuristic-matrices are calculated for a selection of TSP instances including four symmetric instances `gr48` (48 cities), `eil101` (101 cities), `d198` (198 cities), `rd400` (400 cities) and four asymmetric instances `ry48p` (48 cities), `ft70` (70 cities), `kro124p` (100 cities) and `ftv170` (171 cities). The heuristic entropy $H^\eta = -\sum_{i=1}^{i=n} p_i \cdot \log p_i$, with p_i denoting the relative frequency of item i, measures the uniformity of the distribution of items within

the heuristic-matrices. The maximum entropy value is reached for $\beta = 0$, i.e. all items in a matrix have the maximum diversity. For values $\beta \geq 1024$ the entropy converges to its minimum $H^{\eta}_{min} = 0$ for all TSP instances considered. This means all items in a matrix are identical. The vertical lines in Figure 4.49 represent the respective best β values found after executing P-ACO-TSH for 100000 iterations. The corresponding mean entropies are drawn as horizontal lines. All best mean entropies are located in the range $H^{\eta*} \in [18.2\% \cdot H^{\eta}_{max}, 69.4\% \cdot H^{\eta}_{max}]$. Entropy maxima are associated with small TSP instances `gr48` and `ft70`. Eliminating these instances constrains the best mean entropies to a smaller range $H^{\eta^*} \in [18.2\% \cdot H^{\eta}_{max}, 36.7\% \cdot H^{\eta}_{max}]$. Hence the P-ACO-TSH performs best, if heuristic values η_{ij} are weighted by $\beta > 1$ such that favorable items appear with a raised frequency. However, by choosing an appropriate value for β, the entropy of the heuristic-matrices should be restricted to a small range around $27\% \cdot H^{\eta}_{max}$. This entropy range corresponds to a weight $\beta \approx 5$, which is in accordance with desirable β-parameters in the standard ACO algorithm (see Section 2.3.2).

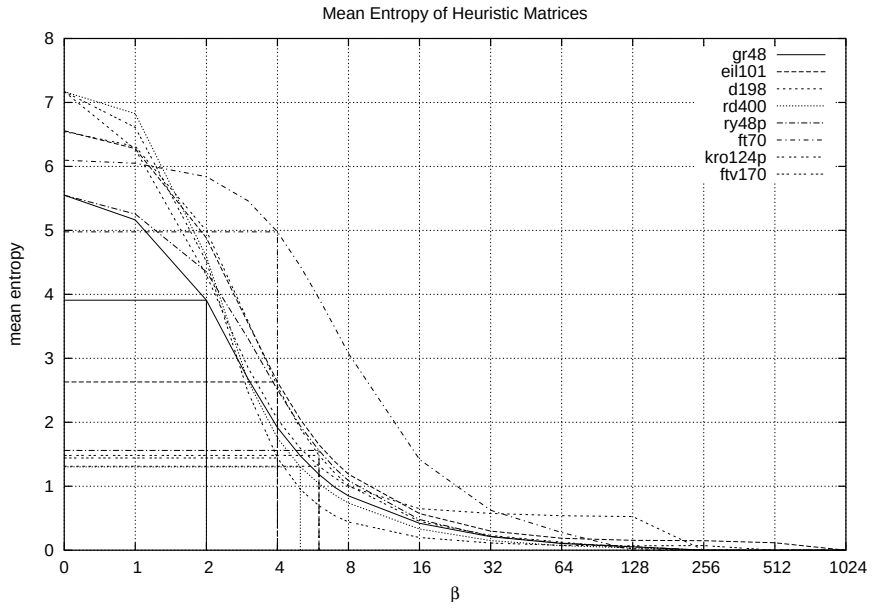


Figure 4.49: Mean entropy of heuristic matrices for different TSPs. Horizontal/vertical lines indicate best mean entropies/β-weights respectively.

In the final experiment, P-ACO is implemented with two kinds of heuristic guidance: the time-scattered and the standard heuristic. The latter is also tested in combination with candidate lists, which are a common technique to reduce the runtime for TSP [Reinelt, 1994]. The candidate list heuristic can be seen as an alternative to TSH, although it is not suitable for a hardware realization: For

every city i, the nearest neighbors are calculated and stored in a candidate list L_i of fixed size C_l. Using candidate lists, an ant located in city i can choose the next city j only from the candidate list. Only if none of these cities can be visited anymore ($S \cap L_i = \emptyset$) then j is randomly chosen from S with uniform distribution. Typically, the size of candidate lists is set to $C_l = 20$ [Johnson and McGeoch, 1995, Stützle and Dorigo, 1999].

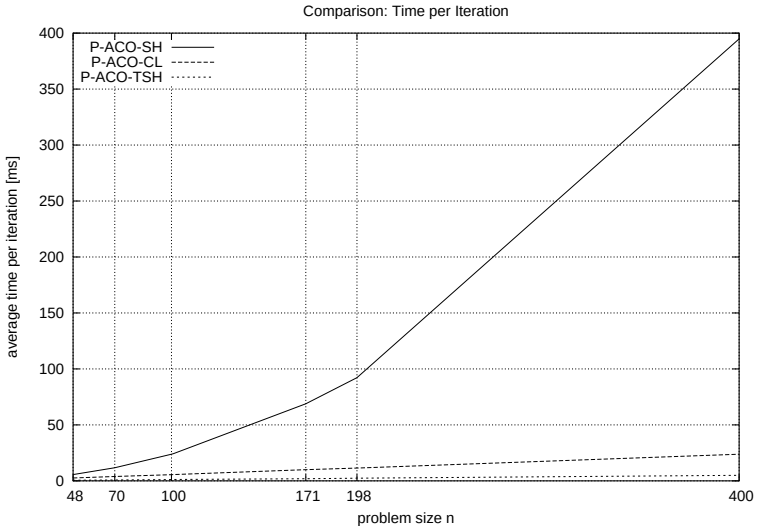


Figure 4.50: Time per iteration needed by the three tested heuristic approaches.

The parameter settings chosen to be equal for all three heuristic approaches are: $k = 4$, including one elitist solution, $m = 8$, $\tau_{min} = 1$, $f_P = 10$ and $\beta = 5$. For P-ACO-TSH apply the following additional parameters: $q_0 = 0.9$, $\eta_{min} = 1$, $f_H = 10$, $r = 48$, $c = 3$, $p_s = 4$ and random shift policy. The standard heuristic approach (P-ACO-SH) is started with $q_0 = 0.3$ and $\alpha = 1$, the candidate list approach (P-ACO-CL) with $q_0 = 0.8$, $\alpha = 1$ and $C_l = 20$. Note that the q_0 parameters are different, preliminary experiments have shown that the P-ACO-TSH and the P-ACO-CL approach benefit from a high degree of exploitation. All three approaches are studied on the same set of TSPLIB instances as used in the previous experiment with problem sizes ranging from $n = 48$ to $n = 400$ cities. Figure 4.50 shows for the heuristic variants under test, the average computation time needed to complete an iteration. The instances are sorted by their problem sizes, which are plotted on the x-axis. The results for $n = 48$ are calculated as the average over the values obtained for TSBLIB instances `gr48` and `ry48p`. Figure 4.50 illustrates that consistently, the time per iteration for P-ACO-TSH is smaller than for P-ACO-CL, and that throughout P-ACO-SH requires the highest

computation time per iteration. Therefore, the maximum total computation time per instance (equal to all three approaches) is chosen such that P-ACO-SH can run for 100000 iterations. For all TSPLIB instances considered, these values (TT) are listed in Table 4.3. Accordingly, the other two approaches (P-ACO-TSH and P-ACO-CL) can execute more iterations within the same time given. The results are shown as the average tour length over 20 repetitions with different random seeds.

Table 4.3: Comparison of TSH, standard heuristic and standard heuristic with candidate lists. Columns show the TSP instance, the total computation time (TT), the average tour length reached (AVG), and the standard error (S-ERR).

		P-ACO-TSH		P-ACO-SH		P-ACO-CL	
TSP	TT[s]	AVG	S-ERR	AVG	S-ERR	AVG	S-ERR
gr48	562.0	5099.45	7.15	5075.85	4.26	**5075.45**	4.18
eil101	2432.4	**645.40**	0.87	649.99	1.06	647.37	0.89
d198	9215.4	15942.83	18.19	**15927.80**	15.38	15970.00	11.33
rd400	39483.4	16553.78	124.86	15557.17	16.77	**15483.90**	14.91
ry48p	562.0	14547.70	20.24	14554.95	28.26	**14518.45**	18.46
ft70	1176.1	**39071.65**	58.76	39150.45	51.58	39189.65	49.97
kro124p	2377.9	**36836.50**	88.51	37057.75	119.80	37227.30	102.17
ftv170	6884.3	**2836.05**	9.38	2858.10	14.54	2866.70	15.97

Table 4.3 shows that in 50% of all TSP test instances, P-ACO-TSH receives the best average tour lengths. TSH performs well compared to the standard heuristic and the candidate list enhanced version in all cases but rd400. The reason for the relatively poor performance of P-ACO-TSH on this instance is probably due to the fact that r and p_s are chosen as constants over all instance sizes n. While the parameter values applied work very well for $n \approx 100$, for smaller ($n \leq 48$) as well as larger ($n \geq 198$) instances, they lead to inferior performance. Preliminary tests have shown that tuning r and p_s to the instance size improves the solution quality significantly and results in a similar performance as the other two heuristic variants.

P-ACO-TSH is constructed specifically with a hardware implementation in mind, which, judging from earlier results obtained in Section 4.4.4.2, runs notedly faster than the software implementation on a sequential CPU. Therefore, P-ACO-TSH has the potential to far outperform the sequential software implementation of P-ACO-SH as well as P-ACO-CL.

4.5 Comparison of Different Approaches

This section compares the previously introduced ACO approaches with each other. Common and differentiating algorithmic properties of the RMesh ACO (cf. Section 4.2), C-ACO, P-ACO and standard ACO are reflected. Furthermore, the asymptotic runtime and resource requirements are compared.

4.5.1 Algorithmic Properties

C-ACO as well as P-ACO replace the normal floating-point representation of pheromone values by integer numbers ranging in an interval between a certain minimum and maximum. This is very important for the economic usage of hardware resources as the maximum bandwidth for pheromone values or the size of match buffers can be exactly tailored during compilation. Another effect is that, in contrast to standard ACO and RMesh ACO, limiting pheromone values by a lower bound prevents an early (possibly unfavorable) convergence. Even though both algorithms, C-ACO and P-ACO, are able to rapidly attain good solution qualities, a complete convergence can hardly be guaranteed unless the global optimum has actually been found. The RMesh ACO represents pheromone values by floating-point numbers, but ant decisions are made based on low-bits and high-bits. Different from C-ACO and P-ACO, the aim of this transformation into discretized values is to accelerate the selection by fast bit-summation algorithms specialized on the dynamically reconfigurable bus.

C-ACO and P-ACO share the replacement of global evaporation by certain other techniques. C-ACO substitutes global evaporation for many local evaporation steps whenever items are selected, whereas P-ACO makes use of a combination of positive and negative population updates. Both techniques cause the effectual total pheromone sum per row to remain constant (except for the initial k iterations of P-ACO). The RMesh ACO, however, working on powerful processing elements provided by the computation model, applies the same conventional global evaporation methodology as standard ACO.

P-ACO is distinguished by its special way of storing pheromone information. Whereas RMesh ACO, C-ACO, and standard ACO keep a pheromone matrix to memorize and communicate preceding ant decisions, P-ACO maintains a population of recently constructed good solutions. Accordingly, P-ACO does not remind the history of past decisions by the development of pheromone values but rather by recent good solutions as the resulting consequence of earlier decisions. RMesh ACO and standard ACO control the influence of past ant decisions via evaporation rate ρ. As global evaporation is not implemented in the proposed alternative approaches, forgetting previous decisions is steered differently. In C-ACO, the choice of τ_{min}, τ_{max}, and the amount of pheromone decremented per

selection can be tuned accordingly. P-ACO controls the influence of old decisions by parameters τ_{min}, Δ_P, and population size k.

RMesh ACO, C-ACO and P-ACO follow different concepts of parallelization. Whereas RMesh ACO and C-ACO pipe ants in a systolic fashion through a statically allocated pheromone matrix (or vice versa in the case of C-ACO), P-ACO is based on the match buffer concept piping population items through parallel solution generators. In the broader sense, this strategy can also be considered as a form of piped pheromones. The P-ACO match buffer concept, however, is a special characteristic of P-ACO that allows to accelerated the selection process with no need for n parallel prefix sum circuits (as required by C-ACO).

4.5.2 Asymptotic Runtime and Resource Requirements

Table 4.4 compares the asymptotic runtime requirements of the parallel RMesh ACO, the parallel FPGA implementations of C-ACO and P-ACO, and the sequential software implementation of standard ACO. The attainable speedup is due to the pipelining capabilities and parallelism embodied in the hardware algorithms. Note that the time needed for arithmetic functions like additions and multiplications strongly depends on the actual hardware platform used and is therefore disregarded in Table 4.4. It is further supposed that in hardware the asymptotic time per evaluation step does not exceed the time per ant decision. This is certainly feasible for optimization problems like QAP, SMTTP, or TSP. Complexity figures in Table 4.4 refer to the algorithms without heuristic support. Furthermore, the amount of available hardware resources is assumed not to impose any restrictions. Methods for adjusting circuit dimensions to the limits of hardware resources are presented in Section 5.1.

Table 4.4: Comparison of asymptotic runtime requirements. Total time (TT) needed for the construction of z solutions is given for the fixed decision sequence (FDS) and cyclic decision sequence (CDS).

	RMesh ACO	**C-ACO**	**P-ACO**	**Standard ACO**
Time per decision	$O(\log^* n)$	$O(\log n)$	$O(k)$	$O(n)$
Time per solution	$O(n \cdot \log^* n)$	$O(n \cdot \log n)$	$O(n \cdot k)$	$O(n^2)$
Period per solution	$O(\log^* n)$	$O(\log n)$	$O(n \cdot k/m)$	$O(n^2)$
TT for z solutions (FDS)	$O((z+n) \cdot \log^* n)$	$O((z+n) \cdot \log n)$	$O(z \cdot n \cdot k/m)$	$O(z \cdot n^2)$
TT for z solutions (CDS)	$O(z \cdot \log^* n)$	$O(z \cdot \log n)$		
Speedup (CDS)	$O(n^2/\log^* n)$	$O(n^2/\log n)$	$O(n \cdot m/k)$	—

Exploiting the capabilities of the dynamically reconfigurable bus system, the RMesh ACO can make a decision in $O(\log^* n)$, which can be considered as con-

stant with respect to all problem sizes n with practical relevance. As outlined in the Section 4.3.4.3, the execution speed of the C-ACO algorithm is mainly determined by the calculation of prefix sums, which requires $O(\log n)$. As P-ACO can build-up a probability distribution using the match buffer concept, a decision can be made in $O(k)$. As mentioned before, experimental studies suggest that population size k can also be regarded as a small constant, which is independent of problem size n. In comparison, a sequential processor that uses the standard ACO approach requires $O(n)$ time per ant decision.

For all algorithms, each ant decision has to wait for the completion of the preceding ant decision. Thus the time needed for constructing a complete solution is the product of n and the time per ant decision. In RMesh ACO and C-ACO, ants can be piped through a statically allocated pheromone matrix such that a complete solution can be created with an average period equal to the time per ant decision. Considering P-ACO, the functional parallelism embodied in processing m ants in parallel on an FPGA allows m solutions to be formed in $O(n \cdot k)$ time such that an average period of $O(n \cdot k/m)$ per solution is attained. The sequential standard ACO, however, cannot support a pipelined flow of ants and therefore needs a quadratic period per solution construction. Assuming that the algorithms are terminated after a total of z solutions generated, the total runtime is the product of z and the average period per solution (assuming CDS for RMesh ACO and C-ACO). In the last row of Table 4.4, the asymptotic speedup of the parallel ACO variants over the sequential standard ACO algorithm are given. The speedup differences between RMesh ACO and C-ACO trace back to the specific methods for the calculation of prefix sums. The speedup for P-ACO depends on the number of parallel solution generators m and population size k. All three parallel variants achieve an asymptotic speedup of at least factor n (if k is a small constant).

Subsequently, the asymptotic resource requirements of the hardware-variants of ACO are compared. The RMesh-ACO occupies a grid of $O(n^2)$ processing elements (PEs). The size of a single PE in the abstract computation model is not further specified. However, sufficient resources must be available to compute the floating-point encoded pheromone information in each PE. The asymptotic amount of resources needed by C-ACO with piped ants are signified by the size of the quadratic pheromone matrix. Each pheromone value is encoded with a bandwidth of $\log(\tau_{min} + n \cdot \tau_{init})$ bits. Assuming τ_{min} and τ_{init} to be independently chosen from problem size n, a total of $O(n^2 \cdot \log n)$ resources are used. Regarding C-ACO with piped pheromones, the resource requirements of $O(M \cdot n \cdot \log n)$ are distinguished by the size of the ant array M. As outlined in Section 4.4.4.1, P-ACO requires resources in the order of $O(m \cdot n \cdot \log n)$ growing linearly in the number of solution generators. The length of each Solution Generator itself is proportional to problem size n, whereas the height is mainly determined by the

bandwidth $O(\log n)$ for encoding item numbers. Population size k is considered as a constant independent of problem size n. Therefore the population does not influence the asymptotic amount of resources needed. Note that the resource complexities stated above assume the remaining parts of the circuitry (e.g. control, comparison, evaluation) not to exceed the size of the constructive part of the algorithm. The evaluation functions of QAP, SMTTP, and TSP do definitely satisfy this assumption. It can be summarized that asymptotically, P-ACO requires less resources than C-ACO if m is smaller than n or M, respectively.

4.6 Summary

In this chapter, two hardware-oriented ACO variants, C-ACO and P-ACO, have been presented, which have shown to be well-suited for the implementation in fine-grained logic provided by FPGAs. In experimental studies, both variants have shown to be competitive substitutes to the standard ACO algorithm.

Research with both FPGA-oriented variants of ACO inspired various new algorithmic techniques with respect to:

Encoding of pheromone information: Experimental studies suggest that the alternative encoding of pheromone values in integer or in the form of population items does not deteriorate optimization performance in comparison to standard ACO. A new place-item-item encoding has been proposed for C-ACO, which supports pipelined solution construction such that conventionally item-item encoded problems can also be covered by the parallel FPGA implementation.

Modified selection procedures: In contrast to standard ACO, P-ACO uses a modified selection procedure. Applying the match buffer concept to P-ACO the selection procedure has been accelerated such that an ant decision can be executed in practically constant time.

Alternative heuristic support: The INTVAL, POTVAL, and τ-heuristics for C-ACO as well as the Time-Scattered Heuristic for P-ACO demonstrated, that it is possible to adequately approximate the original floating-point heuristic by integer values, which is better suited for fine-grained logic. All heuristic variants allow for fast calculations coupled with modest resource requirements. The Time-Scattered Heuristic further permits a straight forward extension to the P-ACO match buffer concept thereby maintaining the expedited P-ACO selection procedure.

Alternative decision sequencing: Empirical studies have shown that the standard fixed decision sequence is not necessarily the best choice. Randomly

sequencing ant decision certainly improves optimization performance but prohibits a pipelined flow of ants through the pheromone matrix. The recommended alternative cyclic decision sequence performs equally well and supports the systolic solution construction.

Systolic solution construction: Two variants of systolic solution construction, piped ants and piped pheromone, have been presented, both of them well-suited for the parallel execution of C-ACO.

Comparison of new solutions: The systolic flow of constructed solutions has motivated the non-generational mode of comparing solutions piped through a solution buffer. Appropriate settings for the number of predecessors and successors have been identified. Opposing this new approach to the conventional generational comparison mode, the non-generational variant has exhibited a competitive or even better optimization performance.

Concurrent pheromone update: Four different variants of parallel pheromone update have been compared in combination with both types of pipelining. Even though the concurrent matrix update intervenes with the parallel solution construction, a significant impact on optimization performance could not be detected. In the case of P-ACO, the update of the pheromone matrix is replaced by a corresponding population update.

Substitutes to global evaporation: Global evaporation, commonly applied in standard ACO, would afford a high number of multiplication circuits and thus tremendous chip resources on the FPGA. C-ACO substitutes global evaporation for a sequence of local evaporation steps during ant decisions, whereas in P-ACO, the negative population update corresponds to locally evaporating the associated pheromone values. For both variants of ACO, experimental studies suggest that waiving global evaporation does not critically affect optimization performance.

It has been shown that the parallel C-ACO implementation can achieve an asymptotic speedup of $O(n^2/\log n)$ over the sequential software implementation of standard ACO. The corresponding asymptotic speedup for P-ACO is $O(n \cdot m/k)$. In experimental studies comparing the FPGA implementation of P-ACO with the software implementation P-ACO on a CPU, speedup values between approximately 2 and 10 could be measured for various instances of SMTTP. Experimental FPGA implementations of P-ACO for SMTTP suggest that the amount of logic resources available on modern chip devices is sufficient to accommodate the circuitry for large applications with practical relevance.

Although C-ACO and P-ACO have been examined with the FPGA implementation in mind, these two ACO variants have been compared with standard ACO

in sequential software simulations. In these experiments, C-ACO and P-ACO have exhibited a competitive or even superior optimization performance. Therefore, these two hardware-oriented variants may also be attractive alternatives in software. Also the software implementation of some of the proposed algorithmic modifications, e.g., the match-buffer concept, the Time-Scattered Heuristic, the non-generational comparison mode, the cyclic decision sequence, or the place-item-item matrix encoding may offer attractive substitutes to the conventionally applied techniques.

Chapter 5

Concepts of Using Runtime Reconfiguration

The hardware-oriented variants of ACO introduced in the previous section demonstrate how the operations typically executed in software on sequential machines can be suitably modified for an accelerated execution on FPGAs. The achievable speedup is due to different data types, adapted algorithmic procedures, and certainly due to a high degree of concurrency. Reconfigurable architectures, however, offer the potential for further improvements beyond the typical acceleration induced by parallelism and pipelining. As outlined in Section 3.4, applying reconfiguration at runtime can lead to faster execution speed, reduced space requirements, less power consumption, and a higher degree of flexibility in comparison to conventional hard-wired logic.

In the following, various concepts are proposed to enhance the ACO hardware algorithms by the usage of runtime reconfiguration. One way of utilizing runtime reconfiguration, sharing a common population among multiple solution generators, has already been presented for the P-ACO circuit (cf. Section 4.4.3.5). The usual approach to apply runtime reconfiguration is to identify parts of the circuit which are temporarily not required and to load these parts as they are needed by the algorithm. Considering C-ACO, the whole circuit is permanently used for the parallel construction and evaluation of new solutions (except for the phase of initial solution constructions after start-up). Only the S-arrays of the P-ACO circuit offer a possibility of sharing chip resources by partial reconfiguration as outlined later.

For the ACO implementation on FPGA, three different applications of runtime reconfiguration are proposed beyond the traditional technique of purely sharing logic and routing resources:

- The circuit is too large to entirely fit onto the available resources. The

hardware design needs to be partitioned and scheduled.

- The optimization problem is subject to dynamic changes during the runtime of the algorithm. The circuit is required to efficiently adapt to these changes.
- Runtime reconfiguration is applied to accelerate the convergence and execution speed of the algorithm.

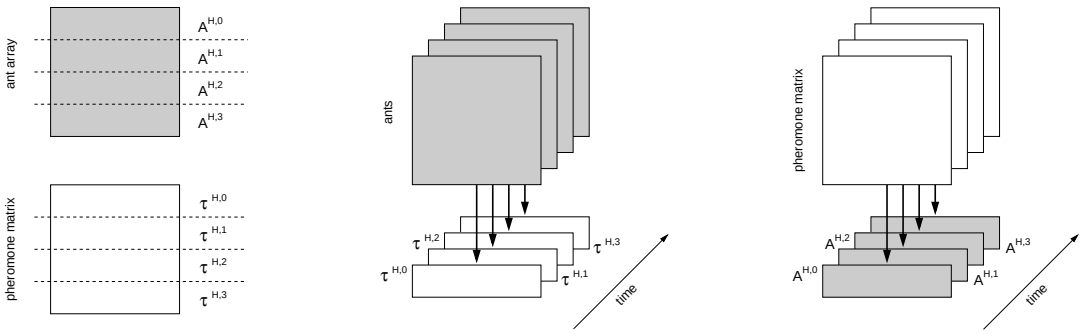
5.1 Partitioning and Scheduling

In the case of very large problem instances (or comparatively small chips), it may happen that the dimensions of the ACO circuit exceed the available amount of resources on a single FPGA device. There exist mainly two alternatives to cope with this problem: either spatial partitioning over multiple FPGA devices on a multi-FPGA system or temporal partitioning combined with an appropriate execution schedule. Various tools and techniques for the automated partitioning [Hauck, 1995, Acock and Dimond, 1997, Vahid, 1997, Brasen and Saucier, 1998, Khalid, 1999, Hidalgo et al., 2003] and routing [Mak and Wong, 1997, Ejnioui and Ranganathan, 1999] of multi-FPGA systems are available. Therefore the aspect of spatial partitioning shall not further be considered. The emphasis is put on the temporal partitioning of ACO algorithms, i.e. the circuit is sub-divided into several sequentially executed partitions.

5.1.1 Counter-based ACO

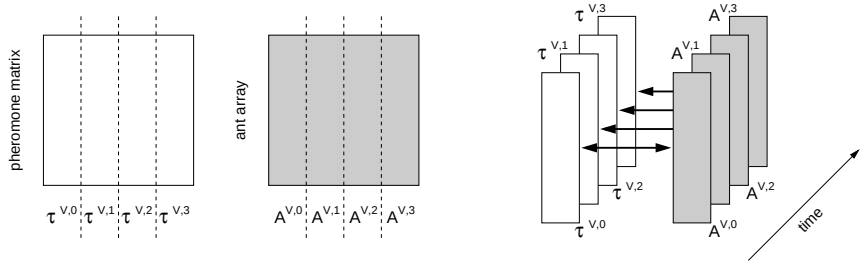
With respect to C-ACO, standard techniques for temporal partitioning can be applied. It is supposed that the cells of the pheromone matrix (in the case of piped ants) or the ant array (in the case of piped pheromones) are placed in rows and columns forming a rectangular circuitry, which can be forced by specific constraint definitions.

Dividing the pheromone matrix horizontally yields p_h partitions $\tau^{H,i}$ with $i \in \{0, \ldots, p_h - 1\}$ as visualized in Figure 5.1a. Accordingly, a horizontal partitioning of the ant array produces p_h sub-arrays $A^{H,i}$. Considering C-ACO with piped ants, only one of the p_h partitions is configured at a time (see Figure 5.1b). After one partition $\tau^{H,i}$ has processed a sequence of m ants, partition $\tau^{H,i+1}$ is configured processing the same sequence of ants again. This procedure is repeated until the last partition has processed all ants and a total of m solutions has been constructed. Correspondingly, the partitions $A^{H,i}$ of the ant array can be configured sequentially each of them processing an entire pheromone matrix as shown in Figure 5.1c.



(a) cut lines (b) schedule PA (c) schedule PP

Figure 5.1: Horizontal division of ant array or pheromone matrix into $p_h = 4$ partitions. Configuration schedules for parallel execution with piped of ants (PA) or piped pheromone (PP).



(a) cut lines (b) schedule

Figure 5.2: Vertical division of ant array and pheromone matrix into p_v partitions. Configuration schedule for parallel execution for both pipelining variants: piped ants (PA) and piped pheromone (PP).

The circuitry can also be divided vertically into p_v partitions (see Figure 5.2a). The partitions of the pheromone matrix are denoted by $\tau^{V,i}$ with $i \in \{0, \ldots, p_v - 1\}$. Accordingly, p_v partitions $A^{V,i}$ of the ant array are created. Depending on the type of pipelining, either partitioned ants are processed on the currently configured partition of the pheromone matrix or vice versa (see Figure 5.2b). All n ants are evenly distributed over the pheromone matrix, one ant per row. After processing the complete sequence of p_v partitions, the prefix sums describing the selection probabilities have been determined. The selected items are identified during the next sequence of processed partitions. Afterwards,

pipelined data is cyclically shifted one row forward to continue with the next decision. This kind of scheduling ant decisions is equivalent to the cyclic decision sequence (CDS) introduced in Section 4.3.6.3.

Overall, partitioning the circuit permits mapping large problems instances onto the available resources, which would otherwise not suffice to accommodate the complete circuit. Applying one of both types of partitioning, horizontal or vertical, potentially needs about p_h or p_v times less logic on the chip device than the mapping of the entire circuit. However, the circuit of a partition needs to provide more storage to communicate between different partitions and to allow for a proper execution. Furthermore, scheduling a partitioned circuit requires a more intricate control logic. Compared to the entirely mapped algorithm, temporal partitioning reduces the degree of pipelining resulting in a reduced throughput and an increased computation time. Moreover, the increased number of memory accesses may cause an overhead further reducing execution speed. Especially when using external memory devices instead of embedded memory blocks, long communication delays have to be expected. In this case, techniques to reduce the amount of transmitted data or to overlap communication with computation need to be investigated.

5.1.2 Population-based ACO

Based on earlier publications by the author of this thesis [Diessel et al., 2002, Scheuermann et al., 2004b], this section presents various approaches for partitioning and compacting the P-ACO circuit. The dimensions of the circuit mainly depend on the size of the problem instance implemented. Specifically, the height of the P-ACO circuit layout increases only logarithmically with n whereas the width grows linearly, resulting in an increasingly flat rectangle shape as a basic structure. Then, fitting the algorithm can be accomplished by folding, which is standard technique for this kind of problem.

For a given problem size n, the available resources might constrain the number of solution generators that can be implemented on the FPGA device. It is proposed to implement only $m' < m$ Solution Generators operating in c cycles, where $m = c \cdot m'$ (see Figure 5.3). In every cycle, m' solutions are generated in parallel. After each cycle the newly created solutions are compared with each other, and the best solution of the current cycle is compared with the best solution of all preceding cycles. Comparison takes t_{cmp} time, and can be done while the next solution is being generated. Thus the time to finish a complete iteration is $t_{it} = \max\{c \cdot t_g + t_{cmp}, c \cdot t_{cmp} + t_g\} + t_u$ with t_g denoting the time for generating a complete solution and t_u the time per population update (see Section 4.4.4.2).

Another method to make better use of the available space takes into account that the size of selection set S decreases over time. At certain points in time it is

possible to move the currently active selection sets to solution generators containing S-arrays with a smaller number of S-cells. Two examples of this modification are given in Figure 5.4.

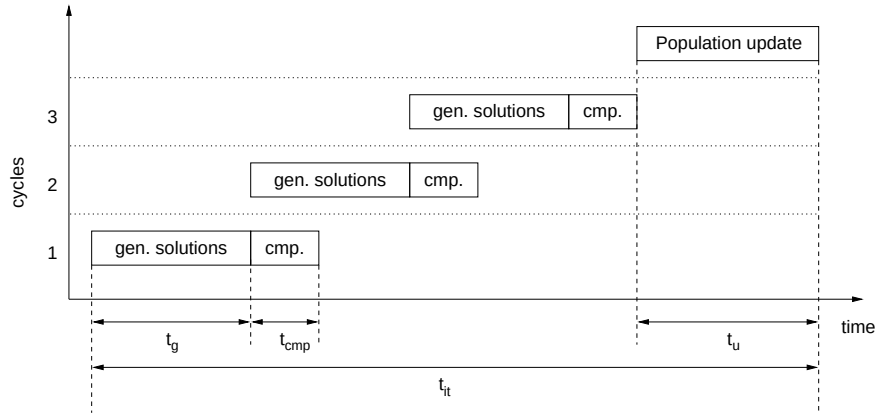


Figure 5.3: Schedule of a complete iteration consisting of $c = 3$ cycles. After the last cycle the population is updated. Here only one solution generator is considered.

The two/four S-Array configurations in figures 5.4a/b save 25/37.5% space at the cost of having the simulated ants be in different stages of completion due to their respective starting delays. In both configurations, the left side shows the S-arrays after a new ant has started, the right side the moment just before the selection sets are shifted down and a new ant starts.

(a) Two S-arrays (b) Four S-arrays

Figure 5.4: Examples of S-Array configurations with decreasing array sizes. Shaded area represents the active portion of the array holding set S.

Common to all partitioning and scheduling concepts presented so far is that the size of the circuit is reduced either by sequentially executing only a small partition of the circuit or by adequately shifting data. This means that following a certain schedule, the actual reconfiguration is merely restricted to the manipulation of data stored in working registers. Certainly, reconfigurable architectures providing various kinds of memory resources distributed over the computational array facilitate the implementation of partitioned circuits. However, applying

runtime reconfiguration also to configuration memory makes even better use of the capabilities of reconfigurable architectures. Therefore, the next proposal also includes reconfiguration of logic and routing structures at runtime.

As mentioned before, the size of the selection set is steadily reduced during the process of solution construction, and the logic of the S-Array is designed to compact the active S-cells to the right hand side. This means that on average, only 50% of the S-Array is actually used. It is proposed to divide every S-Array in the middle into two equally sized sub-arrays, one of them being static, the other reconfigurable during runtime (see Figure 5.5). Furthermore, a different type of mirrored S-Array is needed shifting active S-cells to the left hand side. Mirrored S-arrays are also divided into two sub-arrays. All m S-arrays are arranged pairwise, such that the left column contains the mirrored arrays and the right column comprises the normal arrays. Every pair is configured in a way that the reconfigurable sub-arrays overlap.

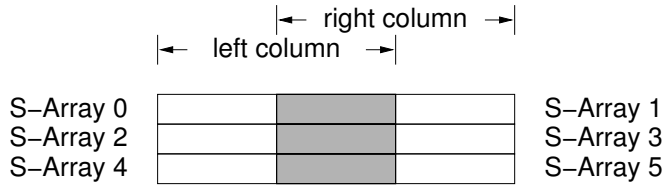


Figure 5.5: Example of $m = 6$ S-arrays arranged in two columns. Shared sub-arrays (shaded) are reconfigured at runtime.

The parallel solution construction is scheduled as follows: The $m/2$ arrays in the right column start generating solutions. After $n/2$ decisions the shared sub-arrays contain only deactivated S-cells. This area can now be partially reconfigured into a left-shifting sub-array. After reconfiguration is finished, the mirrored S-arrays commence and the normal S-arrays continue construction solutions. After $n/2$ further ant decisions the shared area is reconfigured into right-shifting sub-arrays and the process starts over again. Disregarding reconfiguration delays, the average throughput of solutions generated by the circuit is equal to the non-runtime reconfigurable implementation. The real increase in computation time depends of the delay induced by partial reconfiguration. However, by sharing reconfigurable sub-arrays the requirements of logic resources can be reduced by 25%. Furthermore, as the reconfigurable area is aligned in a column, the proposed design is well-suited for Xilinx Virtex architectures supporting only column-wise reconfigurations (see Section 3.4.1).

5.2 Dynamic Changes of the Optimization Problem

All previously regarded optimization problems can be considered as static in as far as the problem-specific parameters do not change during the algorithm runtime. In this section, however, it is supposed that some parameters are subject to dynamic changes, which occur during the optimization process and are not known prior to starting the algorithm. In the following, concepts of dynamic reconfiguration are applied to TSP. By suitable modifications, however, these concepts can also be transferred to other dynamically changing optimization problems. Two possible cases of dynamic changes are distinguished:

- The size of the problem instance changes over time by the deletion or insertion of cities.
- Other problem parameters change during the algorithm runtime.

5.2.1 Dynamic Changes of the Problem Size

In the first case, the problem size changes during the optimization process. It is supposed that the algorithm has already generated tours through the set of n cites when a change of the problem size occurs. Some cities shall not be visited anymore and are deleted from the problem instance while other new cities to be visited are inserted into the problem instance. The ACO algorithm is now required to adapt to these changes and to find good tours through the new set of cities.

One way to react to these changes would be to restart the algorithm from scratch giving the altered problem instance as input. This would be equal to discarding all previously made calculations and all solutions found for the old problem instance. Accordingly, the hardware implementation would need to be completely redesigned, synthesized, and implemented on the FPGA, which would afford an unacceptable amount of time.

If the change to the problem instance, however, is not too severe and these changes do not occur too often, it may be beneficial to re-use the knowledge gained by the optimization on the old problem instance. With respect to the ACO algorithm, this means to appropriately modify the pheromone matrix as well as the previously found elitist solution and to continue the optimization process.

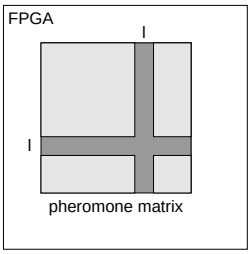
These modifications are accomplished in three steps:

1. Adapting the distribution of pheromones

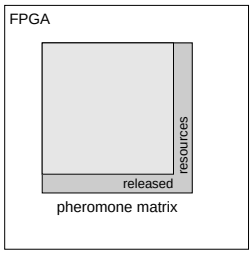
2. Adjusting the size of the pheromone matrix

3. Repairing the elitist solution.

Various strategies for adapting the distribution of pheromones and for repairing elitist solutions in ACO algorithms are examined in [Guntsch, 2004]. As these two steps do not require the capabilities of runtime reconfiguration, the subsequently proposed concept deals with adjusting the size of the pheromone matrix to dynamic changes of problem size n. Whenever a city l is deleted from the problem instance, the corresponding row l and column l of the pheromone matrix can be removed. Assuming the ACO algorithm to be implemented with a statically allocated pheromone matrix with piped ants, this would mean to mark the respective cells in row l and column l as deleted (see Figure 5.6a). Thereafter the remaining circuit can be compacted to the top-left by shifting pheromone values in rows $i > l$ one row up and all pheromones values in columns $j > l$ one column to the left thereby overwriting data stored in row and column l. A compaction towards the bottom-right corner can be accomplished likewise. The released resources along the borders of the pheromone matrix are available to other applications and can be partially reconfigured at runtime (see Figure 5.6b).



(a) Row and column to be deleted.



(b) After deletion and compaction.

Figure 5.6: Example of TSP with city l being deleted from the problem instance. Row l and column l are deleted, and the remaining pheromone matrix is compacted.

Accordingly, additional resources are needed if a new city $n + 1$ is inserted into the instance. These resources can be provided by partially reconfiguring one row and one column along the border of the circuit as shown in Figure 5.7.

Note that shifting and adapting pheromone values temporarily interrupts the optimization process. Thus fast pheromone modification algorithms and quick shifting structures are required. Depending on the capabilities of the target architecture, the actual reconfiguration of released or required resources can be

(a) Row and column to be appended.

(b) After reconfiguration

Figure 5.7: Example of TSP with city $n + 1$ being appended to the problem instance.

executed without interference with the currently running algorithm. It should therefore be possible to overlap the comparatively slow partial reconfiguration with useful computations.

Deleting and inserting cities causes a corresponding size adaption of the pheromone matrix, but also the bandwidths of certain numbers may change. For instance, item numbers (city indices) change logarithmically with the problem size, and pheromone values or update counters can take different maximum values. Thus, the initial circuit should be designed for the maximum possible problem size since adapting the circuit to a larger than the reserved bandwidth would possibly result in a reconfiguration or redesign of the entire circuit.

It should also be noted how the modifications to the pheromone matrix are sequenced. In the case of deleted cities, the distribution of pheromone values would first be adapted and shifted prior to reconfiguring the released resources. Considering the insertion of cities, the pheromone matrix would first be expanded by partial reconfiguration before modifying the pheromone values.

5.2.2 Dynamic Changes of Problem Parameters

Previously regarded dynamic changes concerned the size of the problem instance, i.e. the insertion or deletion of cities, but the algorithm also needs to react to changes of other problem parameters. In the case of TSP, for instance, the distance d_{ij} between two cities i and j may change due to a construction site or a traffic jam on the shortest route connecting these two cities. Accordingly, the traveling salesperson must follow a deviation such that the distance between the two cities is increased. This distance directly affects the heuristic information $\eta_{ij} = 1/d_{ij}$. It is supposed that η_{ij} is integrated into the calculation of selection probabilities by multiplying heuristic values with pheromone values (e.g. using

the INTVAL heuristic).

If distance changes do not occur too often reconfigurable architectures like FPGAs allow to realize these multiplications by constant coefficient multiplier circuits as introduced in Section 3.4.2. These multipliers are synthesized for constant heuristic values, they afford less logic resources, run at faster speed and use less power than conventional multipliers with two variable operands. With every change of distance d_{ij} the multiplier in cell (i, j) is altered by partially reconfiguring the new value η_{ij} as a constant coefficient as demonstrated in Figure 5.8. Like other partially evaluated circuits, constant coefficient multipliers need to account for the maximum possible bandwidth (see Section 5.2.1). Hence, every cell of the pheromone matrix circuitry should reserve sufficient area for the largest possible constant coefficient (see [MacKay and Singh, 1999]) to prevent a potential redesign of the entire circuit.

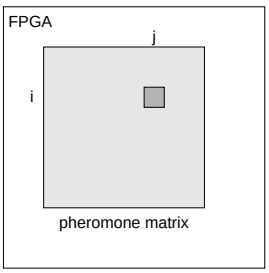


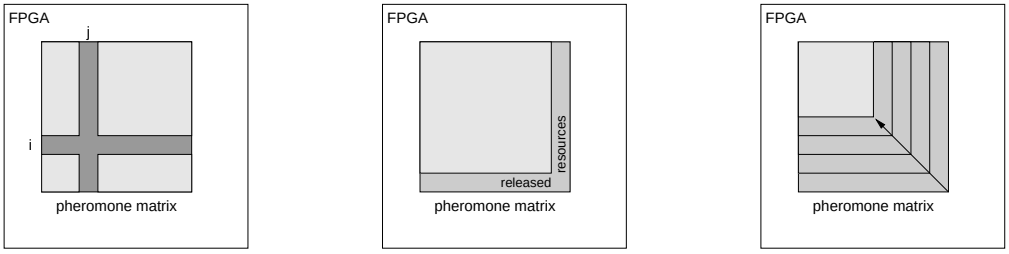
Figure 5.8: Example of a circuit of a statically allocated pheromone matrix with a cell (i, j) affected by a dynamic change of a problem parameter. This cell is subject to a partial reconfiguration at runtime.

Note that constant coefficient multiplications with heuristic values are only one example of accelerating the execution of ACO by means of partial evaluation (see Section 3.4.2). The concept of partial evaluation can be applied in all locations of the ACO circuit working on constant or infrequently changing values. Especially the circuits for the calculation of solution qualities are appropriate candidates for the usage of partial evaluation. These circuits are specialized on, e.g., constant flows between facilities (QAP) or constant weights of items (knapsack problem).

Reconfiguration at runtime offers the possibility to adapt the hardware to infrequent changes of these constants. If parameter changes occur rather frequently, however, it may be advisable to use conventional circuits and to modify register contents since runtime reconfiguration often requires prohibitive long time (cf. Section 3.4). The identification of critical reconfiguration delays or frequencies of parameter changes may be a direction for future work.

5.3 Accelerating Convergence and Runtime

As described in Section 4.2, dynamic reconfiguration can be applied to the RMesh implementation of ACO whenever a pheromone value τ_{ij} exceeds a high threshold value γ [Janson et al., 2002, 2003]. This local convergence would cause the ants to almost always select item j in row number i. The authors therefore propose to permanently assign item j as a constant value to place number i in the solution vector and the respective row and column is deleted. This concept, however, is not restricted to the RMesh, Figure 5.9 shows the application of size reduction steps to the ACO implementation on FPGA. After every size reduction, the resources of one row and one column are released. These resources are partially reconfigured to be re-used by other applications. As size reductions are repeatedly conducted with every detection of a local convergence, the initially occupied chip area can be released successively. Depending on the respective stopping condition, size reduction steps can be applied until eventually the entire circuit is freed.



(a) Row and column to be deleted

(b) After deletion and compaction

(c) Repeated size reductions

Figure 5.9: Example of ACO on FPGA with local convergence in τ_{ij} resulting in deletion of row i and column j. Eventually, repeated size reduction steps can release all logic resources.

Software simulations on QAP presented in [Janson et al., 2002, 2003] show that size reductions lead to an accelerated convergence and that the choice of threshold value γ controls a trade-off between execution speed and optimization performance: On the one hand, the reduced problem size results in a decreased runtime. On the other hand, the attainable solution quality is worsened in comparison to ACO without size reduction. Simulations on sparse QAP instances, however, behave somewhat different: Convergence is accelerated combined with a reduced runtime and even improved attainable solution qualities.

The RMesh model used in these simulations does not consider any reconfiguration delays. Regarding FPGAs, these reconfiguration delays can be very critical

and possibly compensate the advantages of the successively reduced problem size. Further examinations can therefore determine the conditions (e.g. maximum reconfiguration delay) for a successful application of the size reduction concept.

5.4 Summary

In this chapter, various concepts have been presented for applying runtime reconfiguration to the ACO implementation on FPGAs. It has been stated that the ACO hardware variants offer only a few possibilities for using the standard technique of loading and configuring circuit parts as they are needed by the algorithm. However, three different scenarios for the application of runtime reconfiguration have been identified: Temporal partitioning and scheduling, dynamic changes to the optimization problem, and the acceleration of convergence and execution speed.

Temporal partitioning and scheduling offers a way of mapping large problem instances onto the available resources, which would otherwise not be sufficient to accommodate the complete circuit. Under some circumstances, this technique may require more storage, need a more elaborated control logic, and lead to an increased computation time. In the case of unused circuit parts, however, compaction and scheduling can help saving chip resources with potentially less impact on execution speed (as proposed for P-ACO).

Runtime reconfiguration can also be used to adapt the ACO circuit to dynamic changes of the optimization problem, thereby avoiding an entire re-design and re-implementation, which would afford an unacceptable amount of time. It has been described how to react to dynamic changes of the problem size and of other problem parameters. Runtime reconfiguration can be applied to adjust the size of the pheromone matrix and to alter parts of the circuit by means of partial evaluation (e.g. using constant coefficient multipliers).

Finally, it has been shown how the concept of size reduction steps, originally proposed for the RMesh ACO, can be transferred to the ACO implementation on FPGAs. Successively, these size reductions release chip resources, which can be reconfigured at runtime to place other circuits on the chip area.

The major drawback of runtime reconfiguration are the extensively long times currently needed to load the configuration registers. Although there exist several approaches to reduce reconfiguration delays, it is very important to appropriately schedule circuit reconfigurations such that interruptions to the implemented ACO algorithm can be minimized (see Section 3.4.3). The extent and the duration of circuit reconfigurations can also be reduced by designing the layout for the maximum possible bandwidths, although choosing these upper bounds too high would restrict the maximum problem size that can be implemented.

Chapter 6

Conclusion

Concluding this thesis, the summaries and results of the preceding chapters are reviewed, and directions for further study are given.

6.1 Review of Summaries and Results

The scope of this thesis has been to examine the potential of runtime reconfigurable architectures for the efficient execution and acceleration of ACO algorithms. FPGAs as the effectively only commercially available runtime reconfigurable fabrics have been chosen as target platform. As fine-grained architectures FPGAs provide a very versatile and flexible programmability, but the implementation of certain data types and arithmetic operations typically needed by the standard ACO algorithm would afford a large amount of logic and routing resources. Therefore, alternative hardware-oriented variants of ACO, namely C-ACO and P-ACO, have been proposed, which have shown to be well-suited for the implementation on FPGA.

For both ACO variants, new algorithmic techniques have been developed and examined in a range of experimental studies. These algorithmic techniques refer to:

- encoding of pheromone information,
- modified selection procedures,
- alternative heuristic support,
- alternative decision sequencing,
- systolic solution construction,
- comparison of new solutions,

- concurrent pheromone update, and
- substitutes to global evaporation.

The algorithmic modifications allow to speed-up the execution by exploiting the parallelism and pipelining features of hardware algorithms. Furthermore, they suitably support the implementation on fine-grained logic devices like FPGAs. The results obtained in a series of empirical studies indicate that the proposed modifications achieve a competitive or even superior optimization performance.

The two ACO variants can attain asymptotic speedup values of at least factor n (where n is the problem size) over the sequential software implementation of standard ACO. P-ACO has been implemented in hardware on an FPGA and in software on a sequential CPU. Tested on various instances of SMTTP, speedup values between approximately 2 and 10 could be attained (note that these are the speedup values measured, which may differ from the asymptotic speedup). Scaling the circuit for different problem sizes n and number of ants m, it has been shown that the amount of logic resources available on a modern FPGA chip is sufficient to accommodate the P-ACO algorithm for large applications with practical relevance.

Experimental studies suggest that both variants can be considered as competitive substitutes to the standard ACO algorithm. As these experiments have been performed in sequential software simulations, the two hardware-oriented variants may also be of interest as alternative sequential ACO algorithms in software. Furthermore, the software implementation of certain algorithmic modifications (e.g. the match-buffer concept or the Time-Scattered Heuristic) offer an improved asymptotic runtime; other modifications like the non-generational comparison mode, the cyclic decision sequence, or the place-item-item matrix encoding have achieved competitive or even better average solution qualities and may therefore offer attractive alternatives to the conventionally applied techniques.

Various concepts for the application of runtime reconfiguration have been proposed. These concepts cover: temporal partitioning and scheduling, dynamic changes to the optimization problem as well as accelerating convergence and execution speed.

If the available resources on a chip are not sufficient then temporal partitioning, circuit compaction, and scheduling may help reducing the circuit size. These techniques can produce an increased demand of memory resources and a reduced throughput along with longer computation times. With respect to the P-ACO circuit, however, it has been explained, how compaction and scheduling can be used to reduce the amount of required chip resources with potentially less impact on execution speed. Furthermore, it has been outlined, how runtime reconfiguration can be applied to appropriately react to dynamic changes to the

optimization problem. Runtime reconfiguration allows to dynamically adapt the circuit dimensions whenever the problem size is changed. Partial evaluation can be applied to design small and fast circuits, which are reconfigured at runtime in order to react to infrequent changes of certain problem parameters. Runtime reconfiguration can also be used to shrink the size of the circuitry whenever the problem size is reduced in the case of a locally converged pheromone matrix.

6.2 Directions for Further Study

So far all experimental studies with C-ACO have been performed in software simulations, which is entirely sufficient to compare the different algorithmic techniques proposed. However, implementing C-ACO in hardware would allow to measure the actual speedup attainable. Moreover, the FPGA implementations of both ACO algorithms, P-ACO and C-ACO could be compared with each other. These implementations may also be extended by local optimizers or by the alternative heuristics that have been introduced.

Furthermore, it might be worthwhile to compare the hardware implementations of ACO with several kinds of parallel software variants (cf. Section 2.5.2). Creating a hybrid hardware-software implementation of ACO, it would be possible to accelerate computationally expensive parts of the algorithm in hardware, while other conditional branches or mainly sequential instructions are executed on a CPU. Tools for the assisted or automatic hardware-software partitioning are proposed, e.g., in [Gupta and De Micheli, 1993, Balarin et al., 1997].

In this thesis, the description of the hardware-oriented ACO algorithms has been restricted to optimization problems with solutions represented by permutations of item numbers. Most ACO applications in literature have been proposed for this class of optimization problems, although several other non-permutation problems have also been covered (see Section 2.4.1). With respect to the hardware implementation, non-permutation problems may allow to investigate other representations of pheromone information or different pipelining strategies.

In Chapter 5, various concepts of applying runtime reconfiguration to ACO have been described. Regarding current FPGA technology, dynamically changing parts of a circuit demands long reconfiguration times, which may degrade the efficiency of the proposed concepts. Implementing and testing these concepts in reconfigurable hardware would help identifying critical reconfiguration delays. At present, runtime reconfiguration of FPGAs is still a maturing field, and some devices offer a very limited dynamic access to configuration registers (cf. Section 3.4.1). Therefore, the examinations of the potential of runtime reconfiguration should not be constrained to current FPGA technology but should also consider less restrictive computational models.

With respect to the changes of the problem size, which may occur in dynamic optimization problems, new pheromone modification strategies need to be developed. These strategies should efficiently use the fine-grained resources on field-programmable devices. The parallel implementation based on local information may considerably accelerate the execution compared to the conventional software-based techniques.

Design, implementation, and verification of hardware ACO algorithms is still a very time-consuming procedure, which may far exceed the time needed for the development of ACO algorithms in software. Therefore, the creation of a hardware library of ACO modules coupled with an intuitive user interface would support the automatic (or semi-automatic) generation of ACO circuits and speed-up the development process. Such a library-based system could be realized as an intellectual property (IP) core. Alternatively, the ACO algorithm could be described in a higher-level programming language, which is then compiled including the binding of modules from a pre-compiled hardware library (such a library-based system is proposed, e.g., for Genetic Algorithms [Bland and Megson, 1998b,c]).

Automated circuit design is also the aim of many researchers in the field of Evolvable Hardware (EHW) (see [Gordon and Bentley, 2002] for an overview). Different from traditional design methodologies, circuits are generated by means of evolutionary techniques, mainly Evolutionary Algorithms. In [Coello Coello et al., 2000], it is shown that EHW can also be generated by means of ant-based optimization. Accordingly, it may be possible to apply hardware-implemented ACO algorithms to the automated circuit generation. Integrating both, the optimization algorithm and the evolved circuitry, on a runtime reconfigurable device would bring the self-(re-)configuring and self-adaptive chip one step closer.

Bibliography

A. L. Abbott, P. M. Athanas, L. Chen, and R. L. Elliott. Finding lines and building pyramids with SPLASH 2. In *[Buell and Pocek, 1994]*, pages 155–163, 1994.

M. Abramovici and J. T. de Sousa. A SAT solver using reconfigurable hardware and virtual logic. *Journal of Automated Reasoning*, 24(1/2):5–36, 2000.

M. Abramovici and D. G. Saab. Satisfiability on reconfigurable hardware. In *[Luk et al., 1997a]*, pages 448–456, 1997.

D. Abramson, P. Logothetis, A. Postula, and M. Randall. FPGA based custom computing machines for irregular problems. In *Proceedings of the 4th International Symposium on High Performance Computer Architecture (HPCA)*, pages 324–333, 1998a.

D. Abramson, K. Smith, P. Logothetis, and D. Duke. FPGA based implementation of a hopfield neural network for solving constraint satisfaction problems. In *Proceedings EuroMicro Workshop on Computational Intelligence*, pages 688–693, 1998b.

J. Ackermann, U. Tangen, B. Bödekker, J. Breyer, E. Stoll, and J.S. McCaskill. Parallel random number generator for inexpensive configurable hardware cells. *Computer Physics Communications*, 140(3):293–302, 2001.

S. J. B. Acock and K. R. Dimond. Automatic mapping of algorithms onto multiple FPGA-SRAM modules. In *[Luk et al., 1997a]*, pages 255–264, 1997.

Actel Corp. ProASIC PLUS flash family FPGAs v4.1, 2004.

Y. Adachi, K. Ishikawa, S. Tsutsumi, and H. Amano. An implementation of the Rijndael on async-WASMII. In *[Asada and Fujita, 2003]*, pages 44–51, 2003.

G. C. Ahlquist, B. E. Nelson, and M. Rice. Optimal finite field multipliers for FPGAs. In *[Lysaght et al., 1999]*, pages 51–60, 1999.

M. Alderighi, E. L. Gummati, V. Piuri, and G. R. Sechi. A FPGA-based implementation of a fault-tolerant neural architecture for photon identification. In *[Ebeling and Cong, 1997]*, pages 166–172, 1997.

A. Alex, J. Rose, R. Isserlin-Weinberger, and C. Hogue. Hardware accelerated novel protein identification. In *[Becker et al., 2004]*, pages 13–22, 2004.

Altera Corp. Excalibur device overview, 2002.

Altera Corp. FLEX 10K embedded programmable logic device family, 2003.

Altera Corp. Stratix II device handbook, 2004.

Altera Corp. Nios II processor reference handbook, 2005.

A. Amira, A. Bouridane, and P. Milligan. Accelerating matrix product on reconfigurable hardware for signal processing. In *[Brebner and Woods, 2001]*, pages 101–111, 2001.

J. Andrejas and A. Trost. Reusable DSP functions in FPGAs. In *[Hartenstein and Grünbacher, 2000]*, pages 456–461, 2000.

E. Angel, E. Bampis, M. Rahoual, H. Bouchema, and C. Dhaenens. Ants metaheuristic and constraint programming cooperation for the protein folding problem. In R. F. Hartl et al., editors, *Abstract Proceedings of the Sixth Metaheuristics International Conference (MIC 2005)*, 2005.

Annapolis Microsystems Inc. Wildfire reference manual, 1998.

C. Aporntewan and P. Chongstitvatana. Hardware implementation of the compact genetic algorithm. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pages 624–629, Seoul, South Korea, 2001.

Arbeitsgruppe Organic Computing. VDE/ITG/GI-Positionspapier. Organic Computing. Computer- und Systemarchitektur im Jahr 2010. 2002.

M. Arias-Estrada and J. M. Xicotencatl. Multiple stereo matching using an extended architecture. In *[Brebner and Woods, 2001]*, pages 203–212, 2001.

J. M. Arnold and K. L. Pocek, editors. *Proceedings of the 1995 IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, USA*. IEEE Computer Society, 1995.

S. Arora. Polynomial time approximation schemes for the euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

K. Asada and M. Fujita, editors. *Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology (FPT)*. The University of Tokyo, Japan, IEEE Computer Society, 2003.

P. Athanas and K. L. Pocek, editors. *Proceedings of the 1995 IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, USA*. IEEE Computer Society, 1995.

Atmel Corp. AT40K05/10/20/40AL complete datasheet, 2004.

J. Babb, R. Tessier, M. Dahl, S. Hanono, D. Hoki, and A. Agarwal. Logic emulation with virtual wires. *IEEE Transactions on Computer Aided Design*, 1997.

J. W. Babb, M. I. Frank, and A. Agarwal. Solving graph problems with dynamic computation structures. In J. Schewel, editor, *High-Speed Computing, Digital Signal Processing, and Filtering Using reconfigurable Logic, Proceedings SPIE 2914*, pages 225–236, Bellingham, WA, 1996. SPIE – The International Society for Optical Engineering.

S. Babvey, J. A. Fernández-Zepeda, A. G. Bourgeois, and S. W. McLaughlin. An efficient R-Mesh implementation of LDPC codes message-passing decoder. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS), Workshop Advances in Parallel and Distributed Computing Models (APDCM), Abstracts and CD-ROM*. IEEE Computer Society, 2005.

S. Bade and B.L. Hutchings. FPGA based stochastic neural network implementation. In *[Buell and Pocek, 1994]*, pages 189–198, 1994.

Z. K. Baker and V. K. Prasanna. Time and area efficient pattern matching on FPGAs. In *[Tessier and Schmit, 2004]*, pages 223–232, 2004.

F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Publishers, 1997.

E. Balas. A class of location, distribution and scheduling problems: Modeling and solution methods. In P. Gray and L. Yuanzhang, editors, *Proceedings of the Chinese-U.S. Symposium on System Analysis*. Wiley & Sons, 1983.

A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. Minimizing total tardiness on a single machine using ant colony optimization. *Central European Journal for Operations Research and Economics*, 8(2):125–141, 2000.

V. Baumgarten, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt. PACT XPP - a self-reconfigurable data processing architecture. *The Journal of Supercomputing*, 26(2):167–184, 2003.

J. Becker and M. Glesner. A parallel dynamically reconfigurable architecture designed for flexible application-tailored hardware/software systems in future mobile communication. *The Journal of Supercomputing*, 19(1):105–127, 2001.

J. Becker, T. Pionteck, and M. Glesner. DReAM: A dynamically reconfigurable architecture for future mobile communications applications. In *[Hartenstein and Grünbacher, 2000]*, pages 312–321, 2000.

J. Becker, M. Platzner, and S. Vernalde, editors. *Field-Programmable Logic and Applications. Proceedings of the 14th International Conference, FPL 2004, Leuven, Belgium*, volume 3203 of *LNCS*. Springer-Verlag, 2004.

R. Beckers, J. L. Deneubourg, and S. Goss. Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology*, 159:397–415, 1992.

R. Beckers, J.-L. Deneubourg, and S. Goss. Modulation of trail laying in the ant Lasius niger (hymenoptera: Formicidae) and its role in the collective selection of a food source. *Journal of Insect Behavior*, 6(6):751–759, 1993.

M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. Reconfigurable implementation of elliptic curve crypto algorithms. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2002.

S. J. Bellis and W. P. Marnane. A CORDIC arctangent FPGA implementation for a high-speed 3D-camera system. In *[Hartenstein and Grünbacher, 2000]*, volume 1896, pages 485–494, 2000.

P. Bellows, J. Flidr, L. Gharai, C. Perkins, P. Chodowiec, and K. Gaj. IPsec-protected transport of HDTV over IP. In *[Cheung et al., 2003a]*, pages 869–879, 2003.

G. Beni. The concept of cellular robotic systems. In *Proceedings of the 1988 IEEE International Symposium on Intelligent Control*, pages 57–62. IEEE Computer Society Press, 1988.

G. Beni and S. Hackwood. Stationary waves in cyclic swarms. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 234–242. IEEE Computer Society Press, 1992.

G. Beni and J. Wang. Swarm intelligence. In *Proceedings of the Sevent Annual Meeting of the Robotics Society of Japan*, pages 425–428. RSJ Press, 1989.

G. Beni and J. Wang. Theoretical problems for the realization of distributed robotic systems. In *Proceedings of the 1991 IEEE International Conference on Robotic and Automation*, pages 1914–1920. IEEE Computer Society Press, 1991.

A. Benkrid, K. Benkrid, and D. Crookes. Design and implementation of a novel FIR filter architecture with boundary handling on Xilinx Virtex FPGAs. In *[Cheung et al., 2003a]*, pages 553–564, 2003.

J.-L. Beuchat. FPGA implementations of the RC6 block cipher. In *[Cheung et al., 2003a]*, pages 101–110, 2003.

L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO metaheuristic. In *[Dorigo et al., 2002]*, pages 176–187, 2002.

A. Blaickner, O. Nagy, and H. Grünbacher. Fast carrier and phase synchronization units for digital receivers based on re-configurable logic. In *[Hartenstein and Grünbacher, 2000]*, pages 322–331, 2000.

I. M. Bland and G. M. Megson. Implementing a generic systolic array for genetic algorithms. In *Proceedings First On-Line Workshop on Soft Computing*, pages 268–273, 1996.

I. M. Bland and G. M. Megson. Synthesis of a systolic array genetic algorithm. In *IPPS: 11th International Parallel Processing Symposium*, pages 316–320, 1998a.

I. M. Bland and G. M. Megson. The systolic array genetic algorithm, an example of systolic arrays as a reconfigurable design methodology. In *[Pocek and Arnold, 1998]*, pages 260–261, 1998b.

I. M. Bland and G. M. Megson. A systolic array library for hardware genetic algorithms. In *Proceedings 2nd International Conference Massively Parallel Computing Systems*, 1998c.

C. Blum. ACO applied to group shop scheduling: A case study on intensification and diversification. In *[Dorigo et al., 2002]*, pages 14–27, 2002.

C. Blum. An ant colony optimization algorithm to tackle shop scheduling problems. Technical Report TR/IRIDIA/2003-1, Université Libre de Bruxelles, 2003.

C. Blum and M J. Blesa. Metaheuristics for the edge-weighted k-cardinality tree problem. Technical Report LSI-03-1-R, Departament de Llenguatges i Sistemes Informátics, Universitat Politécnica de Catalunya, Barcelona, 2003.

C. Blum, A. Roli, and M. Dorigo. HC-ACO: The hyper-cube framework for ant colony optimization. In *Proceedings of the 4th Metaheuristics International Conference (MIC 2001)*, pages 399–403, 2001.

C. Bobda and T. Lehmann. Efficient building of word recognizer in FPGAs for term-document matrices construction. In *[Hartenstein and Grünbacher, 2000]*, pages 759–768, 2000.

M. Bolondi and M. Bondaza. Parallelizzazione di un algoritmo per la risoluzione del problema del comesso viaggiatore. Master's thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 1993.

M. Bolotski, A. DeHon, and T. F. Knight Jr. Unifying FPGAs and SIMD arrays. In *[Rose and Sangiovanni-Vincentelli, 1994]*, pages 1–10, 1994.

J. A. Boluda and F. Pardo. Synthesizing on a reconfigurable chip an autonomous robot image processing system. In *[Cheung et al., 2003a]*, pages 458–467, 2003.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press, 1999.

E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in telecommunication networks with "smart" ant-like agents. In *Proceedings of IATA'98, Second International Workshop on Intelligent Agents for Telecommunication Applications*, volume 1437 of *LNCS*, pages 60–72. Springer-Verlag, 1998.

K. Bondalapati. *Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures*. PhD thesis, University of Southern California, Los Angeles, 2001.

K. Bondalapati and V. K. Prasanna. Reconfigurable meshes: Theory and practice. In R. W. Hartenstein and V. K. Prasanna, editors, *Proceedings of the Reconfigurable Architectures Workshop*, Geneva, Switzerland, 1997. ITpress Verlag.

K. Bondalapati and V. K. Prasanna. Reconfigurable computing systems. *Proceedings of the IEEE*, 90(7):1201–1217, 2002.

J. L. Bordim, T. Watanabe, K. Nakano, and T. Hayashi. A tool for algorithm visualization on the reconfigurable mesh. In *1999 International Symposium on*

Parallel Architectures, Algorithms and Networks (ISPAN '99), pages 406–413. IEEE Computer Society, 1999.

M. Boyd and T. Larrabee. ELVIS – a scalable, loadable custom programmable logic device for solving boolean satisfiability problems. In *[Hutchings, 2000]*, pages 13–21, 2000.

J. Branke and M. Guntsch. New ideas for applying ant colony optimization to the probabilistic TSP. In S. Cagnoni et al., editors, *Applications of Evolutionary Computing*, volume 2611 of *LNCS*, pages 165–175. Springer-Verlag, 2003.

D. R. Brasen and G. Saucier. Using cone structures for partitioning into FPGA packages. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 17 (7):592–600, 1998.

F. Braun, J. W. Lockwood, and M. Waldvogel. Reconfigurable router modules using network protocol wrappers. In *[Brebner and Woods, 2001]*, pages 254–263, 2001.

G. J. Brebner and R. Woods, editors. *Field-Programmable Logic and Applications. Proceedings of the 11th International Conference, FPL 2001, Belfast, Northern Ireland, UK*, volume 2147 of *LNCS*. Springer-Verlag, 2001.

M. Brucke, A. Schulz, and W. Nebel. Auditory signal processing in hardware: A linear gammatone filterbank design for a model of the auditory system. In *[Lysaght et al., 1999]*, pages 11–20. Springer-Verlag, 1999.

B. Brugger, K. Doerner, R. F. Hartl, and M. Reimann. AntPacking – an ant colony optimization approach for the one-dimensional bin packing problem. In J. Gottlieb and G. R. Raidl, editors, *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2004*, volume 3004 of *LNCS*, pages 41–50. Springer-Verlag, 2004.

D. A. Buell and K. L. Pocek, editors. *Proceedings of the 1993 IEEE Workshop on FPGAs for Custom Computing Machines, Napa Valley, California, USA*. IEEE Computer Society, 1993.

D. A. Buell and K. L. Pocek, editors. *Proceedings of the 1994 IEEE Workshop on FPGAs for Custom Computing Machines, Napa Valley, California, USA*. IEEE Computer Society, 1994.

T. N. Bui and J. R. Rizzo. Finding maximum cliques with distributed ants. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta,

D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part I*, volume 3102 of *LNCS*, pages 24–35. Springer-Verlag, 2004.

T. N. Bui and G. Sundarraj. Ant system for the k-cardinality tree problem. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part I*, volume 3102 of *LNCS*, pages 36–47. Springer-Verlag, 2004.

B. Bullnheimer, R. F: Hartl, and C. Strauss. A new rank based version of the ant system - a computational study. Technical report, Institute of Management Science, University of Vienna, 1997.

B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 285–296. Kluwer Academic Publishers, 1999a.

B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999b.

B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the ant system: A computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999c.

B. Bullnheimer, G. Kotsis, and C. Strauss. Parallelization strategies for the ant system. In R. De Leone, A. Murli, P. Pardalos, and G. Toraldo, editors, *High Performance Algorithms and Software in Nonlinear Optimization*, volume 24 of *Applied Optimization*, pages 87–100. Kluwer, 1998.

R.E. Burkard, S.E. Karisch, and F. Rendl. QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997.

M. Butts, A. DeHon, and S. C. Goldstein. Molecular electronics: devices, systems and tools for gigagate, gigabit chips. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international Conference on Computer-Aided Design*, pages 433–440. ACM Press, 2002.

Cadence Design Systems Inc. Incisive Palladium familiy with Incisive XE software. Datasheet, 2005.

M. J. Canet, F. Vicedo, V. Almenar J. Valls, and E. R. de Lima. Hardware design of a FPGA-based synchronizer for Hiperlan/2. In *[Becker et al., 2004]*, pages 494–504, 2004.

F. Cardells-Tormo, J. Valls-Coquillat, V. Almenar-Terre, and V. Torres-Carot. Efficient FPGA-based QPSK demodulation loops: Application to the DVB standard. In *[Glesner et al., 2002]*, pages 102–111, 2002.

A. Carreira, T.W. Fox, and L.E. Turner. A method for implementing bit-serial finite impulse response digital filters in FPGAs using JBits. In *[Glesner et al., 2002]*, pages 222–231, 2002.

J. Casillas, O. Cordón, and F. Herrera. Learning cooperative fuzzy linguistic rules using ant colony algorithms. Technical Report DECSAI-00-01-19, Department of Computer Science and Artificial Intelligence, University of Granada, Granada, 2000.

E. Cela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, 1998.

J. Cerdá, R. Gadea, V. Herrero, and A. Sebastiá. On the implementation of a Margolus neighborhood cellular automata on FPGA. In *[Cheung et al., 2003a]*, pages 776–785, 2003.

V. Cerný. A thermodynamical approach to the traveling salesman problem. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

Chameleon Systems Inc. CS2000 advance product specification, 2000. San Jose, CA.

P. K. Chan and J. Rose, editors. *FPGA 1995: Proceedings of the 1995 ACM/SIGDA 3rd International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 1995.

S. Charlwood and P. James-Roxby. Evaluation of the XC6200-series architecture for cryptographic applications. In *[Hartenstein and Keevallik, 1998]*, pages 218–227, 1998.

F. Charot, E. Yahya, and C. Wagner. Efficient modular-pipelined AES implementation in counter mode on ALTERA FPGA. In *[Cheung et al., 2003a]*, pages 282–291, 2003.

O. Y. H. Cheung and P. H. W. Leong. Implementation of an FPGA based accelerator for virtual private networks. In *[Leong and Luk, 2002]*, pages 34–43, 2002.

P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors. *Field-Programmable Logic and Applications. Proceedings of the 13th International Conference, FPL 2003, Lisbon, Portugal*, volume 2778 of *LNCS*. Springer-Verlag, 2003a.

R. C. C. Cheung, W. Luk, and P. Y. K. Cheung. Reconfigurable elliptic curve cryptosystems on a chip. In *2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005), 7-11 March 2005, Munich, Germany*, pages 24–29. IEEE Computer Society, 2005.

R. C.C. Cheung, K.P. Pun, S. C.L. Yuen, K.H. Tsoi, and P. H.W. Leong. An FPGA-based re-configurable 24-bit 96kHz sigma-delta audio DAC. In *[Asada and Fujita, 2003]*, pages 110–117, 2003b.

D. M. Chitty and M. L. Hernandez. A hybrid ant colony optimisation technique for dynamic vehicle routing. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part I*, volume 3102 of *LNCS*, pages 48–59. Springer-Verlag, 2004.

P. Chodowiec, P. Khuon, and K. Gaj. Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining. In *[Schlag and Tessier, 2001]*, pages 94–102, 2001.

S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang. Energy-efficient signal processing using FPGAs. In *[Trimberger and Tessier, 2003]*, pages 225–234, 2003.

M. Chugh, D. Bhatia, and P. T. Balsara. Design and implementation of configurable W-CDMA rake receiver architectures on FPGA. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2005.

C. R. Clark and D. E. Schimmel. A pattern-matching co-processor for network intrusion detection. In *[Asada and Fujita, 2003]*, pages 68–74, 2003.

C. A. Coello Coello, R. Laura Zavala, B. Mendoza García, and A. Hernández Aguirre. Ant colony system for the design of combinational logic circuits. In J. F. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware, Proceedings of the Third International*

Conference, ICES 2000, volume 1801 of *LNCS*, pages 21–30. Springer-Verlag, 2000.

A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34:39–53, 1994.

K. Compton, J. Cooley, S. Knol, and S. Hauck. Configuration relocation and defragmentation for FPGAs. Technical report, Nortwestern University, 2000.

K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, 2002.

J. Cong and S. Kaptanoglu, editors. *FPGA 1998: Proceedings of the 1998 ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 1998.

O. Cordón, I. F. de Viana, F. Herrera, and L. Moreno. A new ACO model integrating evolutionary computation concepts: The best-worst ant system. In *[Dorigo et al., 2000]*, pages 22–29, 2000.

R. Cordone and F. Maffioli. Coloured ant system and local search to design local telecommunication networks. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, volume 2037 of *LNCS*, pages 60–69. Springer-Verlag, 2001.

S. Coric, M. Leeser, E. Miller, and M. Trepanier. Parallel-beam backprojection: an FPGA implementation optimized for medical imaging. In *[Schlag and Trimberger, 2002]*, pages 217–226, 2002.

P. Corsonello, S. Perri, M. A. Iachino, and G. Cocorullo. Variable precision multipliers for FPGA-based reconfigurable computing systems. In *[Cheung et al., 2003a]*, pages 661–669, 2003.

D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

T. V. Court, Y. Gu, and M. Herbordt. FPGA acceleration of rigid molecule interactions. In *[Becker et al., 2004]*, pages 862–867, 2004.

T. Courtney, R. H. Turner, and R. Woods. Multiplexer based reconfiguration for Virtex multipliers. In *[Hartenstein and Grünbacher, 2000]*, pages 749–758, 2000.

F. Crowe, A. Daly, T. Kerins, and W. Marnane. Single-chip FPGA implementation of a cryptographic co-processor. In *[Diessel and Williams, 2004]*, pages 279–285, 2004.

A. Daly and W. Marnane. Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. In *[Schlag and Trimberger, 2002]*, pages 40–49, 2002.

A. Daly, W. Marnane, T. Kerins, and E. Popovici. Fast modular division for application in ECC on reconfigurable logic. In *[Cheung et al., 2003a]*, pages 786–795, 2003.

A. Dandalis, A. Mei, and V. K. Prasanna. Domain specific mapping for solving graph problems on reconfigurable devices. In J. D. P. Rolim et al., editors, *Parallel and Distributed Processing, 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, volume 1586 of *LNCS*, pages 652–660. Springer-Verlag, 1999.

A. Dandalis and V. K. Prasanna. Configuration compression for FPGA-based embedded systems. In *[Schlag and Tessier, 2001]*, pages 173–182, 2001.

A. Dandalis, V. K. Prasanna, and B. Thiruvengadam. Run-time performance optimization of an FPGA-based deduction engine for SAT solvers. In *[Brebner and Woods, 2001]*, pages 315–325, 2001.

S. Dasasathyan, R. Radhakrishnan, and R. Vemuri. Framework for synthesis of virtual pipelines. In *Proceedings of the ASPDAC 2002 / VLSI Design 2002*, pages 326–331. IEEE Computer Society, 2002.

A. Datta. Constant-time algorithm for medial axis transform on the reconfigurable mesh. In *Proceedings of the 13th International Parallel Processing Symposium, 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP '99)*, pages 431–435. IEEE Computer Society, 1999.

L. M. de Campos, J. A. Gámez, and J. M. Puerta. Learning bayesian networks by ant colony optimisation: Searching in the space of orderings. *Mathware and Soft Computing*, 9(2-3):251–268, 2002.

J. de Sousa, J. M. da Silva, and J. P. Abramovici. A configurable hardware/software approach to SAT solving. In *[Pocek and Arnold, 2001]*, pages 239–248, 2001.

L. de Souza, P. Ryan, J. Crawford, K. Wong, G. Zyner, and T. McDermott. Prototyping for the concurrent development of an IEEE 802.11 wireless LAN chipset. In *[Cheung et al., 2003a]*, pages 51–60, 2003.

A. DeHon and M. J. Wilson. Nanowire-based sublithographic programmable logic arrays. In *[Tessier and Schmit, 2004]*, pages 123–132, 2004.

P. Delisle, M. Gravel, M. Krajecki, C. Gagné, and W. L. Price. A shared memory parallel implementation of ant colony optimization. In R. F. Hartl et al., editors, *Abstract Proceedings of the Sixth Metaheuristics International Conference (MIC 2005)*, 2005.

M. deLorimier and A. DeHon. Floating-point sparse matrix-vector multiply for FPGAs. In *[Schmit and Wilton, 2005]*, pages 75–85, 2005.

M. den Besten, T. Stützle, and M. Dorigo. An ant colony optimization application to the single machine total weighted tardiness problem. In *[Dorigo et al., 2000]*, pages 39–42, 2000a.

M. den Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer et al., editors, *Parallel Problem Solving from Nature: 6th international conference*, volume 1917 of *LNCS*, pages 611–620, Berlin, 2000b. Springer-Verlag.

J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior*, 3:159–168, 1990.

J.-L. Deneubourg, S. Goss, N. R. Franks, and J. M. Pasteels. The blind leading the blind: Modeling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, 2:719–725, 1989.

J. L. Deneubourg, J. M. Pasteels, and J. C. Verhaege. Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology*, 105:259–271, 1983.

D. Denning, J. Irvine, and M. Devlin. A key agile 17.4 Gbit/sec Camellia implementation. In *[Becker et al., 2004]*, pages 546–554, 2004.

K. Denolf, A. Chirila-Rus, R. Turney, P. Schumacher, and K. Vissers. Memory efficient design of an MPEG-4 video encoder for FPGAs. In *[Rissa et al., 2005]*, 2005.

D. Deshpande, A. K. Somani, and A. Tyagi. Configuration caching vs. data caching for striped FPGAs. In *[Kaptanoglu and Trimberger, 1999]*, pages 206–214, 1999.

G. Di Caro and M. Dorigo. AntNet: A mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, IRIDIA, Université Libre de Bruxelles, 1997.

G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998a.

G. Di Caro and M. Dorigo. Extending AntNet for the best-effort quality-of-service routing. unpublished presentation. In *[Dorigo, 1998]*, 1998b.

G. Di Caro and M. Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In Y. Pan, S. G. Akl, and K. Li, editors, *Proceedings of PDCS'98 - 10th International Conference on Parallel and Distributed Computing and Systems*, pages 541–546, Las Vegas, Nevada, 1998c. ACTA Press.

C. Dick. Computing the discrete fourier transform on FPGA based systolic arrays. In *[Rose and Ebeling, 1996]*, pages 129–135, 1996.

C. Dick and F. Harris. FIR filtering with FPGAs using quadrature sigma-delta modulation encoding. In *[Hartenstein and Glesner, 1996]*, pages 361–365, 1996.

C. Dick and F. Harris. FPGA QAM demodulator design. In *[Glesner et al., 2002]*, pages 112–121, 2002.

O. Diessel, H. ElGindy, M. Middendorf, M. Guntsch, B. Scheuermann, H. Schmeck, and K. So. Population based ant colony optimization on FPGA. In *[Leong and Luk, 2002]*, pages 125–132, 2002.

O. Diessel, H. ElGindy, M. Middendorf, B. Schmidt [Scheuermann], and H. Schmeck. Dynamic scheduling on partially reconfigurable FPGAs. *IEE Proceedings - Computers and Digital Techniques. Special issue on reconfigurable systems*, 147(3):181–188, 2000.

O. Diessel and J. A. Williams, editors. *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology (FPT)*. University of Queensland, Brisbane, Australia, IEEE Computer Society, 2004.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, pages 269–271, 1959.

R. Dimond, O. Mencer, and W. Luk. CUSTARD - a customisable threaded FPGA soft processor and tools. In *[Rissa et al., 2005]*, 2005.

J. Ditmar, K. Torkelsson, and A. Jantsch. A dynamically reconfigurable FPGA-based content addressable memory for internet protocol characterization. In *[Hartenstein and Grünbacher, 2000]*, pages 19–28, 2000.

T.-T. Do, H. Kropp, C. Reuter, and P. Pirsch. A flexible implementation of high-performance FIR filters on Xilinx FPGAs. In *[Hartenstein and Keevallik, 1998]*, pages 441–445, 1998.

K. Doerner, M. Gronalt, R. F. Hartl, M. Reimann, C. Strauss, and M. Stummer. SavingsAnts for the vehicle routing problem. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, editors, *Proceedings of the EvoWorkshops 2002, Applications of Evolutionary Computing*, LNCS 2279, Kinsale, Ireland, 2002. Springer-Verlag.

K. Doerner, R. F. Hartl, G. Kiechle, M. Lucka, and M. Reimann. Parallel ant systems for the capacitated vehicle routing problem. In J. Gottlieb and G. R. Raidl, editors, *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2004*, volume 3004 of *LNCS*, pages 72–83. Springer-Verlag, 2004.

K. Doerner, R. F. Hartl, V. Maniezzo, and M. Reimann. An ant system metaheuristic for the capacitated arc routing problem. In *Proceedings of the Fifth Metaheuristics International Conference (MIC 2003)*, 2003. URL: http://www.ruca.ua.ac.be/eume/MIC2003/pdf/MIC03_16.pdf.

A. Dollas, E. Sotiriades, and A. Emmanouelides. Architecture and design of GE1, a FCCM for golomb ruler derivation. In *[Pocek and Arnold, 1998]*, pages 48–57, 1998.

M. Dorigo. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, 1992.

M. Dorigo. Parallel ant system: An experimental study. Unpublished manuscript, 1993.

M. Dorigo, editor. *From Ant Colonies to Artificial Ants. Abstract Proceedings of the 1st International Workshop on Ant Algorithms, ANTS 1998.* Université Libre de Bruxelles, Belgium, 1998.

M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors. *Ant Colony Optimization and Swarm Intelligence. Proceedings of the 4th International Workshop ANTS 2004*, volume 3172 of *LNCS*. Université Libre de Bruxelles, Belgium, Springer-Verlag, 2004.

M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

M. Dorigo, G. Di Caro, and M. Sampels, editors. *From Ant Colonies to Artificial Ants. Proceedings of the 3rd International Workshop on Ant Algorithms, ANTS 2002*, volume 2463 of *LNCS*. Université Libre de Bruxelles, Belgium, Springer-Verlag, 2002.

M. Dorigo and L. M. Gambardella. Ant-Q: A reinforcement learning approach to combinatorial optimization. Technical Report IRIDIA/95-01, Universite Libre de Bruxelles, Belguim, 1995.

M. Dorigo and L. M. Gambardella. A study of some properties of Ant-Q. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-S. Schwefel, editors, *Proceedings of PPSN IV – Fourth International Conference on Parallel Problem Solving from Nature*, pages 656–665, 1996.

M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):73–81, 1997a.

M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997b.

M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimization process. Technical Report 91-016 revised, Dipartimento di Elettronica, Politecnico di Milano, 1991a.

M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, Italy, 1991b.

M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

M. Dorigo, M. Middendorf, and T. Stützle, editors. *From Ant Colonies to Artificial Ants. Abstract Proceedings of the 2nd International Workshop on Ant Algorithms, ANTS 2000*. Université Libre de Bruxelles, Belgium, 2000.

M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 251–285. Kluwer Academic Publishers, 2002.

M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

Y. Dou, S. Vassiliadis, G. K. Kuzmanov, and G. N. Gaydadjiev. 64-bit floating-point FPGA matrix multiplication. In *[Schmit and Wilton, 2005]*, pages 86–95, 2005.

J. Du and J. Y.-T. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15:483–485, 1990.

P. A. Dunn and P. I. Corke. Real-time stereopsis using FPGAs. In *[Luk et al., 1997a]*, pages 400–409, 1997.

S. Dydel and P. Bala. Large scale protein sequence alignment using FPGA reprogrammable logic devices. In *[Becker et al., 2004]*, pages 23–32, 2004.

C. Ebeling and J. Cong, editors. *FPGA 1997: Proceedings of the 1997 ACM/SIGDA 5th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 1997.

C. Ebeling, D. Conquist, and P. Franklin. RaPiD – reconfigurable pipelined datapath. In *[Hartenstein and Glesner, 1996]*, pages 126–135, 1996.

C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu. Implementing an OFDM receiver on the RaPiD reconfigurable architecture. In *[Cheung et al., 2003a]*, pages 21–30, 2003.

A. Ejnioui and N. Ranganathan. Multi-terminal net routing for partial crossbar-based multi-FPGA systems. In *[Kaptanoglu and Trimberger, 1999]*, pages 176–184, 1999.

A. J. Elbirt and C. Paar. An FPGA implementation and performance evaluation of the serpent block cipher. In *[Trimberger and Hauck, 2000]*, pages 33–40, 2000.

J. G. Eldredge and B. L. Hutchings. Density enhancement of a neural network using FPGAs and run-time reconfiguration. In *[Buell and Pocek, 1994]*, pages 180–188, 1994.

H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt [Scheuermann]. Task rearangement on partially reconfigurable FPGAs with restricted buffer. In *[Hartenstein and Grünbacher, 2000]*, pages 379–388, 2000.

H. ElGindy, M. Middendorf, B. Schmidt [Scheuermann], and H. Schmeck. An evolutionary approach to dynamic task scheduling on FPGAs with restricted buffer. *Journal of Parallel and Distributed Computing*, 62(9):1407–1420, 2002.

H. ElGindy, A. K. Somani, H. Schröder, H. Schmeck, and A. Spray. RMB – a reconfigurable multiple bus network. In *Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture*, pages 108–117, 1996.

Elixent Corp. DFA 1000 accelerator datasheet, 2003.

I. Ellabib, O. A. Basir, and P. Calamai. An experimental study of a simple ant colony system for the vehicle routing problem with time windows. In *[Dorigo et al., 2002]*, pages 53–64, 2002.

I. Ellabib, O. A. Basir, and P. Calamai. A new ant colony system updating strategy for vehicle routing problem with time windows. In *Proceedings of the Fifth Metaheuristics International Conference (MIC 2003)*, 2003. URL: http://webhost.ua.ac.be/eume/MIC2003/pdf/MIC03_18.pdf.

J. Esquiagola, G. Ozari, M. Teruya, M. Strum, and W. Chau. A dynamically reconfigurable bluetooth base band unit. In *[Rissa et al., 2005]*, 2005.

M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *[Schlag and Tessier, 2001]*, pages 103–110, 2001.

G. Estrin, B. Bussel, R. Turn, and J. Bibb. Parallel processing in a restructurable computer system. *IEEE Transactions on Electronic Computers*, pages 747–755, 1963.

C. J. Eyckelhof and M. Snoek. Ant systems for a dynamic TSP. In *[Dorigo et al., 2002]*, pages 88–99, 2002.

B. S. Fagin and P. Chintrakulchai. A reprogrammable processor for fractal image compression. In *[Hartenstein and Servit, 1994]*, pages 129–131, 1994.

S. A. Fahmya, P. Y. K. Cheung, and W. Luk. Novel FPGA-based implementation of median and weighted median filters for image processing. In *[Rissa et al., 2005]*, 2005.

J. Fender and J. Rose. A high-speed ray tracing engine built on a field-programmable system. In *[Asada and Fujita, 2003]*, pages 188–195, 2003.

S. Fenet and C. Solnon. Searching for maximum cliques with ant colony optimization. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardala, D. W. Corne, J. Gottlieb, A. Guillot ant E. Hart, C. G. Johnson, and E. Marchiori, editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2003*, volume 2611 of *LNCS*, pages 236–245. Springer-Verlag, 2003.

S. Fidanova. Evolutionary algorithm for multiple knapsack problem. In *Proceedings of PPSN-VII, Seventh International Conference on Parallel Problem Solving from Nature*, LNCS. Springer-Verlag, Berlin, Germany, 2002.

A. Fidjeland, W. Luk, and S. Muggleton. Scalable acceleration of inductive logic programs. In *[Leong and Luk, 2002]*, pages 252–259, 2002.

L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, 1966.

W. K. Foong, H. R. Maier, and A. R. Simpson. Ant colony optimization for power plant maintenance scheduling optimization. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO 2005)*, pages 249–256. ACM Press, 2005.

W. Fornaciari and V. Piuri. Virtual FPGAs: Some steps behind the physical barriers. In *IPPS/SPDP Workshops*, pages 7–12, 1998.

P. Forsyth and A. Wren. An ant system for bus driver scheduling. Technical Report Report 97.25, University of Leeds - School of Computer Studies, 1997.

K. Fujita, A. Saito, T. Matsui, and H. Matsuo. An adaptive ant-based routing algorithm used routing history in dynamic networks. In *4th Asia-Pacific Conference on Simulated Evolution And Learning, SEAL 2002*, Singapore, 2002.

C. Gagné, W. L. Price, and M. Gravel. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *Journal of the Operational Research Society*, 53:895–906, 2002.

L. M. Gambardella and M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In A Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pages 252–260. Morgan Kaufmann, 1995.

L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *Proceedings of the IEEE Conference on Evolutionary Computing*, pages 622–627. IEEE Press, 1996.

L. M. Gambardella and M. Dorigo. HAS-SOP: A hybrid ant system for the sequential ordering problem. Technical Report IDSIA-11-97, IDSIA, Lugano, Switzerland, 1997.

L. M. Gambardella and M. Dorigo. Ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal of Computing*, 12(3):237–255, 2000.

L. M. Gambardella, E. D. Taillard, and G. Agazzi. *New Ideas in Optimization*, chapter MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, pages 63–76. McGraw Hill, London, UK, 1999a.

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

M. Gangadhar and D. Bhatia. FPGA based EBCOT architecture for JPEG 2000. In *[Asada and Fujita, 2003]*, pages 228–233, 2003.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1983.

A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan. A dual-Vdd low power FPGA architecture. In *[Becker et al., 2004]*, pages 145–157, 2004.

V. George, H. Zhang, and J. Rabaey. The design of a low energy FPGA. In *ISLPED '99: Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, pages 188–193. ACM Press, 1999.

M. Glesner, P. Zipf, and M. Renovell, editors. *Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream. Proceedings of the 12th International Conference, FPL 2002, Montpellier, France*, volume 2438 of *LNCS*. Springer-Verlag, 2002.

F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

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

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor. PipeRench: A reconfigurable architecture and compiler. *Computer*, 33(4):70–77, 2000.

I. Gonzalez, S. Lopez-Buedo, F. J. Gomez, and J. Martinez. Using partial reconfiguration in cryptographic applications: An implementation of the IDEA algorithm. In *[Cheung et al., 2003a]*, pages 194–203, 2003.

T. G. W. Gordon and P. J. Bentley. *Soft Computing in Industrial Electronics*, chapter On Evolvable Hardware. Physica-Verlag, 2002.

M. Gorgon, S. Cichon, and M. Pac. Real-time Handel-C based implementation of DV decoder. In *[Rissa et al., 2005]*, 2005.

S. Goss, S. Aron, J.-L. Deneubourg, and J.-M. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76, 1989.

P. Graham and B. Nelson. A hardware genetic algorithm for the traveling salesman problem on Splash2. In *[Moore and Luk, 1995]*, pages 352–361, 1995.

P. Graham and B. Nelson. Genetic algorithms in software and in hardware – a performance analysis of workstation and custom computing machine implementations. In *[Arnold and Pocek, 1995]*, pages 216–225, 1996.

Mentor Graphics. VStation PRO, high-performance system verification. Datasheet, 2003.

P.-P. Grassé. La reconstruction du nid et les coordinations interindividuelle chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interpretation des termites constructeurs. *Insectes Sociaux*, 6:41–80, 1959.

P.-P. Grassé. Termitologia, tome II. *Fondation des Sociétés. Construction*, 1984.

M. Guntsch. *Ant Algorithms in Stochastic and Multi-Criteria Environments*. PhD thesis, Institute AIFB, University of Karlsruhe, 2004.

M. Guntsch, J. Branke, M. Middendorf, and H. Schmeck. ACO strategies for dynamic TSP. In *[Dorigo et al., 2000]*, pages 59–62, 2000.

M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In E.J.W. Boers et al., editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, number 2037 in LNCS, pages 213–222. Springer-Verlag, 2000.

M. Guntsch and M. Middendorf. Applying population based ACO to dynamic optimization problems. In *[Dorigo et al., 2002]*, pages 111–122, 2002a.

M. Guntsch and M. Middendorf. A population based approach for ACO. In S. Cagnoni and others, editor, *Applications of Evolutionary Computing - EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, number 2279 in LNCS, pages 72–81. Springer-Verlag, 2002b.

M. Guntsch, M. Middendorf, and H. Schmeck. An ant colony optimization approach to dynamic TSP. In L. Spector et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 860–867. Morgan Kaufmann Publishers, 2001.

Z. Guo, W. Najjar, F. Vahid, and K. Vissers. A quantitative analysis of the speedup factors of FPGAs over processors. In *[Tessier and Schmit, 2004]*, pages 162–170, 2004.

R. K. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3):29–41, 1993.

J. Gutjahr. A generalized ant system and its convergence. Technical Report 98-10, Dept. of Statics, Operations Research and Computer Science, University of Vienna, 1998.

W. Gutjahr. A graph-based ant system and its convergence. *Future Generations Computer Systems*, 16:873–888, 2000.

W. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82(3):145–153, 2002.

W. Gutjahr. A converging ACO algorithm for stochastic combinatorial optimization. In A. Albrecht and K. Steinhöfel, editors, *Stochastic Algorithms: Foundations and Applications: Second International Symposium, SAGA*, volume 2827 of *LNCS*, pages 10–25. Springer-Verlag, 2003a.

W. Gutjahr. A generalized convergence result for the graph-based ant system metaheuristic. *Probability in the Engineering and Informational Sciences*, 17: 545–569, 2003b.

S. Hackwood and G. Beni. Self-organizing sensors by deterministic annealing. In *Proceedings of the 1991 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'91*, pages 1177–1183. IEEE Computer Society Press, 1991.

S. Hackwood and G. Beni. Self-organization of sensors for swarm intelligence. In *Proceedings of the 1992 IEEE International conference on Robotics and Automation*, pages 819–829. IEEE Computer Society Press, 1992.

J. R. Haddy and D. J. Skellern. An asynchronous transfer mode (ATM) stream demultiplexer and switch. In *[Hartenstein and Glesner, 1996]*, pages 260–269, 1996.

R. Hadji, M. Rahoual, E. Talbi, and V. Bachelet. Ant colonies for the set covering problem. In *[Dorigo et al., 2000]*, pages 63–66, 2000.

H. Haken. *Synergetics*. Springer-Verlag, 1983.

Y. Hamadi and D. Merceron. Reconfigurable architectures: A new vision for optimizing problem. In *Proceedings 3rd International Conference Principles Practice Constraint Programming (CP'97)*, LNCS 1330, pages 209–221. Springer-Verlag, 1997.

R. W. Hartenstein and M. Glesner, editors. *Field-Programmable Logic: Smart Applications, New Paradigms and Compilers. Proceedings of the 6th International Workshop on Field Programmable Logic and Applications, FPL 1996, Darmstadt, Germany*, volume 1142 of *LNCS*. Springer-Verlag, 1996.

R. W. Hartenstein and H. Grünbacher, editors. *Field-Programmable Logic and Applications. The Roadmap to Reconfigurable Computing. Proceedings of the 10th International Conference, FPL 2000, Villach, Austria*, volume 1896 of *LNCS*. Springer-Verlag, 2000.

R. W. Hartenstein and A. Keevallik, editors. *Field-Programmable Logic and Applications. From FPGAs to Computing Paradigm. Proceedings of the 8th International Workshop, FPL1998, Tallinn, Estonia*, volume 1482 of *LNCS*. Springer-Verlag, 1998.

R. W. Hartenstein and M. Z. Servit, editors. *Field-Programmable Logic: Architectures, Synthesis and Applications. Proceedings of the 4th International Workshop on Field Programmable Logic and Applications, FPL 1994, Prague, Czech Republic*, volume 849 of *LNCS*. Springer-Verlag, 1994.

S. Hauck. *Multi-FPGA Systems*. PhD thesis, University of Washington, Dept. of Computer Science and Engineering, 1995.

S. Hauck. Configuration prefetch for single context reconfigurable systems. In *[Cong and Kaptanoglu, 1998]*, pages 65–74, 1998.

S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimaera reconfigurable functional unit. In *[Pocek and Arnold, 1997]*, pages 87–96, 1997.

S. Hauck, Z. Li, and E. J. Schwabe. Configuration compression for the Xilinx XC6200 FPGA. In *[Pocek and Arnold, 1998]*, pages 138–146, 1998.

S. Hauck and W. D. Wilson. Runlength compression techniques for FPGA configurations. Technical report, Department of ECE, Northwestern University, 1998.

J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In *[Pocek and Arnold, 1997]*, pages 12–21, 1997.

S. D. Haynes and P. Y. K. Cheung. A reconfigurable multiplier array for video image processing tasks, suitable for embedding in an FPGA structure. In *[Pocek and Arnold, 1998]*, pages 226–234, 1998.

S. D. Haynes, P. Y. K. Cheung, W. Luk, and J. Stone. SONIC - a plug-in architecture for video processing. In *[Lysaght et al., 1999]*, pages 21–30, 1999.

M. Heusse, D. Snyers, S. Guérin, and P. Kuntz. Adaptive agent-driven routing and load balancing in communication networks. *Advances in Complex Systems*, 1(2):237–254, 1998.

J. I. Hidalgo, F. Fernández de Vega, J. Lanchares, J. M. Sánchez-Pérez, R. Hermida, M. Tomassini, R. Baraglia, R. Perego, and O. Garnica. Multi-FPGA systems synthesis by means of evolutionary computation. In E. Cantú-Paz et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), Part II*, volume 2724 of *LNCS*, pages 2109–2120. Springer Verlag, 2003.

P. E. H. Hofmann et al. *Evolutionäre E/E-Architektur. Vision einer neuartigen Elektronik-Architekur für Fahrzeuge.* DaimlerChrysler, 2002.

J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

B. L. Hutchings, editor. *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2000, Napa Valley, California, USA.* IEEE Computer Society, 2000.

S. Ichikawa, H. Saito, L. Udorn, and K. Konishi. Evaluation of accelerator designs for subgraph isomorphism problem. In *[Hartenstein and Grünbacher, 2000]*, pages 729–738, 2000.

T. Ichikawa, T. Sorimachi, T. Kasuya, and M. Matsui. On the criteria of hardware evalution of block ciphers(1). Technical Report ISEC2001-53, IEICE, 2001.

S. Iredi, D. Merkle, and M. Middendorf. Bi-criterion optimization with multi colony ant algorithms. In E. Zitzler et al., editors, *Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01)*, LNCS 1993, pages 359–372. Springer-Verlag, 2001.

P. James-Roxby and E. Cerro-Prada. A wildcarding mechanism for acceleration of partial configurations. In *[Lysaght et al., 1999]*, pages 444–449, 1999.

A. Jamin and P. Mähönen. FPGA implementation of the wavelet packet transform for high speed communications. In *[Glesner et al., 2002]*, pages 212–221, 2002.

J. Jang, S. Choi, and V. K. Prasanna. Area and time efficient implementation of matrix multiplication on FPGAs. In *[Leong and Luk, 2002]*, pages 93–101, 2002.

J. Jang, H. Park, and V. K. Prasanna. A bit model of reconfigurable mesh. In *Proceedings of the Workshop on Reconfigurable Architectures*, Los Alamitos, CA, 1994. IEEE Computer Society.

M. Janiaut, C. Tanougast, H. Rabah, Y. Berviller, C. Mannino, and S. Weber. Configurable hardware implementation of a conceptual decoder for a real-time MPEG-2 analysis. In *[Rissa et al., 2005]*, 2005.

S. Janson, D. Merkle, M. Middendorf, H. ElGindy, and H. Schmeck. On enforced convergence of ACO and its implementation on the reconfigurable mesh architecture using size reduction tasks. In *Proceedings of the Second International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'02)*, pages 3–9. CSREA Press, 2002.

S. Janson, D. Merkle, M. Middendorf, H. ElGindy, and H. Schmeck. On enforced convergence of ACO and its implementation on the reconfigurable mesh architecture using size reduction tasks. *Journal of Supercomputing*, 26(3):221–238, 2003.

K. U. Järvinen, M. T. Tommiska, and J. O. Skyttä. A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In *[Trimberger and Tessier, 2003]*, pages 207–215, 2003.

T. Jebelean. FPGA implementation of a rational adder. In *[Moore and Luk, 1995]*, pages 251–260, 1995.

D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley & Sons, 1995.

C. C. Jong, Y. Y. H. Lam, and L. S. Ng. FPGA implementation of a digital IQ demodulator using VHDL. In *[Luk et al., 1997a]*, pages 410–417, 1997.

H. Kagotani and H. Schmit. Asynchronous PipeRench: Architecture and performance estimations. In *[Pocek and Arnold, 2003]*, pages 121–132, 2003.

T. Kajitani, T. Hoshino, D. Nishikawa, H.Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iwata, D. Keymeulen, and T. Higuchi. A gate-level EHW chip: Implementing GA operations and reconfigurable hardware on a single LSI. In *Proceedings of International Conference on Evolvable Systems (ICES98)*, pages 1–12, 1998.

K. Kanazawa and T. Maruyama. An FPGA solver for WSAT algorithms. In *[Rissa et al., 2005]*, 2005.

S. Kaptanoglu and S. Trimberger, editors. *FPGA 1999: Proceedings of the 1999 ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 1999.

J. F. Keane, C. Bradley, and C. Ebeling. A compiled accelerator for biological cell signaling simulations. In *[Tessier and Schmit, 2004]*, pages 233–241, 2004.

M. A. S. Khalid. *Routing architecture and layout synthesis for multi-FPGA systems*. PhD thesis, Dept. of ECE, University of Toronto, 1999.

C. H. Kim, S. Kwon, J. J. Kim, and C. P. Hong. A new arithmetic unit in GF(2^m) for reconfigurable hardware implementation. In *[Cheung et al., 2003a]*, pages 670–680, 2003.

H. J. Kim and W. H. Mangione-Smith. Factoring large numbers with programmable hardware. In *[Trimberger and Hauck, 2000]*, pages 41–48, 2000.

S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

T. Kobori and T. Maruyama. A high speed computation system for 3D FCHC lattice gas model with FPGA. In *[Cheung et al., 2003a]*, pages 755–765, 2003.

S. Kumar and C. Paar. Reconfigurable instruction set extension for enabling ECC on an 8-bit processor. In *[Becker et al., 2004]*, pages 586–595, 2004.

M.-K. Kwan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

L. Lagadec, D. Lavenier, E. Fabiani, and B. Pottier. Placing, routing and editing virtual FPGAs. In *[Brebner and Woods, 2001]*, pages 357–366, 2001.

J. Lamoureux and S. J. E. Wilton. On the interaction between power-aware FPGA CAD algorithms. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*, pages 701–708. IEEE Computer Society, 2003.

S. Lange and M. Middendorf. Hyperreconfigurable architectures for fast runtime reconfiguration. In *[Pocek and Arnold, 2004]*, pages 304–305, 2004.

R. Laufer, R. R. Taylor, and H. Schmit. PCI-PipeRench and the SwordAPI: A system for stream-based reconfigurable computing. In *[Pocek and Arnold, 1999]*, pages 200–208, 1999.

B. Laurent, G. Bosco, and Gabriele Saucier. Structural versus algorithmic approaches for efficient adders on Xilinx 5200 FPGA. In *[Luk et al., 1997a]*, pages 462–471, 1997.

E. L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

E. L. Lawler, J. K. Lenstra, A. H. Kan, and D. B. Shmoys. *The Traveling Salesman Problem.* John Wiley & Sons, 1985.

B. Lee and N. Burgess. A dual-path logarithmic number system addition/subtraction scheme for FPGA. In *[Cheung et al., 2003a]*, pages 808–817, 2003a.

B. Lee and N. Burgess. A parallel look-up logarithmic number system addition/subtraction scheme for FPGA. In *[Asada and Fujita, 2003]*, pages 76–83, 2003b.

T. K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu, and N. Dulay. Irregular reconfigurable CAM structures for firewall applications. In *[Cheung et al., 2003a]*, pages 890–899, 2003.

G. Leguizamón and Z. Michalewicz. A new version of ant system for subset problems. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, pages 1459–1464, Washington D.C., USA, 1999.

G. Leguizamón and Z. Michalewicz. Ant systems for subset problems. Unpublished manuscript, 2000.

O. Lehtoranta, E. Salminen, A. Kulmala, M. Hännikäinen, and T.D. Hämäläinen. A parallel MPEG-4 encoder for FPGA based multiprocessor SoC. In *[Rissa et al., 2005]*, 2005.

T. Lei, M. Zhu, and J. Wang. The hardware implementation of a genetic algorithm model with FPGA. In *[Leong and Luk, 2002]*, pages 374–377, 2002.

E. Lemoine and D. Merceron. Run time reconfiguration of FPGA for scanning genomic databases. In *[Athanas and Pocek, 1995]*, pages 90–98, 1995.

J. K. Lenstra, A. H. G. Rinnoy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

J. Leonard and W. H. Mangione-Smith. A case study of partially evaluated hardware circuits: Key-specific DES. In *[Luk et al., 1997a]*, pages 151–160, 1997.

P. Leong and W. Luk, editors. *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology (FPT)*. The Chinese University of Hong Kong, IEEE Computer Society, 2002.

P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, and K. H. Lee. Pilchard – a reconfigurable computing platform with memory slot interface. In *[Pocek and Arnold, 2001]*, pages 170–179, 2001.

P. H. W. Leong and K. H. Leung. A microcoded elliptic curve processor using FPGA technology. *IEEE Transactions on VLSI Systems*, 10(5):550–559, 2002.

L. Lessing, I. Dumitrescu, and T. Stützle. A comparison between ACO algorithms for the set covering problem. In *[Dorigo et al., 2004]*, pages 1–12, 2004.

K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong. FPGA implementation of a microcoded elliptic curve cryptographic processor. In *[Hutchings, 2000]*, pages 68–76, 2000.

J. Levine and F. Ducatelle. Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716, 2004.

Z. Li and S. Hauck. Don't care discovery for FPGA configuration compression. In *[Kaptanoglu and Trimberger, 1999]*, pages 91–98, 1999.

Y.-C. Liang and A. E. Smith. An ant system approach to redundancy allocation. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pages 1478–1484. IEEE Press, 1999.

G. Lienhart, R. Männer, K. H. Noffz, and R. Lay. An FPGA-based video compressor for H.263 compatible bit streams. In *[Schlag and Tessier, 2001]*, pages 207–212, 2001.

S. Lin and B. W. Kernighan. An effectice heuristic algorithm for the traveling salesman problem. *Operations Research*, 21, 1973.

J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *[Schlag and Tessier, 2001]*, pages 87–93, 2001.

J. W. Lockwood, J. S. Turner, and D. E. Taylor. Field programmable port extender (FPX) for distributed routing and queuing. In *[Trimberger and Hauck, 2000]*, pages 137–144, 2000.

M. López-Ibáñez, L. Paquete, and T. Stützle. On the design of ACO for the biobjective quadratic assignment problem. In *[Dorigo et al., 2004]*, pages 214–225, 2004.

L. Louca, T. A. Cook, and W. H. Johnson. Implementation of IEEE single precision floating point addition and multiplication on FPGAs. In *[Arnold and Pocek, 1995]*, pages 107–116, 1996.

H. Lourenço and D. Serra. Adaptive approach heuristics for the generalized assignment problem. Technical Report 304, Universitat Pompeu Fabra, Department of Economics and Management, Barcelona, 1998.

H. Lourenço and D. Serra. Adaptive search heuristics for the generalized assignment problem. *Mathware and Soft Computing*, 9(2-3):209–234, 2002.

H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, 2002.

J. Lu and J. W. Lockwood. IPSec implementation on Xilinx Virtex-II Pro FPGA and its application. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2005.

R. Ludewig, O. Soffke, P. Zipf, M. Glesner, K. P. Pun, K. H. Tsoi, K. H. Lee, and P. Leong. IP generation for an FPGA-based audio DAC sigma-delta converter. In *[Becker et al., 2004]*, pages 526–535, 2004.

W. Luk, P. Y. K. Cheung, and M. Glesner, editors. *Field-Programmable Logic and Applications. Proceedings of the 7th International Workshop, FPL 1997, London, UK*, volume 1304 of *LNCS*. Springer-Verlag, 1997a.

W. Luk, N Shirazi, and P. Y. K. Cheung. Compilation tools for run-time reconfigurable designs. In *[Pocek and Arnold, 1997]*, pages 56–65, 1997b.

W. Luk, N. Shirazi, S. Guo, and P. Y. K. Cheung. Pipeline morphing and virtual pipelines. In *[Luk et al., 1997a]*, pages 111–120, 1997c.

P. Lysaght, J. Irvine, and R. W. Hartenstein, editors. *Field-Programmable Logic and Applications. Proceedings of the 9th International Workshops, FPL 1999, Glasgow, UK*, volume 1673 of *LNCS*. Springer-Verlag, 1999.

P. Lysaght, J. Stockwood, J. Law, and D. Girma. Artificial neural network implementation on a fine-grained FPGA. In *[Hartenstein and Servit, 1994]*, pages 421–431, 1994.

N. MacKay and S. Singh. Debugging techniques for dynamically reconfigurable hardware. In *[Pocek and Arnold, 1999]*, pages 114–122, 1999.

D. MacVicar, J. W. Patterson, and S. Singh. Rendering postscript fonts on FPGAs. In *[Lysaght et al., 1999]*, pages 223–232, 1999.

M. Majer, C. Bobda, A. Ahmadinia, and J. Teich. Packet routing in dynamically changing networks on chip. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2005.

W. K. Mak and D. F. Wong. Board-level multi net routing for FPGA-based logic emulation. *ACM Transactions on Design Automation of Electronic Systems*, 2 (2):151–167, 1997.

T. Makimoto. The rising wave of field programmability. In *[Hartenstein and Grünbacher, 2000]*, pages 1–6, 2000.

V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4): 358–369, 1999.

V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8):927–935, 2000.

V. Maniezzo, A. Carbonaro, M. Golfarelli, and S. Rizzi. An ANTS algorithm for optimizing the materialization of fragmented views in data warehouses: Preliminary results. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing, EvoWorkshops 2001*, LNCS 2037, pages 80–89, 2001a.

V. Maniezzo, A. Carbonaro, M. Golfarelli, and S. Rizzi. ANTS for data warehouse logical design. In J. P. de Sousa, editor, *Proceedings of the 4th Metaheuristics International Conference, MIC 2001*, pages 249–254, Porto, Portugal, 2001b.

V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Data and Knowledge Engineering*, 11(5):769–778, 1999.

V. Maniezzo, A. Colorni, and M. Dorigo. The ant system applied to the quadratic assignment problem. Technical Report IRIDIA/94-28, IRIDIA, IRIDIA, Université Libre de Bruxelles, 1994.

P. Marchal and E. Sanchez. CAFCA (compact accelerator for cellular automata): the metamorphosable machine. In *[Buell and Pocek, 1994]*, pages 66–71, 1994.

A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings. A reconfigurable arithmetic array for multimedia applications. In *[Kaptanoglu and Trimberger, 1999]*, pages 135–143, 1999.

K. Maruo, M. Ichikawa, N. Miyamoto, L. Karnan, T. J. Yamaguchi, K. Kotani, and T. Ohmi. A dynamically-reconfigurable image recognition processor. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2004.

S. Maya, R. Reynoso, C. Torres, and Miguel Arias-Estrada. Compact spiking neural network implementation in FPGA. In *[Hartenstein and Grünbacher, 2000]*, pages 270–276, 2000.

J. S. McCaskill and P. Wagler. From reconfigurability to evolution in construction systems: Spanning the electronic, microfluidic and biomolecular domains. In *[Hartenstein and Grünbacher, 2000]*, pages 286–299, 2000.

R. McCready. Real-time face detection on a configurable hardware system. In *[Hartenstein and Grünbacher, 2000]*, volume 1896, pages 157–162, 2000.

T. McDermott, P. Ryan, M. Shand, D. Skellern, T. Percival, and N. Weste. A wireless LAN demodulator in a Pamette: design and experience. In *[Pocek and Arnold, 1997]*, pages 40–45, 1997.

J. T. McHenry, P. W. Dowd, F. A. Pellegrino, T. M. Carrozzi, and W. B. Cocks. An FPGA-based coprocessor for ATM firewalls. In *[Pocek and Arnold, 1997]*, pages 30–39, 1997.

M. McLoone and J. V. McCanny. Single-chip FPGA implementation of the advanced encryption standard algorithm. In *[Brebner and Woods, 2001]*, pages 152–161, 2001.

M. McLoone and J. V. McCanny. Very high speed 17 Gbps SHACAL encryption architecture. In *[Cheung et al., 2003a]*, pages 111–120, 2003.

S. McMillan and C. Patterson. JBits implementations of the advanced encryption standard (Rijndael). In *[Brebner and Woods, 2001]*, pages 162–171, 2001.

G. M. Megson and I. M. Bland. Mapping a generic systolic array for genetic algorithms onto FPGAs – theory and practice. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and U. Trottenberg, editors, *Parallel Computing: Fundamentals, Applications and New Directions, Proceedings of the Conference ParCo'97*, volume 12, pages 723–726, Amsterdam, 1997. Elsevier, North-Holland.

B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *[Cheung et al., 2003a]*, pages 61–70, 2003.

S. J. Melnikoff, P. James-Roxby, S. F. Quigley, and M. J. Russell. Reconfigurable computing for speech recognition: Preliminary findings. In *[Hartenstein and Grünbacher, 2000]*, pages 495–504, 2000.

S. J. Melnikoff, S. F. Quigley, and M. J. Russell. Implementing a hidden markov model speech recognition system in programmable logic. In *[Brebner and Woods, 2001]*, pages 81–90, 2001.

S. J. Melnikoff, S. F. Quigley, and M. J. Russell. Speech recognition on an FPGA using discrete and continuous hidden markov models. In *[Glesner et al., 2002]*, pages 202–211, 2002.

O. Mencer, N. Boullis, W. Luk, and H. Styles. Parameterized function evaluation for FPGAs. In *[Brebner and Woods, 2001]*, pages 544–554, 2001.

D. Merkle. *Ameisenalgorithmen – Optimierung und Modellierung.* PhD thesis, Institut AIFB, Universität Karlsruhe(TH), 2002.

D. Merkle and M. Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In S. Cagnoni, R. Poli, G. D. Smith, D. Corne, M. Oates, E. Hart, P. L. Lanzi, E. J. Willem, Y . Li, B. Paechter, and T. C. Fogarty, editors, *Real-World Applications of Evolutionary Computing. Proceedings of the EvoWorkshops 2000*, number 1803 in LNCS, pages 287–296. Springer-Verlag, 2000.

D. Merkle and M. Middendorf. Collective optimization: The artificial ants way. In *Workshop From Worker to Colony: Understanding the Organisation of Insect Societies*, Isaac Newton Institute for Mathematical Sciences, Cambridge, 2001a.

D. Merkle and M. Middendorf. Fast ACO optimization on reconfigurable processor arrays. In *8th Reconfigurable Architectures Workshop 2001 (RAW 2001)*, San Francisco, 2001b.

D. Merkle and M. Middendorf. A new approach to solve permutation scheduling problems with ant colony optimization. In *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, number 2037 in LNCS, pages 484–493. Springer-Verlag, 2001c.

D. Merkle and M. Middendorf. Prospects for dynamic algorithm control: Lessons from the phase structure of ant scheduling algorithms. In R. B. Heckendorn, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference – Workshop Program. Workshop: The Next Ten Years of Scheduling Research*, pages 121–126, San Francisco, 2001d.

D. Merkle and M. Middendorf. Fast ant colony optimization on runtime reconfigurable processor arrays. *Genetic Programming and Evolvable Machines*, 3(4): 345–361, 2002a.

D. Merkle and M. Middendorf. Modelling ACO: Composed permutation problems. In *[Dorigo et al., 2002]*, pages 149–162, 2002b.

D. Merkle and M. Middendorf. Modelling the dynamics of ant colony optimization algorithms. *Evolutionary Computation*, 10(3):235–262, 2002c.

D. Merkle and M. Middendorf. An ant algorithm with global pheromone evaluation for scheduling a single machine. *Applied Intelligence*, 18(1):105–111, 2003.

D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 893–900, Las Vegas, Nevada, 2000. Morgan Kaufmann.

D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346, 2002.

U. Meyer-Bäse. Coherent demodulation with FPGAs. In *[Hartenstein and Glesner, 1996]*, pages 166–175, 1996.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1999.

A. Michalski, K. Gaj, and T. El-Ghazawi. An implementation comparison of an IDEA encryption cryptosystem on two general-purpose reconfigurable computers. In *[Cheung et al., 2003a]*, pages 204–219, 2003.

R. Michels and M. Middendorf. An island model based ant system with lookahead for the shortest supersequence problem. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 692–701. Springer-Verlag, 1998.

R. Michels and M. Middendorf. An ant system for the shortest common supersequence problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. Mc Graw Hill, 1999.

M. Middendorf, F. Reischle, and H. Schmeck. Multi colony ant algorithms. *Journal of Heuristics*, 8(3):305–320, 2002.

M. Middendorf, H. Schmeck, H. Schröder, and G. Turner. Multiplication of matrices with different sparseness properties on dynamically reconfigurable meshes. *VLSI Design*, 9:69–81, 1999.

M. Middendorf, H. Schmeck, and G. Turner. Sparse matrix multiplication on a reconfigurable mesh. *The Australian Computer Journal*, 27:37–40, 1995.

R. Miller, V. K. Prasanna, D. I. Reisis, and Q. F. Stout. Parallel computations on reconfigurable meshes. *IEEE Transactions on Computers*, 42(6):678 – 692, 1993.

T. Miomo, K. Yasuoka, and M. Kanazawa. The fastest multiplier on FPGAs with redundant binary representation. In *[Hartenstein and Grünbacher, 2000]*, pages 515–524, 2000.

E. Mirsky and A. DeHon. MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *[Arnold and Pocek, 1995]*, pages 157–166, 1996.

Y. Miyajima and T. Maruyama. A real-time stereo vision system with FPGA. In *[Cheung et al., 2003a]*, pages 448–457, 2003.

T. Miyamori and K. Olukotun. A quantitative analysis of reconfigurable coprocessors for multimedia applications. In *[Pocek and Arnold, 1998]*, pages 2–11, 1998.

K. Miyashita and R. Hashimoto. A Java applet to visualize algorithms on reconfigurable mesh. In J. D. P. Rolim, editor, *Parallel and Distributed Processing, 15 IPDPS 2000 Workshops Proceedings*, volume 1800 of *LNCS*, pages 137–142. Springer-Verlag, 2000.

T. Miyazaki, T. Murooka, N. Takahashi, and M. Hashimoto. Real-time packet editing using reconfigurable hardware for active networking. In *[Leong and Luk, 2002]*, pages 26–33, 2002.

N. D. Monekosso, P. Remagnino, and A. Szarowicz. An improved Q-Learning algorithm using synthetic pheromones. In B. Dunin-Keplicz and E. Nawarecki, editors, *From Theory to Practice in Multi-Agent Systems*, LNCS 2296, pages 197–206. Springer-Verlag, 2002.

W. Moore and W. Luk, editors. *Field-Programmable Logic and Applications. Proceedings of the 5th International Workshop, FPL 1995, Oxford, UK*, volume 975 of *LNCS*. Springer-Verlag, 1995.

C. A. Moritz, D. Yeung, and A. Agarwal. Exploring optimal cost-performance designs for RAW microprocessors. In *[Pocek and Arnold, 1998]*, pages 12–27, 1998.

M. J. Myjak and J. G. Delgado-Frias. Pipelined multipliers for reconfigurable hardware. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2004.

H. Nagano, A. Matsuura, and A. Nagoya. An efficient implementation method of fractal image compression on dynamically reconfigurable architecture. In J. D. P. Rolim et al., editors, *Parallel and Distributed Processing, 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, volume 1586 of *LNCS*, pages 670–678. Springer-Verlag, 1999.

N. Nagata and T. Maruyama. Real-time detection of line segments using the line Hough transform. In *[Diessel and Williams, 2004]*, pages 89–96, 2004.

P. E. Nastou and Y. C. Stamatiou. Dynamically modifiable ciphers using a reconfigurable CAST-128 based algorithm on ATMEL's FPSLIC reconfigurable FPGA architecture. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2002.

G. Navarro Varela and M. C. Sinclair. Ant colony optimisation for virtual-wavelength-path routing and wavelength allocation. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pages 1809–1816. IEEE Press, 1999.

K. Neumann and M. Morlock. *Operations Research*. Hanser, 2002.

N. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi. Implementation of elliptic curve cryptosystems on a reconfigurable computer. In *[Asada and Fujita, 2003]*, pages 60–67, 2003.

O. Nibouche, M. Nibouche, A. Bouridane, and A. Belatreche. Fast architectures for FPGA-based implementation of RSA encryption algorithm. In *[Diessel and Williams, 2004]*, pages 271–278, 2004.

G. Nicolis and I. Priogogine. *Self-Organization in Non-Equilibrium Systems*. Wiley & Sons, 1977.

J. Oliver and V. Akella. Improving DSP performance with a small amount of field programmable logic. In *[Cheung et al., 2003a]*, pages 520–532, 2003.

T. Oliver, B. Schmidt, and D. Maskell. Hyper customized processors for bio-sequence database scanning on FPGAs. In *[Schmit and Wilton, 2005]*, pages 229–237, 2005.

G. C. Onwubolu. Ant system approach for the flowshop problem with bicriteria of makespan and total flow time minimisation. In *[Dorigo et al., 2000]*, 2000.

Y. Osana, T. Fukushima, and H. Amano. Implementation of ReCSiP: A reconfigurable cell simulation platform. In *[Cheung et al., 2003a]*, pages 766–775, 2003.

Y. Pan. Constant-time Hough transform on a 3D reconfigurable mesh using fewer processors. In J. D. P. Rolim, editor, *Parallel and Distributed Processing, 15 IPDPS 2000 Workshops Proceedings*, volume 1800 of *LNCS*, pages 966–973. Springer-Verlag, 2000.

Y. Pan, J. Li, and R. Vemuri. Continuous wavelet transform on reconfigurable meshes. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*. IEEE Computer Society, 2001.

G. Panneerselvam, P. J. W. Graumann, and L. E. Turner. Implementation of fast fourier transforms and discrete cosine transforms in FPGAs. In *[Moore and Luk, 1995]*, pages 272–281, 1995.

R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6 (4):321–332, 2002.

S. Paschalakis and M. Bober. A low cost FPGA system for high speed face detection and tracking. In *[Asada and Fujita, 2003]*, pages 214–221, 2003.

R. Payne. Run-time parameterised circuits for the Xilinx XC6200. In *[Luk et al., 1997a]*, pages 161–172, 1997.

B. Pfahringer. Multi-agent search for open shop scheduling: Adapting the Ant-Q formalism. Technical Report TR-96-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1996.

T. Pionteck, L.D. Kabulepa, and M. Glesner. Reconfiguration requirements for high speed wireless communication systems. In *[Asada and Fujita, 2003]*, pages 118–125, 2003.

M. Platzner. Reconfigurable accelerators for combinatorial problems. *Computer*, 33(4):58–60, 2000.

C. Plessl and M. Platzner. Instance-Specific Accelerators for Minimum Covering. In *Proceedings of the 1st International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pages 85–91, Las Vegas, Nevada, USA, 2001. CSREA Press.

H. Ploog, M. Schmalisch, and D. Timmermann. Security upgrade of existing ISDN devices by using reconfigurable logic. In *[Hartenstein and Grünbacher, 2000]*, pages 505–514, 2000.

K. L. Pocek and J. M. Arnold, editors. *Proceedings of the 1997 IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, USA*. IEEE Computer Society, 1997.

K. L. Pocek and J. M. Arnold, editors. *Proceedings of the 1998 IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, USA*. IEEE Computer Society, 1998.

K. L. Pocek and J. M. Arnold, editors. *Proceedings of the 1999 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 1999, Napa Valley, California, USA*. IEEE Computer Society, 1999.

K. L. Pocek and J. M. Arnold, editors. *Proceedings of the 2001 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2001, Napa Valley, California, USA*. IEEE Computer Society, 2001.

K. L. Pocek and J. M. Arnold, editors. *Proceedings of the 2003 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2003, Napa Valley, California, USA*. IEEE Computer Society, 2003.

K. L. Pocek and J. M. Arnold, editors. *Proceedings of the 2004 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2003, Napa Valley, California, USA*. IEEE Computer Society, 2004.

I. Pournara, C.-S. Bouganis, and G. A. Constantinides. FPGA-accelerated bayesian learning for reconstruction of gene regulatory networks. In *[Rissa et al., 2005]*, 2005.

N. Pramstaller and J. Wolkerstorfer. A universal and efficient AES co-processor for field programmable logic arrays. In *[Becker et al., 2004]*, pages 565–574, 2004.

D. V. Pryor, M. R. Thistle, and N. Shirazi. Text searching on Splash 2. In *[Buell and Pocek, 1993]*, pages 172–177, 1993.

K. Puttegowda, D. I. Lehn, J. H. Park, P. M. Athanas, and M. T. Jones. Context switching in a run-time reconfigurable system. *The Journal of Supercomputing*, 26(3):239–257, 2003.

J. M. Rabaey. Silicon platforms for the next generation wireless systems - what role does reconfigurable hardware play? In *[Hartenstein and Grünbacher, 2000]*, pages 277–285, 2000.

J.-M. Raczinski and S. Sladek. The modular architecture of SYNTHUP, FPGA based PCI board for real-time sound synthesis and digital signal processing. In *[Hartenstein and Grünbacher, 2000]*, pages 834–837, 2000.

A. Rahman and V. Polavarapuv. Evaluation of low-leakage design techniques for field programmable gate arrays. In *[Tessier and Schmit, 2004]*, pages 23–30, 2004.

M. Rahoual, R. Hadji, and V. Bachelet. Parallel ant system for the set covering problem. In *[Dorigo et al., 2002]*, pages 262–267, 2002.

S. Ramachandran and S. Srinivasan. FPGA implementation of a novel, fast motion estimation algorithm for real-time video compression. In *[Schlag and Tessier, 2001]*, pages 213–219, 2001.

J. Ramírez, A. García, P. G. Fernández, L. Parrilla, and A. Lloris-Ruíz. Analysis of RNS-FPL synergy for high throughput DSP applications: Discrete wavelet transform. In *[Hartenstein and Grünbacher, 2000]*, pages 342–351, 2000.

M. Randall and A. Lewis. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing*, 62(9):1421–1432, 2002.

N. K. Ratha, A. K. Jain, and D. T. Rover. Convolution on Splash 2. In *[Athanas and Pocek, 1995]*, pages 204–213, 1995.

R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 172–180. ACM/IEEE, 1994.

I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach den Prinzipien der biologischen Information.* Fromman Verlag, 1973.

M. Redekopp and A. Dandalis. A parallel pipelined SAT solver for FPGAs. In *[Hartenstein and Grünbacher, 2000]*, pages 462–468, 2000.

M. Reimann, K. Doerner, and R. F. Hartl. Insertion based ants for vehicle routing problems with backhauls and time windows. In *[Dorigo et al., 2002]*, pages 135–148, 2002a.

M. Reimann, M. Stummer, and K. Doerner. A savings based ant system for the vehicle routing problem. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 1317–1325. Morgan Kaufmann, 2002b.

G. Reinelt. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *LNCS*. Springer-Verlag, 1994.

M. Rencher and B. L. Hutchings. Automated target recognition on SPLASH 2. In *[Pocek and Arnold, 1997]*, pages 192–200, 1997.

J. Resano, D. Mozos, and F. Catthoor. A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE 2005)*, pages 106–111. IEEE Computer Society, 2005.

X. Revés, A. Gelonch, F. Casadevall, and J. L. García. Software radio reconfigurable hardware system (SHaRe). In *[Hartenstein and Grünbacher, 2000]*, pages 332–341, 2000.

T. Rissa, S. Wilton, and P. Leong, editors. *Field-Programmable Logic and Applications. CD-ROM Proceedings of the 15th International Conference, FPL 2005, Tampere, Finland*. IEEE Computer Society, 2005.

J. Ritter and P. Molitor. A pipelined architecture for partitioned DWT based lossy image compression using FPGA's. In *[Schlag and Tessier, 2001]*, pages 201–206, 2001.

N. Roma, T. Dias, and L. Sousa. Customisable core-based architectures for real-time motion estimation on FPGAs. In *[Cheung et al., 2003a]*, pages 745–754, 2003.

J. Rose and C. Ebeling, editors. *FPGA 1996: Proceedings of the 1996 ACM/SIGDA 4th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 1996.

J. Rose and A. Sangiovanni-Vincentelli, editors. *FPGA 1994: Proceedings of the 1994 ACM/SIGDA 2nd International Symposium on Field Programmable Gate Arrays, Berkeley, California, USA*. ACM Press, 1994.

G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Design strategies and modified descriptions to optimize cipher FPGA implementations: Fast and compact results for DES and triple-DES. In *[Cheung et al., 2003a]*, pages 181–193, 2003.

K. Rowley and C. Lyden. Implementing sigma delta modulator prototype designs on an FPGA. In *[Hartenstein and Glesner, 1996]*, pages 389–393, 1996.

R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:127–190, 1999.

C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. M. Arnold, and M. Gokhale. The NAPA adaptive processing architecture. In *[Pocek and Arnold, 1998]*, pages 28–37, 1998.

G. P. Saggese, A. Mazzeo, N. Mazzocca, and A. G. M. Strollo. An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In *[Cheung et al., 2003a]*, pages 292–302, 2003.

S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

M. Saito, H. Fujisawa, N. Ujiie, and H. Yoshizawa. Cluster architecture for reconfigurable signal processing engine for wireless communication. In *[Rissa et al., 2005]*, 2005.

T. Saito, T. Maruyama, T. Hoshino, and S. Hirano. A music synthesizer on FPGA. In *[Brebner and Woods, 2001]*, pages 377–387, 2001.

O. Sammoud, C. Solnon, and K. Ghédira. Ant algorithm for the graph matching problem. In G. R. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *Springer-Verlag*, pages 213–223, 2005.

C. Sanz, L. de Zulueta, and J. M. Meneses. FPGA implementation of the block-matching algorithm for motion estimation in image coding. In *[Hartenstein and Glesner, 1996]*, pages 146–155, 1996.

N. A. Saqib, F. Rodríguez-Henríquez, and A. Díaz-Pérez. Two approaches for a single-chip FPGA implementation of an encryptor/decryptor AES core. In *[Cheung et al., 2003a]*, pages 303–312, 2003.

T. Saxe and B. Faith. Less is more with FPGAs. *EE Times*, 2004. URL: http://www.eetimes.com/showArticle.jhtml?articleID=47203801.

S. M. Scalera and J. R. Vazquez. The design and implementation of a context switching FPGA. In *[Pocek and Arnold, 1998]*, pages 78–85, 1998.

G. Schelle and D. Grunwald. CUSP: a modular framework for high speed network applications on FPGAs. In *[Schmit and Wilton, 2005]*, pages 246–257, 2005.

B. Scheuermann. FPGA task arrangement with genetic algorithms. diploma thesis, Institute AIFB, University of Karlsruhe, Germany, 1999.

B. Scheuermann, M. Guntsch, M. Middendorf, and H. Schmeck. Time-Scattered Heuristic for the hardware implementation of Population-based ACO. In *[Dorigo et al., 2004]*, pages 250–261, 2004a.

B. Scheuermann, S. Janson, and M. Middendorf. On accelerating ant algorithms in hardware — studies with counter-based ant colony optimization. *Genetic Programming and Evolvable Machines*, 2005. submitted.

B. Scheuermann and M. Middendorf. Counter-based ant colony optimization as a hardware-oriented meta-heuristic. In F. Rothlauf et al., editors, *Applications of Evolutionary Computing. Proceedings of EvoWorkshops 2005*, volume 3449 of *LNCS*, pages 235–244. Springer-Verlag, 2005.

B. Scheuermann, K. So, M. Guntsch, M. Middendorf, O. Diessel, H. ElGindy, and H. Schmeck. FPGA implementation of population based ant colony optimization. In *Proceedings of the Dagstuhl Seminar (03301) Dynamically Reconfigurable Architectures*, 2003. URL: http://www.dagstuhl.de/03301/Materials.

B. Scheuermann, K. So, M. Guntsch, M. Middendorf, O. Diessel, H. ElGindy, and H. Schmeck. FPGA implementation of population-based ant colony optimization. *Applied Soft Computing*, 4:303–322, 2004b.

M. Schlag and R. Tessier, editors. *FPGA 2001: Proceedings of the 2001 ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 2001.

M. Schlag and S. Trimberger, editors. *FPGA 2002: Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 2002.

H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor. PipeRench: A virtualized programmable datapath in 0.18 micron technology. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, pages 63–66, 2002.

H. Schmit and S. Wilton, editors. *FPGA 2005: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 2005.

B. Schoner, J. D. Villasenor, S. Molloy, and R. Jain. Techniques for FPGA implementation of video compression systems. In *[Chan and Rose, 1995]*, pages 154–159, 1995.

R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents*, pages 209–216, 1997.

R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5:168–207, 1996.

H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, 1981.

S. D. Scott, S. Seth, and A. Samal. HGA: A hardware-based genetic algorithm. In *[Chan and Rose, 1995]*, pages 53–59, 1995.

S. D. Scott, S. Seth, and A. Samal. Hardware engine for genetic algorithms. Technical Report UNL-CSE-97-001, University of Nebraska-Lincon, 1997a.

S. D. Scott, S. Seth, and A. Samal. A synthesizable VHDL coding of a genetic algorithm. Technical Report UNL-CSE-97-009, University of Nebraska-Lincon, 1997b.

N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk. A reconfigurable platform for real-time embedded video image processing. In *[Cheung et al., 2003a]*, pages 606–615, 2003.

S. P. Seng, W. Luk, and P. Y. K. Cheung. Flexible instruction processors. In *Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 193–200. ACM Press, 2000.

S. P. Seng, W. Luk, and P. Y. K. Cheung. Run-time adaptive flexible instruction processors. In *[Glesner et al., 2002]*, pages 545–555, 2002.

B. Shackleford, E. Okushi, M. Yasuda, H. Koizumi, K. Seo, T. Iwamoto, and H. Yasuura. An FPGA-based genetic algorithm machine (poster abstract). In *[Trimberger and Hauck, 2000]*, page 218, 2000.

B. Shackleford, M. Tanaka, R. J. Carter, and G. Snider. FPGA implementation of neighborhood-of-four cellular automata random number generators. In *[Schlag and Trimberger, 2002]*, pages 106–112, 2002.

A. M. Shankiti and M. Leeser. Implementing a RAKE receiver for wireless communications on an FPGA-based computer system. In *[Trimberger and Hauck, 2000]*, pages 145–151, 2000.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

S. Sheidaei, H. Noori, A. Akbari, and H. Pedram. Motivation from a full-rate specific design to a DSP core approach for GSM vocoders. In *[Brebner and Woods, 2001]*, pages 388–397, 2001.

N. Shirazi, P. M. Athanas, and A. L. Abbott. Implementation of a 2-D fast fourier transform on an FPGA-based custom computing machine. In *[Moore and Luk, 1995]*, pages 282–292, 1995a.

N. Shirazi, W. Luk, and P. Y. K. Cheung. Automating production of run-time reconfigurable designs. In *[Pocek and Arnold, 1998]*, pages 147–156, 1998.

N. Shirazi, A. Walters, and P. Athanas. Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In *[Athanas and Pocek, 1995]*, pages 155–162, 1995b.

A. Shmygelska, R. Aguirre-Hernández, and H. H. Hoos. An ant colony optimization algorithm for the 2D HP protein folding problem. In *[Dorigo et al., 2002]*, pages 14–27, 2002.

N. Sidahao, G. A. Constantinides, and P. Y. K. Cheung. Multiple restricted multiplication. In *[Becker et al., 2004]*, pages 374–383, 2004.

R. P. S. Sidhu, A. Mei, and V. K. Prasanna. Genetic programming using self-reconfigurable FPGAs. In *[Lysaght et al., 1999]*, pages 301–312, 1999a.

R. P. S. Sidhu, A. Mei, and V. K. Prasanna. String matching on multicontext FPGAs using self-reconfiguration. In *[Kaptanoglu and Trimberger, 1999]*, pages 217–226, 1999b.

A. Simpson, J. Hunter, M. Wylie, Y. Hu, and D. Mann. Demonstrating real-time JPEG image compression-decompression using standard component IP cores on a programmable logic based platform for DSP and image processing. In *[Brebner and Woods, 2001]*, pages 441–450, 2001.

H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Filho. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Transactions on Computers*, 49(5):465–481, 2000.

I. Skliarova and A. B. Ferrari. A hardware/software approach to accelerate boolean satisfiability. In *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, pages 270–277, 2002.

V. Sklyarov, I. Skliarova, B. Pimentel, and J. Arrais. Hardware/software implementation of FPGA-targeted matrix-oriented SAT solvers. In *[Becker et al., 2004]*, pages 922–926, 2004.

K. Socha, J. Knowles, and M. Sampels. A MAX-MIN ant system for the university course timetabling problem. In *[Dorigo et al., 2002]*, pages 1–13, 2002.

K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori, editors, *Applications of Evolutionary Computing, Proceedings of Evo Workshops 2003*, volume 2611 of *LNCS*, pages 334–345. Springer-Verlag, 2003.

C. Solnon. Solving permutation constraint satisfaction problem with artificial ants. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 118–122. IOS Press, 2000.

C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, 2002.

H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using FPGA. In *[Schmit and Wilton, 2005]*, pages 238–245, 2005.

E. Sotiriades, A. Dollas, and P. Athanas. Hardware-software codesign and parallel implementation of a golomb ruler derivation engine. In *[Hutchings, 2000]*, pages 227–235, 2000.

I. Sourdis and D. Pnevmatikatos. Fast, large-scale string match for a 10Gbps FPGA-based network intrusion detection system. In *[Cheung et al., 2003a]*, pages 880–889, 2003.

L. Sousa, P. Tomás, F. Pelayo, A. Martinez, C. A. Morillas, and S. Romero. An FPL bioinspired visual encoding system to stimulate cortical neurons in real-time. In *[Cheung et al., 2003a]*, pages 691–700, 2003.

N. Srivastava, J. L. Trahan, R. Vaidyanathan, and S. Rai. Adaptive image filtering using run-time reconfiguration. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2003.

F.-X. Standaert, S. B. Örs, J.-J. Quisquater, and B. Preneel. Power analysis attacks against FPGA implementations of the DES. In *[Becker et al., 2004]*, pages 84–94, 2004.

F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat. A methodology to implement block ciphers in reconfigurable hardware and its application to fast and compact AES Rijndael. In *[Trimberger and Tessier, 2003]*, pages 216–224, 2003a.

F.-X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J.-J. Quisquater. Power analysis of FPGAs: How practical is the attack? In *[Cheung et al., 2003a]*, pages 701–711, 2003b.

C. Steckel, M. Middendorf, H. ElGindy, and H. Schmeck. A simulator for the reconfigurable mesh architecture. In J. Rolim, editor, *Parallel and Distributed Computing, Proceedings of the 10 IPPS/SPDP'98 Workshops, 5th Reconfigurable Architectures Workshop RAW-98*, LNCS 1388, pages 99–104. Springer-Verlag, 1998.

G. Stitt, F. Vahid, and S. Nematbakhsh. Energy saving and speedups from partitioning critical software loops to hardware in embedded systems. *ACM Transactions on Embedded Computing Systems*, 3(1):218–232, 2004.

P. Stogiannos, A. Dollas, and V. Digalakis. A configurable logic based architecture for real-time continuous speech recognition using hidden Markov models. *Journal of VLSI Signal Processing Systems*, 24(2-3):223–240, 2000.

R. Stubs, P. Heitor, S. Lopes, and A. A. Freitas. Antminer: an ant colony based system for mining medical data. In L. Spector et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2001*, pages 791–797, San Francisco, 2001. Morgan Kaufmann.

T. Stützle. MAX-MIN ant system for the quadratic assignment problem. Technical Report AIDA-97-4, FG Intellektik, FB Informatik, TU Darmstadt, 1997.

T. Stützle. An ant approach for the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT '98)*, volume 3, pages 1560–1564. Verlag Mainz, 1998a.

T. Stützle. Parallelization strategies for ant colony optimization. In A. E. Eiben, T. Bäck, M. Schonauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V*, LNCS 1498, pages 722–731. Springer-Verlag, 1998b.

T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, volume 220 of *DISKI*. Infix, 1999.

T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw-Hill, 1999.

T. Stützle and M. Dorigo. A short convergence proof for a class of ACO algorithms. *IEEE Transaction on Evolutionary Computation*, 6(4):358–365, 2002.

T. Stützle and H. H. Hoos. Improving the ant system: A detailed report on the MAX-MIN ant system. Technical Report AIDA-96-12, FG Intellektik, FB Informatik, TU Darmstadt, 1996.

T. Stützle and H. H. Hoos. The MAX-MIN ant system and local search for the traveling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97*, pages 309–314. IEEE Press, 1997.

T. Stützle and H. H. Hoos. MAX-MIN ant system. *Future Generation Computer Systems Journal*, 16(8):889–914, 2000.

D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 832–838. Morgan Kaufmann, 1997.

Y. Sugawara, M. Inaba, and K. Hiraki. Over 10Gbps string matching mechanism for multi-stream packet scanning systems. In *[Becker et al., 2004]*, pages 484–493, 2004.

Y. Sugawara, M. Inaba, and K. Hiraki. High-speed and memory efficient TCP stream scanning using FPGA. In *[Rissa et al., 2005]*, 2005.

T. Suyama, M. Yokoo, and A. Nagoya. Solving satisfiability problems on FPGAs using experimental unit propagation heuristic. In J. D. P. Rolim et al., editors, *Parallel and Distributed Processing, 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, volume 1586 of *LNCS*, pages 709–711. Springer-Verlag, 1999.

T. Suyama, M. Yokoo, and H. Sawada. Solving satisfiability problems on FPGAs. In *[Hartenstein and Glesner, 1996]*, pages 136–145, 1996.

T. Suyama, M. Yokoo, and H. Sawada. Solving satisfiability problems using logic synthesis and reconfigurable hardware. In *Proceedings 31st Annual Hawaii International Conference on System Sciences*, 1998.

T. Suyama, M. Yokoo, H. Sawada, and A. Nagoya. Solving satisfiability problems using reconfigurable computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):109–116, 2001.

S. Swaminathan, R. Tessier, D. Goeckel, and W. Burleson. A dynamically reconfigurable adaptive viterbi decoder. In *[Schlag and Trimberger, 2002]*, pages 227–236, 2002.

W. Szwarc, A. Grosso, and F. Della Croce. Algorithmic paradoxes of the single machine total tardiness problem. *Journal of Scheduling*, 4(2):93–104, 2001.

E. D. Taillard. FANT: Fast ant system. Technical Report IDSIA-46-98, IDSIA, Lugano, Switzerland, 1998.

E. D. Taillard and L. M. Gambardella. An ant approach for structured quadratic assignment problems. In *Second Metaheuristic International Conference*, Sophia-Antipolis, 1997.

E-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel ant colonies for combinatorial optimization problems. In J. Rolim et al., editors, *Parallel and Distributed Processing, 11 IPPS/SPDP'99 Workshops*, number 1586 in LNCS, pages 239–247. Springer-Verlag, 1999.

S. H. Tang, K. S. Tsui, and P. H. W. Leong. Modular exponentiation using parallel multipliers. In *[Asada and Fujita, 2003]*, pages 52– 59, 2003.

C. J. Tavares, C. Bungardean, G.M. Matos, and J.T. de Sousa. Solving SAT with a context-switching virtual clause pipeline and an FPGA embedded processor. In *[Becker et al., 2004]*, pages 344–353, 2004.

M. Taylor, J. Kim, J. Miller, D. Wentzla, F. Ghodrat, B. Greenwald, H. Ho, M. Lee, P. Johnson, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Frank, S. Amarasinghe, and A. Agarwal. The RAW microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

T. Teich, M. Fischer, A. Vogel, and J. Fischer. A new ant colony algorithm for the job shop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 803, 2001.

J. Teifel and R. Manohar. Programmable asynchronous pipeline arrays. In *[Cheung et al., 2003a]*, pages 345–354, 2003.

N. Telle, W. Luk, and R. C. C. Cheung. Customising hardware designs for elliptic curve cryptography. In Andy D. Pimentel and Stamatis Vassiliadis, editors, *Computer Systems: Architectures, Modeling, and Simulation, Third and Fourth International Workshops, SAMOS 2003 and SAMOS 2004*, volume 3133 of *LNCS*, pages 274–283. Springer-Verlag, 2004.

R. Tessier and H. Schmit, editors. *FPGA 2004: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 2004.

Y. Thoma, E. Sanchez, J.-M. M. Arostegui, and G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. In *[Cheung et al., 2003a]*, pages 681–690, 2003.

L.-K. Ting, R. Woods, and C. Cowan. Virtex implementation of pipelined adaptive lms predictor in electronic support measures receiver. In *[Brebner and Woods, 2001]*, pages 367–376, 2001.

T.J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung. Reconfigurable computing: Architectures and design methods. *IEE Proceedings Computers and Digital Techniques*, 152(2):193– 207, 2005.

D. K. Y. Tong, P. S. Lo, K. H. Lee, and P. H. W. Leong. A system level implementation of Rijndael on a memory-slot based FPGA card. In *[Leong and Luk, 2002]*, pages 102–109, 2002.

J. L. Trahan, R. Vaidyanathan, and R. K. Thirachelvan. On the power of segementing and fusing buses. *Journal of Parallel and Distributed Computing*, 34 (1):82–94, 1996.

S. Trimberger, D. Carberry, A. Johnson, and J. Wong. A time-multiplexed FPGA. In *[Pocek and Arnold, 1997]*, pages 22–28, 1997.

S. Trimberger and S. Hauck, editors. *FPGA 2000: Proceedings of the 2000 ACM/SIGDA 8th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 2000.

S. Trimberger and R. Tessier, editors. *FPGA 2003: Proceedings of the 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*. ACM Press, 2003.

R.H. Turner, R. Woods, and T. Courtney. Multiplier-less realization of a polyphase filter using LUT-based FPGAs. In *[Glesner et al., 2002]*, pages 192–201, 2002.

K. Underwood. FPGAs vs. CPUs: trends in peak floating-point performance. In *[Tessier and Schmit, 2004]*, pages 171–180, 2004.

I. S. Uzun, A. Amira, and A. Bouridane. FPGA implementations of fast fourier transforms for real-time signal and image processing. In *[Asada and Fujita, 2003]*, pages 102–109, 2003.

F. Vahid. I/O and performance tradeoffs with the FunctionBus during multi-FPGA partitioning. In *[Ebeling and Cong, 1997]*, pages 27–34, 1997.

R. van der Put. Routing in the faxfactory using mobile agents. Technical Report R&D-SV-98-276, KPN Research, The Netherlands, 1998.

M. Verderber, A. Zemva, and A. Trost. HW/SW codesign of the MPEG-2 video decoder. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2003.

F.-J. Veredas, M. Scheppler, W. Moffat, and B. Mei. Custom implementation of the coarse-grained reconfigurable adres architecture for multimedia purposes. In *[Rissa et al., 2005]*, 2005.

A. Vesel and J. Zerovnik. How good can ants color graphs? *Journal of Computing and Information Technology - CIT*, 8:131–136, 2000.

J. Villasenor, B. Schoner, C. Kang-Ngee, C. Zapata, K. J. Hea, C. Jones, S. Lansing, and B. Mangione-Smith. Configurable computing solutions for automatic target recognition. In *[Arnold and Pocek, 1995]*, pages 70–79, 1996.

C. Visavakul, P. Y. K. Cheung, and W. Luk. A digit-serial structure for reconfigurable multipliers. In *[Brebner and Woods, 2001]*, pages 565–573, 2001.

N. Voß and B. Mertsching. Design and implementation of an accelerated gabor filter bank using parallel hardware. In *[Brebner and Woods, 2001]*, pages 451–460, 2001.

J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard. Programmable active memories: Reconfigurable systems come of age. *IEEE Transactions on VLSI Systems*, 4(1):56–69, 1996.

A. Wade and S. Salhi. An ant system algorithm for the vehicle routing problem with backhauls. In J. P. de Sousa, editor, *Proceedings of the 4th Metaheuristics International Conference, MIC 2001*, pages 199–204, Porto, Portugal, 2001.

S. Wakabayashi and K. Kikuchi. An instance-specific hardware algorithm for finding a maximum clique. In *[Becker et al., 2004]*, pages 516–525, 2004.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

M. Weinhardt and W. Luk. Pipeline vectorization for reconfigurable systems. In *[Pocek and Arnold, 1999]*, pages 52–62, 1999.

T. White, B. Pagurek, and F. Oppacher. Connection management using adaptive mobile agents. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 802–809. CSREA Press, 1998.

J. A. Williams, A. S. Dawood, and S. J. Visser. FPGA-based cloud detection for real-time onboard remote sensing. In *[Leong and Luk, 2002]*, pages 110–116, 2002.

R. S. Williams and P. J. Kuekes. Molecular nanoelectronics. In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 5–7. IEEE Computer Society, 2000.

K. E. Wires, M. J. Schulte, and D. McCarley. FPGA resource reduction through truncated multiplication. In *[Brebner and Woods, 2001]*, pages 574–583, 2001.

M. J. Wirthlin and B. McMurtrey. Efficient constant coefficient multiplication using advanced FPGA architectures. In *[Brebner and Woods, 2001]*, pages 555–564, 2001.

R. Wittig and P. Chow. OneChip: An FPGA processor with reconfigurable logic. In *[Arnold and Pocek, 1995]*, pages 126–135, 1996.

P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW), Abstracts and CD-ROM*. IEEE Computer Society, 2005.

C. Wong, A. J. Martin, and P. Thomas. An architecture for asynchronous FPGAs. In *Proceedings of International Conference on Field Programmable Technology*, pages 170–177. IEEE Computer Society, 2003.

Xilinx Inc. XC6200 advanced product specification. In the programmable logic data book, 1996.

Xilinx Inc. Virtex 2.5V FPGA detailed functional description, 2002.

Xilinx Inc. Spartan-II 2.5V FPGA family: Complete data sheet, 2004.

Xilinx Inc. MicroBlaze processor reference guide, 2005a.

Xilinx Inc. PowerPC 405 processor block reference guide, 2005b.

Xilinx Inc. Spartan-3 FPGA family: Complete data sheet, 2005c.

Xilinx Inc. Virtex-4 user guide, 2005d.

Xilinx Inc. Virtex-II platform FPGA user guide, 2005e.

Xilinx Inc. Virtex-II Pro and Virtex-II Pro X FPGA user guide, 2005f.

H. Yamada, T. Tominaga, and M. Ichikawa. An autonomous flying object navigated by real-time optical flow and visual target detection. In *[Asada and Fujita, 2003]*, pages 222–227, 2003.

Y. Yamaguchi, T. Maruyama, and A. Konagaya. Three-dimensional dynamic programming for homology search. In *[Becker et al., 2004]*, pages 505–515, 2004.

R. H. C. Yap, S. Z. Q. Wang, and M. J. Henz. Hardware implementations of real-time reconfigurable WSAT variants. In *[Cheung et al., 2003a]*, pages 488–496, 2003.

A. G. Ye and D. M. Lewis. Procedural texture mapping on FPGAs. In *[Kaptanoglu and Trimberger, 1999]*, 1999.

C.-H. Yeh, B. Parhami, H. Lee, and E. A. Varvarigos. $2.5n$-step sorting on $n \times n$ meshes in the presence of $o(\sqrt{n})$ worst-case faults. In *Proceedings of the 13th International Parallel Processing Symposium, 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP '99)*, pages 436–440. IEEE Computer Society, 1999.

K. S. Yeung and S. C. Chan. Multiplier-less FIR digital filters using programmable sum-of-power-two (SOPOT) coefficients. In *[Leong and Luk, 2002]*, pages 78–84, 2002.

M. Yoshimi, Y. Osana, T. Fukushima, and H. Amano. Stochastic simulation for biochemical reactions on FPGA. In *[Becker et al., 2004]*, pages 105–114, 2004.

S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget, and D. Levi. A high I/O reconfigurable crossbar switch. In *[Pocek and Arnold, 2003]*, pages 3–10, 2003.

W. H. Yung, Y. W. Seung, K. H. Lee, and P. H. W. Leong. A runtime reconfigurable implementation of the GSAT algorithm. In *[Lysaght et al., 1999]*, pages 526–531, 1999.

A. Zakerolhosseini, P. Lee, and E. Horne. An FPFA based object recognition machine. In *[Hartenstein and Keevallik, 1998]*, pages 228–237, 1998.

J. Zambreno, D. Nguyen, and A. Choudhary. Exploring area/delay tradeoffs in an AES FPGA implementation. In *[Becker et al., 2004]*, pages 575–585, 2004.

P. Zhong, P. Ashar, S. Malik, and M. Martonosi. Using reconfigurable computing techniques to accelerate problems in the CAD domain: A case study with boolean satisfiability. In *Design Automation Conference*, pages 194–199, 1998.

J. Zhu and G. J. Milne. Implementing Kak neural networks on a reconfigurable computing platform. In *[Hartenstein and Grünbacher, 2000]*, pages 260–269, 2000.

J. Zhu, G. J. Milne, and B. K. Gunther. Towards an FPGA based reconfigurable computing environment for neural network implementations. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN'99)*, volume 2, pages 661–666. IEEE Computer Society, 1999.

J. Zhu and P. Sutton. An FPGA implementation of Kak's instantaneously-trained, fast-classification neural networks. In *[Asada and Fujita, 2003]*, pages 126–133, 2003.

L. Zhuo and V. K. Prasanna. Sparse matrix-vector multiplication on FPGAs. In *[Schmit and Wilton, 2005]*, pages 63–74, 2005.

P. Zipf, O. Soffke, A. Schumacher, R. Dogaru, and M. Glesner. Programmable and reconfigurable hardware architectures for the rapid prototyping of cellular automata. In *[Rissa et al., 2005]*, 2005.

Index

accelerating configuration speed 45
    caching 46
    compression 45
    prefetching 45
    rearrangement and defragmentation 46
ACO . . . . . *see* Ant Colony Optimization
ACO algorithm 16
agent
    natural 8
    software 12
algorithm step 68
ant
    active 68
    Argentine 9, 10
    artificial 12, 17
    biological 9
    Iridomyrmex humilis 9, 10
    Lasius niger 9, 12
ant algorithm 7, 16
Ant Colony Optimization 15, 16
Ant Colony System 22, 27
Ant System 22, 24, 26
    Ant-cycle 27
    Ant-density 27
    Ant-quantity 27
Ant-Q 22, 27
ANTS 24
Application-Specific Integrated Circuit 32
applications
    ACO 19
    reconfigurable computing 46
AS . . . . . *see* Ant System
AS-rank 22, 27
ASIC . *see* Application-Specific Integrated Circuit

buffer
    H-Buffer 117
    P-Buffer 96, 103, 117
    PH-Buffer 117
Buffer Module 98

C-ACO . . . . . *see* Counter-based ACO
candidate list 122
CDS . . . . . *see* decision sequence, cyclic
CLB . . . . . *see* configurable logic block
clock frequency 110
communication 8, 9
    direct 8
    indirect 8
compaction 134
comparison mode
    generational 72, 76
    non-generational 77
comparison queue 77
comparison, type of
    AVERAGE 78
    BEST 78
configurable logic block 35, 40
constant coefficient multiplier 45, 140
Counter-based ACO 56
    algorithm 56
    mapping 92
critical path 111
Cross-Entropy-Method 30
crossbar switch 106

daemon action 19

decision sequence
cyclic . . . 86, 134
fixed . . . 85
random . . . 86
sorted fixed . . . 86
differential path length . . . 14
double bridge . . . 10
DTSP . . . *see* Dynamic TSP
dynamic changes
of problem parameters . . . 139
of problem size . . . 137
dynamic optimization problem . . . 137
Dynamic TSP . . . 21, 137

elitism . . . 27, 98
emergence . . . 8
entropy
heuristic . . . 121
pheromone . . . 83
eusocial insects . . . 7
evaluation . . . 18
offline . . . 18
online . . . 18, 71
Evaluation Module . . . 98, 104
evaporation
artificial . . . 12, 14
global . . . 14
local . . . 56, 59
natural . . . 9
rate . . . 14
Evolvable Hardware . . . 49
execution mode
parallel . . . 69
sequential . . . 68
experiments, biological . . . 10
exploitation . . . 18
exploration . . . 14, 18

FANT . . . *see* Fast Ant System
Fast Ant System . . . 24
FDS . . . *see* decision sequence, fixed
feedback
negative . . . 9, 19
positive . . . 9, 19
Field-Programmable Gate Array 29, 33, 52
restrictions . . . 53
technology . . . 35
flexibility . . . 8
FPGA . . . *see* Field-Programmable Gate Array

GBAS . . . *see* Graph-based Ant System
general-purpose processors . . . 32
generation . . . 68
generation size . . . 68
Generator Module . . . 98, 99
granularity . . . 34, 38
coarse-grained . . . 34, 39
fine-grained . . . 34, 38
medium-grained . . . 34, 38
mixed-grained . . . 34, 39
Graph-based Ant System . . . 30

Hamiltonian cycle . . . 21
hardware core . . . 50
HAS . . . *see* Hybrid AS
heuristic information . . . 18, 65, 114
η-heuristics . . . 65
τ-heuristic . . . 66
dynamic . . . 18
earliest due date rule . . . 26
for QAP . . . 25
for SMTTP . . . 26
for TSP . . . 24, 139
INTVAL . . . 66
modified due date rule . . . 26
POTVAL . . . 66
REALVAL . . . 65
static . . . 18
heuristic-matrix . . . 115, 119
heuristic-vector . . . 115, 117
active . . . 115
shift . . . 115, 116
heuristics . . . 16
host coupling . . . 33, 36
attached processing unit . . . 37
co-processor . . . 37
embedded processing unit . . . 37

reconfigurable functional unit 37
stand-alone processing unit 37
Hybrid AS 24
Hybrid System Architecture Model 40
Hyper-cube Framework for ACO 30
hyper-reconfiguration 45
HySAM . *see* Hybrid System Architecture Model

IIE ... *see* pheromone encoding, item-item
intellectual property 50
IP ..... *see* intellectual property

local convergence 55, 141
local optimizer 19
log-time delay model 42
look-up table 36, 108
LUT ..... *see* look-up table

Makimoto's wave 32
match buffer 96
matching result 96, 117
MAX-MIN Ant System . 22, 24, 27, 30, 58
memory 34, 52
meta-heuristic 16
MMAS ..... *see* MAX-MIN Ant System
modeling ant behavior 12

neighborhood 13, 17
NP-hard problems 15

Organic Computing 9

P-ACO ..... *see* Population-based ACO
parallel ACO
hardware 29, 68
software 28
partial evaluation 44, 140
partitioning 132, 134
horizontal 132
spatial 132
temporal 132
vertical 133
pheromone 9
artificial 13, 17, 22
discretized 54, 56
matrix 22
trail 9
pheromone encoding
item-item 22, 88, 106
place-item 24, 26, 88, 106
place-item-item 89
pheromone matrix 93, 132
pheromone update
analytical examination 59
ATONCE 73
C-ACO 58
global 14, 19, 27
local 27, 57
negative 28, 95
PIPED 73
positive 28, 94
STATIONARY 73
STATIONARY delayed 74
PIE .. *see* pheromone encoding, place-item
PIIE ..... *see* pheromone encoding, place-item-item
pipelining, type of
piped ants 69
piped pheromone 69
population 28, 93
size 93
update 94
vector 96, 117
Population Module 98, 105
Population-based ACO 28, 93
algorithm 93
generic framework 106
mapping 97
potential
distance potential 25
flow potential 25, 85

Q-learning 27
QAP . *see* Quadratic Assignment Problem
Quadratic Assignment Problem 15, 21, 24
queue matrix 95, 96, 98, 105

random number 71, 92, 103, 118

rank
average rank 62
best rank 62
RDS . . . . . *see* decision sequence, random
reconfigurable computing 31
reconfigurable computing model 39
reconfigurable computing system . . . 33, 36
reconfigurable fabric 33, 35, 36
Reconfigurable Mesh 29, 40, 41
reconfiguration method 34
compile time reconfiguration 34
load time reconfiguration 34
runtime reconfiguration 34
reconfiguration model 43
multi-context reconfiguration 43
partial reconfiguration 32, 43
pipeline reconfiguration 44
single context reconfiguration 43
reinforcement 9, 27
resource requirements 108, 127
RMesh . . . . . *see* Reconfigurable Mesh
RMesh ACO 53
robustness 8
runtime 126
P-ACO 97, 110
RMesh ACO 55
standard ACO 19
TSH 118
runtime reconfiguration 34, 42, 131

S-ACO . . . . . *see* Simple Ant Colony Optimization
S-Array 101
S-Cell 101
S-FDS . *see* decision sequence, sorted fixed selection
bias 57, 85, 91
probability 13, 18, 22, 70, 97, 118
process 13, 18, 22, 71
set 22, 96, 117, 134
Selector 103
self-organization 8, 9
shift
cyclic 116, 121
period 116, 121
policy 116, 120
random 116, 121
Shortest Path Problem 12, 15
Simple Ant Colony Optimization 12
Single Machine Total Tardiness Problem 26, 104
size reduction 55, 141
SMTTP . . . . . *see* Single Machine Total Tardiness Problem
social insects 7
software core 50
speedup 113, 127
stigmergy 8, 9
stopping condition 19
swarm intelligence 9
swarm-intelligent systems 8

tabu-table 89
Time-Scattered Heuristic 114
Time-Scattered Heuristic, mapping . . . 118
transition rule
pseudo-random proportional . . 18, 118
random proportional 18
Traveling Salesperson Problem 15, 21
asymmetric 21
symmetric 21, 24, 58
TSH . . . . . *see* Time-Scattered Heuristic
TSP . . *see* Traveling Salesperson Problem

unit time delay model 42
update counter 58
update delay 81

virtual FPGA 39, 40
Virtual Wires 40