Ant Algorithms in Stochastic and Multi-Criteria Environments

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.-Inform. Michael Guntsch

Tag der mündlichen Prüfung:	13.01.2004
Referent:	Prof. Dr. H. Schmeck
Korreferent:	Prof. Dr. K.-H. Waldmann

2004 Karlsruhe

"These weren't ordinary ants. Centuries of magical leakage into the walls of the University had done strange things to them."
Terry Pratchett, *Sourcery*

Contents

Preface

Ant Algorithms are an iterative, probabilistic meta-heuristic for finding solutions to combinatorial optimization problems. They are based on the foraging mechanism employed by real ants attempting to find a short path from their nest to a food source. While foraging, the ants communicate indirectly via pheromone, which they use to mark their respective paths and which attract other ants. In the Ant Algorithm, artificial ants use virtual pheromone to update their path through the decision graph, i.e. the path that reflects which alternative an ant chose at certain points. The amount of pheromone an ant uses to update its path depends on how good the solution implied by the path is in comparison to those found by competing ants of the same iteration. Ants of later iterations use the pheromone markings of previous good ants as a means of orientation when constructing their own solutions, which ultimately results in the ants focusing on promising parts of the search space.

This thesis introduces new Ant Algorithms with superior performance for stochastic and multi-criterial optimization problems. The first part deals with adapting Ant Algorithms to probabilistic and dynamic optimization problems. The standard Ant Algorithm is not very well suited for these types of problems, since it does not take into account the possibility of a change in the problem instance. When dealing with probabilistic problems, a probability distribution over a number of structurally related instances is given, one of which will actually be realized after the optimization process has terminated. Since only the probability distribution and not the final, realized instance is known at runtime, the expected quality of a solution is used as the optimization criterion. The improvements to the regular Ant Algorithm result in a faster evaluation function, leaving more time for the construction of solutions, as well as in a state-of-the-art solution quality through better heuristic information available to the ants when building a solution.

In dynamic optimization problems, changes to the problem instance can occur while the optimization is still going on, giving the algorithm a chance to explic-

itly react to these changes. The goal of the optimization process in a dynamic environment is to achieve a good average solution quality over time. Hence, the algorithm must be able to refocus its search after a change in order to keep finding good solutions, even after many changes. The pheromone modifications strategies proposed for Ant Algorithms, which change the pheromone values in a characteristic fashion depending on the proximity of a change, allow for the algorithm to exploit previous information which is still accurate while at the same time exploring in places of severe change. This combination outperforms resetting the entire pheromone information uniformly.

Inspired in part by the methods used on dynamic problems, a new Ant Algorithm called Population-based Ant Colony Optimization (PACO) is proposed. PACO maintains a population of solutions instead of explicit pheromone values, rendering it faster and more versatile than the standard approach. The existence of a population also creates a potential interface for this Ant Algorithm with other, population-driven meta-heuristics. PACO is shown to offer performance on par with the best performing Ant Algorithms on static and dynamic problem instances, which it solves with the aid of a repair mechanism for old solutions.

In the last part of this thesis, the PACO algorithm is extended to be able to handle multi-criteria optimization, i.e. the concurrent optimization of several objectives which cannot be aggregated into a single evaluation function, with an arbitrary number of criteria. When more than one criterion exists, one solution dominates another when it is at least as good as the other in all criteria and better in at least one. The goal in multicriteria optimization is finding the set of solutions which themselves cannot be dominated, called the Pareto-optimal set. The PACO algorithm maintains the non-dominated solutions as a super-population from which a subset is chosen to generate the probability distributions for the individual criteria. Two methods for performing a weighted aggregation of the individual probability distributions are investigated, the better of which outperforms previous Ant Algorithms devised for bi-criterial optimization.

Acknowledgments

This thesis was written during the five years I spent working at the Institute for Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe. In January 2004, the thesis was submitted as a partial fulfillment for obtaining a doctoral degree from the Department of Economics

and Business Engineering at the University of Karlsruhe. I would like to express my gratitude to a number of people who helped to make this possible. First, I would like to thank my adviser, Prof. H. Schmeck, for his support of and interest in my work, and for giving me the time and freedom to follow my research interests. I am also indebted to Prof. M. Middendorf for the many insightful discussions on Ant Algorithms and research collaborations we had over the years.
I regard having worked at the AIFB as a privilege due to the wonderful atmosphere I shared with my colleagues, and I owe all of them thanks for their support and encouragement over the years, especially to the fellow members of the Efficient Algorithms group.
Finally, I would like to thank my family for their unconditional support and limitless patience, which were instrumental in helping me keep my sanity while finishing this thesis.

Karlsruhe, February 2004 M.G.

Chapter 1

Introduction

1.1 Combinatorial Optimization Problems

The field of combinatorial optimization problems is vast and omnipresent in the transportation, telecommunication, and producing industries. With the continuing increase in computational power, many situations which are encountered in real world scenarios can be simulated and solutions sought with specific algorithms engineered to work for many if not all instances of a given problem. Neglecting to optimize results in higher costs compared to competitors, which in turn ultimately leads to either adopting the strategy of optimization or economic ruin.

Many optimization problems exist for which well-known, efficient algorithms have been found. Some examples, requiring only an efficient method for the sorting of and searching in data, are finding the shortest path connecting two nodes in a graph [Dijkstra, 1959], the minimum weight spanning tree of a graph [Kruskal, 1956], or determining the convex hull to a number of points in the plane [Graham, 1972]. Other problems rely primarily on numerical principles, e.g. finding the greatest common divider of two numbers [Euclid, ca. -350]. And a large number of optimization problems, not falling into one of the earlier two categories, simply admit pleasingly concise methods for efficiently finding the optimal solution, e.g. maximizing Network Flow [Ford and Fulkerson, 1962].

However, there remains an entire class of combinatorial optimization problems which has so far eluded being solved optimally in an efficient, meaning relatively fast, manner. These problems either all admit an efficient deterministic algorithm for finding the optimal solution, or cannot be solved optimally in a time polynomially dependent on the size of the problem instance [Cook, 1971, Garey and Johnson, 1979]. This class of problems, known as *NP*, contains problems such as the Traveling Salesman Problem (TSP), where a salesman, knowing the least costly connection between all pairs of customers, wishes to visit each of his customers in such order that the cost of his entire tour, i.e. the sum of costs incurred by traveling from one customer to the next, is minimal (see [Lawler et al., 1985] for an overview). To date, it is unknown whether an algorithm exists that can solve this problem optimally in a number of steps polynomially bounded by the number of customers. However, the amount of research that has gone into the field of complexity theory and the implications of the existence of such an algorithm both make its existence seem unlikely.

Since finding the optimal solution takes too long, the obvious question to ask next is what level of quality can be found in an acceptable amount of time. This question has been the origin for many new fields in informatics. In approximation theory, problems are analyzed in order to ascertain whether they admit an algorithm that will guarantee coming within a certain relative distance from the optimal solution for all instances. Going a step further, in randomized approximation, this relative distance from the optimal solution is only guaranteed with a certain probability. While these fields have yielded approximation schemes for a number of problems in *NP*, among them the Euclidean TSP [Arora, 1998] and in general dense instances of some problems [Karpinski, 2001], the approximation gap is sometimes too large to be acceptable, being a function of the instance size for example, or the algorithm might have a runtime exponentially dependent on $1/\epsilon$ for getting within a relative distance of ϵ of the optimum, as is the case for the TSP.

Instead of attempting a theoretical analysis of a problem with the goal of determining how good a solution can be guaranteed by a sufficiently fast algorithm, a standard practice has been the application of heuristics. A heuristic method can best be thought of as a more or less complex rule of thumb, something that works well for many problem instances that are based on real world data or constructed randomly. An example for the TSP is the nearest-neighbor heuristic, which, starting at some customer, always chooses the closest customer from its current location as the next one to visit, until all customers have been visited. Heuristics like this are called one-shot heuristics, since they terminate after the construction of one solution. On average, such heuristics generally perform well, but they often allow for the construction of an instance where their behavior is arbitrarily bad, or at least worse than that of a different algorithm. It is possible to remedy this fact somewhat through the introduction of probabilistic behavior into the heuristic, e.g. for the TSP proceeding according to a probability distribution inversely proportional to the distance to the other cities, so that at least a chance exists for it to make the right choices, but, lacking any further guidance, it is unlikely that the heuristic can truly perform better, except by repetition and choosing the best found solution.

A better approach is for the heuristic method itself to determine where a promising portion of solution space lies for a given instance and to intensify the search for solutions in this area. Achieving this requires the heuristic to learn to direct its search, a feat that can be observed by certain processes in nature. Nature has since been a source of inspiration for developing meta-heuristics to deal with a variety of problems. These meta-heuristics are not bound to a single problem but rather represent a general mechanism or framework for solving a whole class of optimization problems.

One of the most direct mappings of the biological learning process to a computer algorithm is the field of Neural Networks (NNs), see [Haykin, 1999] for an introduction. Modeled to simulate neurons and synapses, these algorithms are most often employed for classification, prediction, and pattern matching. They have also seen some application in combinatorial optimization, starting with the proposition of the Hopfield network for the TSP in [Hopfield and Tank, 1985]. Though this work was not without controversy (due to the fact that many researchers could not reproduce the reported results), subsequent work in [Aiyer et al., 1990] and [Gee, 1993] now essentially guarantees the feasibility of using the Hopfield network for combinatorial optimization problems. Although being competitive in terms of solution quality with other meta-heuristics, one of the main challenges NNs face lies in the runtime necessary to achieve these results. The reason for the comparatively large runtime is the fact that the inherently parallel structure of a network must be simulated on a sequential machine. See [Smith et al., 1998, Smith, 1999] for an overview of the applications of NNs in combinatorial optimization.

One of the earliest devised meta-heuristics used for solving combinatorial optimization problems is Simulated Annealing (SA), which is based on the observation that the chemical composition of a metal slowly cooled is crystalline rather than amorphous as results from fast cooling. This is due to the fact that in a slowly cooled metal, the individual atoms have the time necessary to reach the position with the lowest possible energy level, which is achieved in a crystalline structure. The algorithmic exploitation of this phenomenon, first undertaken by [Metropolis et al., 1953], proceeds as follows. Given a starting solution and temperature, the solution is altered a little bit and reevaluated. If the new solution is better, it definitely becomes the starting point of the next perturbation. If the solution is worse, it replaces the old solution only with a certain probability correlated to the temperature, which slowly decreases over time. This has the effect that, at the beginning of the algorithm, nearly every new solution is adopted, while over time it becomes more and more likely that only better solutions are accepted, restricting the algorithm to hill-climbing. It has been shown in [Mitra et al., 1986] that, given a sufficiently slow cooling schedule, SA is guaranteed to converge to the optimal solution.

Another meta-heuristic inspired by nature which has been widely researched since its proposal by [Holland, 1975] is the Evolutionary Algorithm (EA), based on the classic principal of "survival of the fittest" [Darwin and Wallace, 1858] where in a gene pool the individuals corresponding to the fittest genes have better chances for passing their genetic information on to future generations. The desired result of this process is an individual combining the best traits of all its ancestors, or, taking the mutability of a gene into consideration, something even better. In the algorithm, a number of initial solutions are treated as genes, being crossed with one another, with a bias towards better solutions being selected for the cross-over, to create offspring which inherit characteristics from all involved parents. The population of solutions keeps evolving, hopefully towards the area of good solutions which contains the optimal solution. In keeping a multitude of solutions, EAs are capable of exploring several promising areas of the search space at once, if diversity and overall fitness of the population are well balanced. As for SA, theoretical results for certain types of EA show that it too converges towards the optimum, given enough time, see [Rudolph, 1997]. A detailed introduction to EAs is given in [Goldberg, 1989, Michalewicz, 1996].

A relatively new field in terms of its application to combinatorial optimization problems is Swarm Intelligence (SI). The term was first used in [Beni, 1988, Beni and Wang, 1989a,b] to describe the coordination of robots. Since then, two main interpretations of SI have been used for deriving methods to solve optimization problems. On the one hand, Particle Swarm Optimization (PSO), which is an evolutionary computation technique inspired by the social behavior of bird flocking or fish schooling, has been proposed [Kennedy et al., 2001]. In PSO, a population of solutions evolves by moving through the solution space, with good solutions acting as attractors. On the other hand, in [Bonabeau et al., 1999] SI is defined to include any scheme which employs the swarm mechanics derived from the behavior of social insects, particularly ants, for solving optimization problems. The concept of Ant Algorithms was

first introduced in [Dorigo, 1992] and since that time has been applied to both theoretical as well as practical optimization problems with great success. The performance exhibited by Ant Algorithms and the possibility of adaptation to new problems makes the study of this field a very worthwhile pursuit.
However, while Ant Algorithms have been adapted to deal with many different problems in combinatorial optimization, most of these problems were simple, "clean" problems with a single optimization goal. In practice, this is quite often not the case. Instead of having absolute certainty about all parameters of a problem instance, there might be a margin for error which could and should be considered during the optimization. Other times, an instance might be dynamic in nature, changing over time and requiring the algorithm to keep track of the occurring modifications and continually reoptimize in order to be able to present a valid, good solution at all times. Finally, in many real world problems, it is not sufficient to optimize only a single objective while ignoring the effect of the solution on all other objectives. Rather, a multitude of solutions is sought, each presenting a different trade-off between the objectives that need to be optimized, leaving the ultimate decision which solution to realize up to a (human) decision maker. Up to now, little or no work has been done in these fields with Ant Algorithms.

1.2 Scope

In this thesis, we present novel methods for applying the principles of Ant Algorithms to stochastic and multi-criteria optimization problems, which in many cases significantly outperform previous algorithms for these problems. Using an evaporation based Ant Algorithm, we concentrate on improved heuristic guidance and pheromone diversification techniques for the stochastic problem classes. We also devise an entirely new class of Ant Algorithm called Population-based Ant Colony Optimization (PACO), which maintains a population of solutions instead of explicit pheromone values, rendering it faster and more versatile than the standard approach. PACO is shown to offer good performance on static and dynamic problem instances, and to lead to superior results when concurrently optimizing multiple objectives.
Parts of this thesis are based on previous work published in [Guntsch et al., 2000, Guntsch and Middendorf, 2000, Guntsch et al., 2001, Guntsch and Middendorf, 2002a,b, Branke and Guntsch, 2003, Guntsch and Middendorf, 2003]. In detail, the structure is as follows:

- In Chapter 2, we introduce the concept of Ant Algorithms. We describe the probabilistic decision rule employed by virtual ants constructing a solution for the instance of a given combinatorial optimization problem. Different methods for updating the pheromone information are discussed. Afterwards, the ACO meta-heuristic is explained, and we define the variant of ACO we predominately use for the remainder of this thesis. Finally, a small survey of ACO applications is given.

- Chapter 3 deals with Probabilistic Problems, specifically the Probabilistic Traveling Salesman Problem (PTSP), where some amount of uncertainty about the final structure of an instance exists and thus the expected solution quality must be optimized. We introduce an approximation scheme for the computationally expensive evaluation function, which significantly reduces the runtime while achieving virtually identical results. Also, two new guidance schemes for Ant Algorithms are proposed which enhance performance to a degree that lets the ACO meta-heuristic outperform the previously best method for this problem.

- Dynamic optimization problems, specifically the Dynamic Traveling Salesman Problem (DTSP), which is in some ways complementary to the PTSP, are defined in Chapter 4. Here changes occur to the problem instance during the run of the algorithm. We introduce a number of strategies to help the ACO meta-heuristic integrate the changes to an instance, keeping the balance between exploiting valid old information and enforcing exploration in the updated areas.

- A new Ant Algorithm based on a different approach to managing the pheromone values in the ACO meta-heuristic, called Population-based Ant Colony Optimization (PACO), is explored in Chapter 5. Here, we propose to manage a population of solutions rather than maintain an explicit pheromone matrix. This update scheme is partially inspired by one of the strategies for the DTSP from Chapter 4, and we consequently conduct our tests of this modification on static as well as dynamic optimization problems.

- Chapter 6 deals with Multi-criteria Optimization (MCO) based on extensions of the Ant Algorithm defined in Chapter 5. These extensions enable the PACO algorithm to solve MCO problem instances with an arbitrary number of objectives by aggregating the probabilitiy distributions corresponding to the single criteria composing the multi-criterial instance.

We evaluate our algorithm on 2- and 4-dimensional instances of the Single Machine Total Tardiness with Setup Costs Problem (SMTTSCP), and show that it outperforms a previous Ant Algorithm for bi-criterial problems.

- We conclude our thesis with a summary and outlook on what we see as the most promising areas of future work in Chapter 7.

Chapter 2

Ant Algorithms

2.1 Biological Foundations

Although only 2% of all insect species living in nature are social, they comprise more than 50% of the total insect biomass globally [Arnett, 1985], and more than 75% in some areas like the Amazon rain forest [Fittkau and Klinge, 1973]. By social, we mean that these insects, including all ants and termites and some subspecies of bees and wasps, live in colonies composed of many interacting individuals. Insect colonies are capable of solving a number of optimization problems that none of the individual insects would be able to solve by itself. Some examples are finding short paths when foraging for food, task allocation when assigning labor to workers, and clustering when organizing brood chambers, all of which are problems that have counterparts in real world applications. Insect colonies have more than 100 million years of experience in solving these problems, perhaps giving them an evolutionary advantage over human colonies, which have only existed for about 50000 years.

In order for a swarm of insects to cooperate in problem solving, some form of communication is necessary. This communication between the individuals of a colony can be more or less direct, depending on the exact species. When a social bee has found a food source, it communicates the direction and distance of the location where it found the food to the other bees by performing a characteristic dance [von Frisch, 1967]. This is a very direct communication, as the other bees must perceive the dance the one bee is performing in order to be able to locate the food source. Other forms of direct communication include stimulation by physical contact or the exchange of food or liquid.

Indirect communication between the individuals of a colony is more subtle and requires one individual to modify the environment in such a way that it will alter the behavior of individuals passing through this modified environment at a later time. One scenario where this type of environmentally induced action exists in nature is when termites construct a nest, which has a very complicated structure and exhibits properties like climate control [Lüscher, 1961]. Whenever a construction phase has ended, the surroundings of the worker have changed, and the next phase of working is encouraged, which in turn results in new surroundings, and so forth. Another example for indirect communication is the laying of pheromone trails performed by certain species of ants. An ant foraging for food will mark its path by distributing an amount of pheromone on the trail it is taking, encouraging (but not forcing) ants who are also foraging for food to follow its path. The principal of modifying the environment in order to induce a change in behavior as a means of communication is called stigmergy and was first proposed by [Grassé, 1959].

Stigmergy is the basis for the organization in many ant colonies. Although the ants have a queen, this is a specialized ant which is only responsible for laying eggs and does not have any governing function. Instead, the ants of a colony are self-organized. The term self-organization (SO) is used to describe the complex behavior which emerges from the interaction of comparatively simple agents. Its origins lie in the fields of physics and chemistry, where SO is used to describe microscopic operations resulting in macroscopic patterns, see [Nicolis and Prigogine, 1977]. Through SO, the ants are able to solve the complex problems which they encounter on a daily basis. The benefits of SO as a basis for problem solving are especially apparent in its distributed and robust character. Effectively, an ant colony can maintain a meaningful behavior even if a large number of ants are incapable of contributing for some amount of time.

To better understand the mechanism behind an ant colonies' ability to converge to good solutions when looking for a short path from the nest to a food source, some experiments were undertaken by [Goss et al., 1989, Deneubourg et al., 1990]. In [Deneubourg et al., 1990], a nest of the Argentine ant *Linepithema humile* was given two paths of identical length it could take to reach a food source, and after some time had passed, it was observed that the ants had converged to one of the paths, following it practically to the exclusion of the alternative. To test whether this type of ant would converge to the shortest of a number of paths, a double bridge experiment was performed in [Goss et al., 1989] where the ants could choose twice between a short and a long path, see Figure 2.1 for the exact setup.

The Argentine ant is practically blind, so it has no means of identifying a

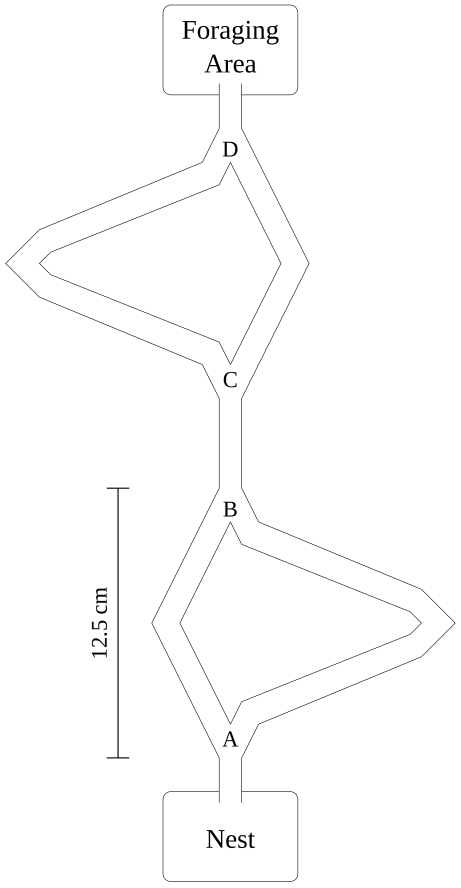


Figure 2.1: Double Bridge experimental setup, from [Beckers et al., 1992].

short path directly. However, despite this deficiency, a swarm of these ants is capable of finding the shortest path connecting the nest to the foraging area containing the food, as the experiment shows. Initially, all ants are located at the nest site. A number of ants start out from the nest in search of food, each ant laying pheromone on its path, and reach the first fork at point A. Since the ants have no information which way to go, i.e. no ant has walked before them and left a pheromone trail, each ant will choose to go either right or left with equal probability. As a consequence, about one half of the foraging ants will take the shorter route, the rest the longer route to intersection B. The ants which were on the shorter track will reach this intersection first, and have to decide which way to turn. Again, there is no information for the ants to use as orientation, so half of the ants reaching intersection B will turn back toward the nest, while the rest continues toward the foraging area containing the food.

The ants on the longer branch between intersections A and B, unaffected by the other ants they met head-on, arrive at intersection B and will also split up; however, since the intensity of the pheromone trail heading back toward the nest is roughly twice as high as that of the pheromone trail heading for the foraging area, the majority will turn back toward the nest, arriving there at the same time the other ants which took the long way back arrive. Interestingly, since more ants have now walked on the short branch between intersections A and B in comparison to the long one, future ants leaving the nest will now already be more inclined to take the short branch, which is a first success in the search for the shortest path.

The behavior of the ants on the second bridge at intersections C and D and in between is virtually identical to the behavior exhibited before on the first bridge between intersections A and B. Ultimately, a number of ants will reach the foraging area and pick up some food to carry back to the nest. Arriving at intersection D, the ants will prefer the short branch by the same argument as used above for ants starting out fresh from the nest, and the same holds again for intersection B. Since the amount of pheromone at intersections A and C on the path back to the nest is (roughly) equal to the sum of the pheromone amounts on the two branches leading away from the nest, the shortest complete path from the foraging area back to the nest is also the most likely to be chosen by the returning ants. Since the ants are continually distributing pheromone as they walk, the short path is continually reinforced by more and more ants, until the amount of pheromone placed thereon in relation to the alternate routes is so high that practically all ants use the shortest path, i.e. the system converges to the shortest path through self-reinforcement.

One point that we have neglected to mention so far is that the pheromone used by ants to mark their trails slowly evaporate over time. This does not render the arguments used to explain the double bridge experiment any less valid, it simply makes some of the math used for explanation less rigorous than implied. Indeed, due to the evaporation of pheromones, path segments that have not been chosen for some time, invariably long ones, will contain almost no traces of pheromone after a sufficient amount of time, further increasing the likelihood of ants taking the short path segments identified by the continually updated pheromone. Note that the amount of pheromone on the shortest path has an inherent maximum value, which is reached when the amount of pheromone deposited by passing ants and evaporated in a given time interval are the same.

In the rest of this chapter, we explain how the concept of marking paths with pheromones can be exploited to construct algorithms capable of solving highly complex combinatorial optimization problems.

2.2 Ant Decision Rule

In Section 2.1, we have given an example of how real ants solve optimization problems in nature. In order to transpose the employed mechanism of stigmergy to an algorithmic principle, some abstractions from and modifications of the natural behavior of ants are necessary. These were first proposed in [Dorigo, 1992].

Consider the problem of finding the shortest path connecting two points $a, b \in V$ in a symmetric, weighted graph $G = (V, E, w)$ with $w : E \to \mathbb{R}^+$ assigning each edge a positive weight. This is a generalization of the problem solved by the real ants looking for the shortest path to the food source. An artificial ant used to construct a solution for this problem would start at one of the two designated nodes and progressively choose which node to visit next, i.e. which edge to traverse, from its current position until it has reached the target node. While the ant is searching, or after it has reached its goal, depending on the implementation, it marks its trail with artificial pheromone, which influences the decisions made by future ants building a path through the graph. In the following, we introduce the process of how artificial ants build solutions to combinatorial optimization problems in general, and how they mark these solutions with pheromone for the benefit of future ants.

One of the first drawbacks that becomes apparent when analyzing the behavior of real ants in the experiments mentioned in Section 2.1 is the possibility of an ant's path including circles, or of an ant returning to the nest without having collected any food. In the shortest path problem, this would correspond to an ant visiting a node multiple times, which, under the assumption that all weights are positive, cannot be an optimal solution. Since the goal of using artificial ants to build solutions is to find the best possible solution, it makes sense to restrict the possibilities an artificial ant has when choosing which node to visit next. To achieve this, we let each artificial ant maintain a selection set S which at any time contains the alternate choices an ant may currently make. The size and composition of S depend largely on the problem for which the ant must construct a solution. For the shortest path problem, S could denote all nodes reachable from the current node, except for those nodes that have already been visited. Note that, depending on the exact structure of the underlying graph G, this could lead to the ant being left with an empty selection set $S = \emptyset$, which would then require some exception handling. For most permutation problems, e.g. the Traveling Salesman Problem (TSP), the Quadratic Assignment Problem (QAP) (see [Koopmans and Beckman, 1957] for a detailed definition), and all scheduling problems, S denotes customers not visited, facilities not assigned, and jobs not scheduled, respectively. In all these cases, S initially contains all possibilities, and is updated each time the virtual ant makes a decision, resulting in an empty set when the construction process is completed. Using the set S, it is also possible to model constraints which a feasible solution must meet. For example, in the Sequential Ordering Problem (SOP), which essentially is a TSP with precedence constraints on the the order in which the customers can be visited, S can be used to eliminate all customers which can not be visited yet from the selection process, and make them available once the respective precedence constraints are satisfied.

Using the selection set S, we guide an artificial ant in its search process for a feasible solution. In order for the ant to find a good solution and maintain the analogy to the foraging behavior of real ants, artificial pheromones are employed. Artificial pheromones are real valued numbers $\tau \in \mathbb{R}$ assigned to the alternatives an artificial ant has when constructing a solution. For example, in the shortest path problem, a pheromone value τ_{ij} is assigned to each edge (i, j), indicating how beneficial it is to traverse the corresponding edge. Formally, an ant located at node i chooses the next node to visit according to the *random proportional transition rule* defined by

$$p_{ij} = \frac{\tau_{ij}}{\sum\limits_{h \in S} \tau_{ih}} \tag{2.1}$$

The p_{ij} values define a probability distribution according to which the ant makes its decision. As the formula shows, larger pheromone values correspond to a higher probability for the implied choice being made, which is modeled to simulate real ants choosing denser pheromone trails with higher probability.

The pheromone values for the shortest path problem are assigned to the edges, which allows for an intuitive representation of all pheromone values in a pheromone matrix encoded in a node×node fashion, i.e. τ_{ij} gives information about the attractiveness of going to node j when located at node i. This method is also used for many other problems, e.g. TSP, SOP, and scheduling problems with setup costs, where the solution quality is derived from a predecessor-successor relationship. However, for problems like QAP and scheduling problems with due dates, where the (immediate) predecessor or successor of a node in the solution are irrelevant, a node×place encoding is employed. As a detailed example, in QAP, facilities are allocated to sites, the goal being to minimize the sum of distance-weighted flows between the facilities, see [Koopmans and Beckman, 1957]. Planning facility j after facility i has, by itself, an indiscernible influence on solution quality, which is why no pheromone information to this point is meaningful. Rather, the pheromone values τ_{ij} pertain to placing facility j at site i, which, due to the location of site i and the known distances to the other sites, gives some information about expected solution quality, especially in combination with other (i, j) allocations.

Not all problems allow for the pheromone values to be arranged sensibly in a matrix fashion. In [Michel and Middendorf, 1999], Ant Algorithms are used for constructing a shortest super-sequence to a given number of strings, i.e. the shortest string which contains all given strings as a sequence. The characters of each string are marked with pheromone information to increase the likelihood of being chosen as the next character when the ants construct a supersequence. Marking individual nodes (or items) with pheromone values is proposed by [Leguizamón and Michalewicz, 1999] when using Ant Algorithms to solve subset problems like the Multiple Knapsack Problem (MKP) , where the optimal subset (in the form of a binary vector stating which items are in the subset) to a given number of items is sought which satisfies a number of linear constraints and maximizes a linear function. The higher the pheromone value assigned to an item is, the more likely its inclusion in the knapsack becomes.

In addition to the pheromone information, many problems allow for some form of heuristic guidance to be employed by the ants for constructing better solutions. This heuristic information is available on the same level as the pheromone information, i.e. for making local decisions. When an ant is located at node i in the decision graph, we denote the perceived benefit of moving to node j by η_{ij}. Used in combination with the pheromone information, the formula for the *random proportional transition rule* including heuristic information is an extension of Formula 2.1:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum\limits_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}} \tag{2.2}$$

As for the pheromone values, a larger heuristic value corresponds to a higher probability of the implied choice being made. The exact form of the heuristic information depends upon the problem class at hand. For the shortest path problem or TSP, a standard practice is to set $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the weight of edge (i, j) or the distance from customer i to customer j respectively, see [Bonabeau et al., 1999]. A small value for d_{ij} implies an immediate benefit if the corresponding edge is traversed, since the path/tour-length is only increased by a small amount. Note that for these two problems the heuristic information is constant, which is not always the case. In Chapter 3, we introduce heuristic information which changes dynamically depending on previous choices made by the ant in the construction process. Other cases where the heuristic information is dynamic are some scheduling problems, e.g. the Single Machine Total Weighted Tardiness Problem (SMTWTP), where jobs with due dates must be scheduled on a single machine, and the heuristic information can be chosen according to the difference between the continually increasing earliest finishing time and the due date of a job, see [Crauwels et al., 1998] for an overview. For some problem classes, the use of heuristic information does not improve the performance of the Ant Algorithm by a significant margin, e.g. for the QAP [Stützle and Hoos, 2000]. There are even some cases, e.g. the Maximum Clique Problem (MCP) , where for a graph the size of the largest completely connected subgraph is sought, studied in [Fenet and Solnon, 2003], where the use of heuristic information ultimately leads to a poorer solution quality.

Two new parameters in Formula 2.2 are α and β, which are used to adjust the weight of the pheromone and the heuristic information respectively. For most applications, setting $\alpha = \beta = 1$ is sufficient. However, tuning these parameters can lead to better performance for some problem classes. For the TSP, choosing $\beta > 1$ has been shown to yield good results, e.g. using $\beta = 2$ in [Dorigo and Gambardella, 1997b, Stützle and Hoos, 2000] or $\beta = 5$ in [Bullnheimer et al., 1999]. Using a steadily decreasing value of β has also been applied successfully in [Merkle et al., 2002]. Using values of $\alpha > 1$ has been studied in [Fenet and Solnon, 2003] to achieve quicker convergence at the cost of a lower solution quality in the long run.

In [Dorigo and Gambardella, 1997b,a], a modification to the random proportional transition rule is proposed which allows for greater control over the balance between exploration and exploitation. A parameter $q_0 \in [0, 1)$ denotes the probability for choosing the best combination of pheromone and heuristic information perceived by the ant, i.e. for choosing j with

$$j = \arg\max_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta} \tag{2.3}$$

instead of proceeding probabilistically according to Equation 2.2, which is done with probability $(1-q_0)$. This *pseudo-random proportional rule* allows for a calibration of the amount of exploration performed by the ants, similar to the temperature setting in Simulated Annealing, but usually kept constant during the run.

The construction process performed by an artificial ant is iterative in nature, as has already been hinted at above. Going back to the shortest path problem, the ant starts at one of the two points to be connected, chosen randomly with equal probability. Afterwards, the ant continually chooses which node to visit next, until it has arrived at the target node, the sequence of the edges/nodes it traversed from the initial node denoting its solution. Note that we are examining only one ant on its way to constructing a solution, and not a swarm. Although some investigations have been made on running the entire swarm of ants as a simulation in [Dorigo and Gambardella, 1997a], it has proven to be beneficial to run the ants sequentially for some variants, while for others it makes no difference. This has to do with the mechanism used for updating the pheromone values, which we cover in the following section.

2.3 Pheromone Update

The purpose of the pheromone update is to focus the search process of the ants on a promising portion of the solution space, which is then explored more extensively in the hope of finding the optimal solution. The field of update strategies is perhaps the most studied part of Ant Algorithms, and many different varieties of updating the pheromone values have been proposed. In this section, we explain the conceptually different approaches which have been taken to update pheromone values, and also explain in detail some of the more successful strategies.

At the most abstract level, the pheromone update should accomplish a positive reinforcement of those search space regions which seem promising and a negative reinforcement of all others. The principal mechanisms used for this are pheromone evaporation (for the negative reinforcement), which diminishes all pheromone values by a relative amount each time it is applied, and pheromone intensification (for the positive reinforcement), achieved by adding an update term to selected pheromone values. Formally, an update takes the form

$$\forall i, j \in [1, n] : \tau_{ij} \mapsto (1-\rho) \cdot \tau_{ij} + \Delta_{ij} \tag{2.4}$$

where $\rho \in (0, 1]$ is a parameter of the algorithm denoting how much of the pheromone information is lost with every application of evaporation, and Δ_{ij} is an update value, which is 0 if the edge (i, j) was not traversed by the ant. The exact value of Δ_{ij} and especially the strategy when an update is performed is the key difference between most types of Ant Algorithm.

Generally, updates to the pheromone values take place after an iteration of m ants has constructed solutions. In the Ant System (AS), which was introduced in [Dorigo, 1992] for solving the TSP, an individual update value $\Delta_{ij}(l)$ for each ant $l \in [1, m]$ is calculated, and the update is performed with the sum of update values $\Delta_{ij} = \sum_{l=1}^{m} \Delta_{ij}(l)$. Three different methods for determining the individual Δ_{ij} were tested: assigning a constant, inverse to the distance d_{ij} between customers i and j, and, performing best and used subsequently, inverse to the length of the entire tour, i.e. the solution quality. In addition to the m ants of an iteration being allowed to perform an update, it was also proposed to let a number of so called elitist ants, which represent the best solution found by all ants so far, update the pheromone trail. Using a small number of these elitist ants, inspired by the elitist strategy in [Holland, 1975], intensifies the search near the currently best solution, leading to better results overall. Further research resulted in the introduction of the Ant Colony System (ACS) in [Dorigo and Gambardella, 1997a,b]. Here, an online update of the pheromone values was proposed in order to enforce exploration: each time an ant traversed an edge (i, j), it would reduce the corresponding pheromone value according to

$$\tau_{ij} \mapsto (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{2.5}$$

with τ_0 being the initial pheromone level on all edges, thus encouraging the following ants to choose different edges (note that this holds only for $\tau_{ij} \geq \tau_0$, i.e. if τ_{ij} has received an update in the not too distant past). Also, the global update by all ants at the end of an iteration was replaced by one update performed along the best tour found so far, i.e. by one elitist ant.

Originating from AS and ACS, many other update schemes have been proposed. In [Stützle and Hoos, 1997, 2000], the *MAX-MIN* Ant System (MMAS) is introduced, which avoids stagnation in the search process by limiting pheromone values to the predetermined interval $[\tau_{min}, \tau_{max}]$. Limiting the pheromone values also bounds the minimum and maximum probability with which an edge is selected according to Equations 2.1 and 2.2, if the heuristic values are bounded as well. Two further characteristics of MMAS are that only the best (either globally or of the iteration) ant updates, and that all pheromone trails are initialized to τ_{max}, leading to a longer initial search phase in which the ants are guided almost exclusively by the heuristic values, and therefore a greater likelihood of not converging on the first good solution found. In [Bullnheimer et al., 1999], a modification the AS called AS-rank is proposed, where the m ants of an iteration are ranked by their solution quality and, together with a number of elitist ants of maximum rank, update the pheromone trail in proportion to their rank.

Some methods also exist that operate without using evaporation. In [Maniezzo, 1999], pheromone update is accomplished by comparing the solution quality of the ant to the average quality of the m previous ants. If it is better, a positive update is performed along the path; if it is worse, the update is negative. Thus, the update is accomplished in $O(m \cdot n)$ time for m ants, compared to $O(n^2)$ for Equation 2.4. In [Guntsch and Middendorf, 2002b], a population of solutions is maintained from which the pheromone matrix is derived. Updates are performed on the population, with the insertion of a solution being equivalent to a positive update of the pheromone matrix, and a deletion to a negative update (corresponding to the previous positive update which was undertaken upon insertion). This update mechanism will be studied in more detail in Chapter 5.

2.4 Ant Colony Optimization

The Ant Colony Optimization (ACO) Meta-Heuristic is defined in [Dorigo and Caro, 1999] as a common framework for Ant Algorithms in general. The main points composing an Ant Algorithm are the capability of the ants to construct a feasible solution, positive and negative reinforcement, and daemon actions of which individual ants are not capable, like local search or finding the best ant of an iteration. In this section, we explain the design decisions made for the ACO algorithm used in this work. A high level overview is given in Algorithm 2.1.

At the beginning of the algorithm, the pheromone matrix is initialized. The level of the initial pheromone values, as well as which values are initialized, depend on the problem being considered. For node$\times$node problems, where the sequence of the nodes is optimized, it is customary to initialize the values $\tau_{ii} = 0$ to emphasize that node i cannot follow itself. This problem typically resolves itself, however, as visiting node i entails a removal of i from S, meaning that it can no longer be chosen when the ant is located at node i.

The selection set S is initialized to all possible choices $\{0, \ldots, n-1\}$ (we use this representation instead of $\{1, \ldots, n\}$ to facilitate modulus calculations in later

Algorithm 2.1 basic, high level ACO algorithm

```
1: initialize pheromone values $\tau_{ij} \mapsto \tau_0$
2: repeat
3:   for each ant $i \in \{1, \dots, m\}$ do
4:     initialize selection set $S \mapsto \{0, 1, \dots, n-1\}$
5:     let ant $i$ construct solution $\pi_i$
6:   end for
7:   for all (i,j) do
8:     $\tau_{ij} \mapsto (1-\rho) \cdot \tau_{ij}$
9:   end for
10:  determine best solution of iteration $\pi^+$
11:  if $\pi^+$ better than current best $\pi^*$ then
12:    set $\pi^* = \pi^+$
13:  end if
14:  for all $(i,j) \in \pi^+$ do
15:    $\tau_{ij} \mapsto \tau_{ij} + \rho/2$
16:  end for
17:  for all $(i,j) \in \pi^*$ do
18:    $\tau_{ij} \mapsto \tau_{ij} + \rho/2$
19:  end for
20: until condition for termination met
```

chapters). Depending on the problem, e.g. TSP, it is possible to reduce the size of S to a so called candidate list of especially good choices, and let the ant choose only from this list, see [Gambardella and Dorigo, 1996]. The benefit of this operation is to significantly reduce the runtime of the algorithm for large instances. However, since this candidate list introduces another parameter that needs tuning, the impact on ultimate solution quality is unclear, and other problems where such a candidate list is less opportune are also studied in this work, we have decided to use the straightforward implementation of the selection set S.

In [Dorigo and Gambardella, 1997b], a method for finding the optimal number of ants is discussed which relies on knowledge of the average size of the pheromone values before and after a change, both being a function of the problem size n. Modeling the local pheromone update as a first-order linear recurrence relation enables m to be expressed as a function of the average pheromone levels before and after an update, and of the initial pheromone values. Although the authors cannot provide these pheromone levels, they argue that experimental results show $m = 10$ to work well, and this is also the case in our own experiments.

The process of constructing a solution itself is intentionally left vague in Algorithm 2.1, since this procedure differs for the individual problems considered. A more detailed pseudo-code fragment is provided in the following chapters for each of the problems studied.

The pheromone update is global and consists of two steps: evaporation of all pheromone values, and positive updates along the paths found by the best ant overall and the best of the m ants of the iteration. Using these two ants for the positive update in an alternating fashion has been studied in [Stützle and Hoos, 1997]. Our own experiments have shown that using more ants for updating the pheromone matrix does not perform significantly different compared to using only the two mentioned above. Note that for the positive update, we write $(i, j) \in \pi$, which is defined more accurately for the individual problems later.

As a condition for termination, the default choice is setting a maximum number of iterations for the algorithm. Other possibilities include using real CPU time, convergence of the pheromone matrix to a single solution, or, since true convergence takes excessively long, number of iterations since last improvement. We choose using a constant number of iterations, since this allows for keeping the same parameter settings, especially rate of evaporation, for different problem instances. Using real CPU time is unreliable due to the background noise of other processes running on the same machine, and conceivably requires some analysis for each instance in order to set a plausible evaporation rate. As already stated, true convergence takes very long (infinitely long if the pheromone values were of arbitrary precision), ruling out this possibility. Lastly, the number of iterations since the last improvement is, once again, a parameter of the algorithm that might need tuning in conjunction with the evaporation rate. We feel that using a maximum number of iterations is the best method since it has no significant disadvantages and allows for easy parameter setting and later comparison of empirical results.

2.5 Applications

In this section, we present a survey of some of the noteworthy applications of ACO. Of course, this survey cannot hope to present a complete overview, and the interested reader is referred to [Corne et al., 1999, Bonabeau et al., 1999, Stützle and Dorigo, 2002].

One of the earliest and most intuitive applications of ACO was the TSP [Dorigo, 1992]. Since all ACO algorithms depend in some fashion on the metaphor of an ant moving through a graph [Dorigo and Caro, 1999], using the TSP to illustrate the basic principles of Ant Algorithms is a logical choice, and it is also used as the introductory example in [Bonabeau et al., 1999]. ACO has delivered good results on many TSP instances, especially when combined with local search [Stützle and Hoos, 2000]. However, due to the existence of very good heuristics like Lin-Kernighan [Lin and Kernighan, 1973] and polynomial time approximation schemes [Arora, 1998] for the Euclidean TSP, ACO algorithms are not the best choice for this problem class. The situation is better for the Sequential Ordering Problem (SOP), an extension of the TSP, where the goal is to find a minimum weight Hamiltonian Path with precedence constraints among the nodes. Here, a form of Ant Algorithm called Hybrid Ant System (HAS-ACO) [Gambardella and Dorigo, 2000] is currently one of the best algorithms available. Other variations of the standard TSP, like the Probabilistic TSP and Dynamic TSP, are also handled well by ACO, which we show in Chapter 3 and Chapter 4 respectively.

Another problem related to the TSP is the Vehicle Routing Problem (VRP), in which a number of customers must be serviced exactly once, and all vehicles begin and end their respective tours at a depot. The goal is to minimize the number of vehicles while meeting constraints such as capacity per vehicle, maximum tourlength per vehicle, and time windows. Solving this problem with Ant Systems was first proposed in [Bullnheimer et al., 1997], and further research has lead to a unified approach for VRPs [Reimann et al., 2003], where the Ant System is combined with an Insertion algorithm from [Solomon, 1987].

The QAP, defined in [Koopmans and Beckman, 1957] and shown to be NP-hard in [Sahni and Gonzales, 1976], is a conceptually different optimization problem compared to the TSP and its derivates in the sense that the pheromone matrix is not interpreted in a node$\times$node fashion, but rather as node$\times$place. Applying Ant System to the QAP was first undertaken by [Maniezzo and Colorni, 1998], including a heuristic guidance scheme for the ants when constructing a solution. Adding local search to the AS algorithm was shown to be beneficial in [Stützle and Hoos, 1998]. In [Gambardella et al., 1999a], the Hybrid Ant System (HAS) was introduced and applied to the QAP with good results. The HAS algorithm uses ants to modify solutions instead of building them, and the pheromone values are used to remember beneficial changes.

Another class of problems in which Ant Algorithms have seen wide and successful application is in scheduling problems. For scheduling with due dates, e.g. the Single Machine Total Weighted Tardiness Problem (SMTWTP), the pheromone matrix is also interpreted in a node$\times$place fashion. However, in contrast to the QAP, "place" in this case refers to the place in the schedule and not a physical location. An ACO algorithm for the SMTWTP was applied by [Bauer et al., 1999], where ACO found the optimal solution to 125 benchmark problems more often than the other heuristics evaluated. Ant Algorithms have also been applied to somewhat more complex scheduling problems, e.g. job shop scheduling [Colorni et al., 1994], flow shop scheduling [Stützle, 1998], and, most notably, the Resource Constrained Project Scheduling Problem (RCPSP) [Merkle et al., 2002], where ACO was state of the art at the time of publishing.

So far, all the problems discussed have been permutation problems, which can be handled quite well by ACO. However, some efforts have been undertaken to apply ACO to areas where solutions are not permutations. As mentioned above, in [Michel and Middendorf, 1999], ACO is successfully applied to the shortest supersequence problem. Also, some partitioning problems, e.g. graph coloring [Vesel and Zerovnik, 2000] and data clustering [Monmarche, 1999], have been solved with ACO, with varying degrees of success. In [Solnon, 2002], ACO is used as a generic algorithm for solving Constraint Satisfaction Problems (CSP) with promising results.

As a final note, although not being an application, in the recent past it has been shown that under certain conditions, some versions of ACO can provably find the optimal solution to the instance of a problem with a probability arbitrarily close to 1 [Gutjahr, 2002, Stützle and Dorigo, 2002]. Although these results have no immediate impact on the applicability of ACO algorithms, they put ACO on the same level as Simulated Annealing or Genetic Algorithms in terms of solution finding capability. Note that with a lower bound greater than 0 on the probability to find the solution or move closer to the solution in a given iteration, any method will find the optimum with a probability arbitrarily close to 1 given enough time. Since the number of steps required for this high probability is never less than exponential, it does not make any of the mentioned meta-heuristics more attractive than enumeration.

Chapter 3

Probabilistic Problems

3.1 Introduction to Probabilistic Combinatorial Optimization Problems

An instance of a Probabilistic COP is an extension of a normal (static) COP instance, consisting of the static instance and a probability distribution over the elements of the instance which indicate how probable it is that each respective element will actually be contained in the final instance to which the solution is applied. A well known example is the Probabilistic TSP introduced by [Jaillet, 1985], where each customer needs to be visited only with a certain probability. This probability is known in advance, and the goal is to find an *a priori* solution with minimum expected length. Since the final instance is known only after the optimization process has finished, the complete tour through all customers is applied to the final instance by skipping customers in the tour that are no longer in the instance. In this chapter, we give a thorough introduction to the PTSP and show how the ACO algorithm can be modified to offer state-of-the-art performance for this problem class.
There are several reasons that motivate the study of PCOPs. On an abstract level, it can serve to identify the robustness of the solutions found by an algorithm to the static problems. For the concrete case of PTSP, realistic scenarios exist in which a driver delivers goods to customers with probabilistic demand and company policy dictates that the same driver should visit the same customers when they require a visit. Also, reconstructing a good tour on short notice might be too expensive computationally, or communicating the changes to a tour too costly or confusing for a driver. Using an *a priori* strategy where each driver is responsible for skipping the customers not requiring a visit solves these problems.

In [Jaillet, 1985], the PTSP is introduced as a nonlinear integer programming model, which is reformulated as a mixed integer and ultimately as a linear integer program solved via a branch and bound algorithm. Dynamic Programming as well as local search heuristics inspired by tour construction and tour improvement techniques for the static TSP are evaluated. These, however, do not perform very well, as is shown in [Jaillet, 1985] for Dynamic Programming and in [Jezequel, 1985] for some of the other heuristics proposed. The adaptation of TSP heuristics for the PTSP is also studied in [Rossi and Gavioli, 1987]. More comparisons of heuristics as well as upper and lower bounds on solution quality depending on the composition of the PTSP instance are analyzed in [Bertsimas and Howell, 1993]. Further bounds for the expected tourlength in Euclidean spaces can be found in [Jaillet, 1993]. In [Laporte et al., 1994], instances with up to 50 customers are solved optimally using a branch-and-cut approach.

Since the ACO metaheuristic is known to perform well for the static TSP [Dorigo and Gambardella, 1997b], it is intuitive to apply it to the PTSP as well. [Bianchi et al., 2002a] study the use of the faster TSP evaluation function instead of the PTSP evaluation function to measure solution quality (see Section 3.2 for more details) in order to have more time to construct solutions. This method is only beneficial, however, if the probability values for inclusion of the customers in the instance are close to 1. In [Bianchi et al., 2002b], more instances are studied and a comparison of ACO to random search and radial sort is performed. The first real modifications to ACO that help in finding better solutions are proposed in [Branke and Guntsch, 2003] and form the basis for this chapter. However, the heuristics we present are an improvement in comparison to those first employed in [Branke and Guntsch, 2003] both in terms of the computational effort necessary for their calculation as well as their applicability to inhomogeneous probability distributions.

In Section 3.2, we examine the evaluation function for the PTSP, which is computationally more expensive to calculate than e.g. the TSP evaluation function. A method is proposed for approximating the quality of a solution and thereby reach a compromise between complexity of computation and accuracy. Section 3.3 introduces the basic ACO algorithm for the PTSP (as well as other node$\times$node problems) and the modifications inspired by the stochastic nature of the problem. We propose two alternative schemes for providing heuristic information used by the ants during tour construction. The benefit of using approximate evaluation as well as using the improved heuristic schemes is shown empirically in Section 3.4, where the ACO algorithm is applied to a multitude of problem instances in a variety of configurations. We also compare the performance of ACO with that of Hilbert-Sorting followed by 1-Shift, which currently is one of the best known heuristics for solving PTSP instances [Bertsimas and Howell, 1993]. At the end of this chapter in Section 3.5, we summarize our results and give some indications for possible future work.

3.2 Evaluation of Solution Quality

This section deals with the evaluation function for PTSP, i.e. the function f which assigns a quality to a specific tour π based on its expected tour-length, where π is a permutation over $[0, n-1]$ and the tour consists of the edges $\{(\pi(i), \pi(i+1)) | i = 0, \dots, n-2\} \cup \{\pi(n-1), \pi(0)\}$. In the first subsection, we examine how to compute the complete PTSP evaluation function, while the second subsection deals with methods for calculating an approximate value for f.

3.2.1 Full Evaluation

For an n customer PTSP instance, there are 2^n possible realizations R_i, $i = 1, \dots, 2^n$, which contain the customers that will ultimately require a visit, i.e. a realization is a partitioning of the customers into the set A of those included and B of those left out of the final instance. Each realization can be assigned a probability $p(R_i) \in [0, 1]$ based on the probabilities for inclusion of the individual customers, i.e. $p(R_i) = (\prod_{i \in A} p_i) \cdot (\prod_{i \in B}(1 - p_i))$, with $\sum_{i=1}^{2^n} p(R_i) = 1$. Let $L_{R_i}(\pi)$ describe the (TSP-)tourlength of π for realization R_i under the assumption that customers not in R_i are skipped in the tour. The expected tour-length can then be formally described as

$$f(\pi) = \sum_{i=1}^{2^n} p(R_i) \cdot L_{R_i}(\pi). \tag{3.1}$$

Instead of using Equation 3.1, which takes exponentially long to compute, a version that needs only $O(n^2)$ steps to calculate the expected tourlength for PTSP was presented in [Jaillet, 1985]. The idea is to calculate the weighted sum over all distances between customers. A weight for the distance d_{ij} between customers i and j is equal to the probability that the corresponding edge e_{ij} will actually be included in the final tour. An edge e_{ij} is included in the final tour if and only if all customers between i and j in the PTSP tour are left out of the final realization. Note that for symmetric PTSPs, where $d_{ij} = d_{ji}$, e_{ij} is also effectively included in the tour if all customers from customer j to customer i are omitted, since the tour is a cycle. Using this notion, the expected tour length can be calculated as

$$f(\pi) = \sum_{i=0}^{n-1} \sum_{j=1}^{n-1} d_{\pi(i)\pi((i+j) \bmod n)} \cdot p_{\pi(i)} \cdot p_{\pi((i+j) \bmod n)} \cdot \prod_{k=i+1}^{i+j-1} (1 - p_{\pi(k \bmod n)}) \tag{3.2}$$

For homogeneous probability distributions, with $\forall i \in [0, n-1] : p_i = p \in [0, 1]$, this formula can be further simplified to

$$f(\pi) = p^2 \cdot \sum_{i=0}^{n-1} \sum_{j=1}^{n-1} d_{\pi(i)\pi((i+j) \bmod n)} \cdot (1 - p)^{j-1} \tag{3.3}$$

3.2.2 Approximate Evaluation

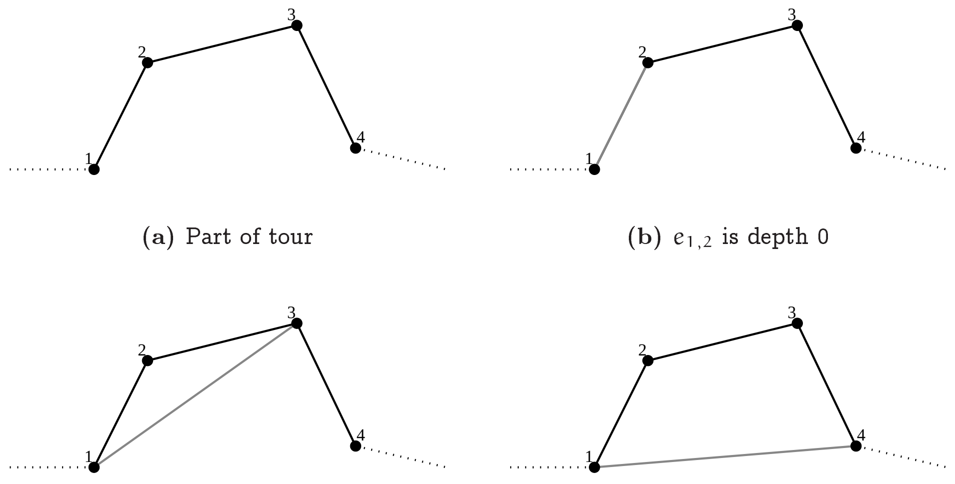
Despite of having a polynomial evaluation time for PTSP via Equation 3.2, the resulting $O(n^2)$ time for calculating the expected tour-length is still very long compared to the standard $O(n)$ evaluation for the non-probabilistic TSP, and [Bianchi et al., 2002a,b] have done some analysis on using the TSP evaluation function instead of the correct PTSP function in order to have more time for constructing solutions. The ACO meta-heuristic needs the quality of solutions primarily for determining the best solution constructed by the ants of an iteration, which, along with the best ant overall, updates the pheromone trail, see Algorithm 2.1, p.24. Thus, it is sufficient to find an approximation function f' which promotes solution $\pi_l, l \in [1, m]$ to best solution of an iteration with high probability, if π_l is indeed (one of) the best solution(s) according to Equation 3.2:

$$\forall j \in [1, m] : f(\pi_l) \leq f(\pi_j) \Rightarrow P(\forall j \in [1, m] : f'(\pi_l) \leq f'(\pi_j)) \geq 1 - \epsilon. \tag{3.4}$$

We call ϵ the relative promotion error.

It is possible to approximate the true PTSP-evaluation only up to a predetermined degree, with different trade-offs between evaluation accuracy and computational effort. We say that an edge e_{ij} or its corresponding distance d_{ij} has a depth $\theta \in [0, n-2]$ with respect to a given tour π if there are exactly θ customers on the tour between i and j, see Figure 3.2 for some examples.

A high θ for an edge e_{ij} implies a small probability for the edge to be actually part of a realized tour (as a large number of consecutive customers would have to be canceled), and thus a small impact on the resulting value for f. For the



(a) Part of tour (b) $e_{1,2}$ is depth 0

(c) $e_{1,3}$ is depth 1 (d) $e_{1,4}$ is depth 2

Figure 3.1: A portion of a tour in (a) and edges of varying depth in (b), (c), and (d) for this tour. Note that n edges of any depth exist.

homogeneous case where $p_i = p \in [0, 1]$ for all customers $i \in [0, n-1]$, this probability has the value $p^2 \cdot (1-p)^\theta$, decreasing exponentially in θ. Therefore, it might make sense to simply stop the evaluation once a certain depth has been reached. We define

$$f^\theta(\pi) = \sum_{i=0}^{n-1} \sum_{j=1}^{1+\theta} d_{\pi(i)\pi((i+j) \bmod n)} \cdot p_{\pi(i)} \cdot p_{\pi((i+j) \bmod n)} \cdot \prod_{k=i+1}^{i+j-1} (1 - p_{\pi(k \bmod n)}) \tag{3.5}$$

as the value of the evaluation up to depth θ. Even though $f^\theta(\pi)$ might be substantially smaller than $f(\pi)$, it nonetheless has the potential to provide a sufficiently small relative promotion error ϵ.

3.3 ACO for the PTSP

This section explains in detail the basic ACO algorithm used for the PTSP, as well as two new heuristic guidance schemes. The first of these schemes, called the Depth-based Heuristic, is based more accurately on the increase to overall solution quality implied by a local decision during tour construction than the TSP Heuristic which was used in [Bianchi et al., 2002a,b]. The other heuristic which is based on the characteristic zigzag patterns in PTSP tours [Jezequel, 1985] takes the angle between adjacent edges of the tour into consideration and is called the Angle-based Heuristic.

3.3.1 ACO algorithm for node×node Problems

Algorithm 2.1, p.24, describes the basic ACO algorithm we use without going into detail on how ant l constructs solution π_l. The process of constructing a solution is different for node×node and node×place problems, which is why this point was left intentionally vague. In Algorithm 3.1, we clarify this point for node×node problems in general and TSP type problems in particular .

Algorithm 3.1 Solution construction by ant l for node×node problems

```
1:  randomly choose start node $i \in_R S$
2:  set $\pi_l(0) = i$
3:  remove $i$ from selection set $S \mapsto S \setminus \{i\}$
4:  for decision $t = 1, \dots, n-1$ do
5:      randomly draw $q \in_R [0, 1]$
6:      if $q \leq q_0$ then
7:          choose $j = \arg\max_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ij}^{\beta}$
8:      else
9:          choose $j$ according to probability distribution
                $$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}}$$
10:     end if
11:     remove $j$ from selection set $S \mapsto S \setminus \{j\}$
12:     set $i \mapsto j$
13:     set $\pi_l(t) = i$
14: end for
```

When randomly drawing an element from a set or a number from an interval, we imply uniform distribution, i.e. equal probability for all results, unless otherwise stated. For the sake of completeness, Algorithm 3.1 also includes the pseudo-random proportional rule, even though it will not be used in this chapter (i.e. for evaluation we set $q_0 = 0$ in Section 3.4).

After the construction process, the solution π contains all cities in the order they were visited, i.e. the edges $(\pi(l), \pi(l+1))$ for $l \in [0, n-2]$ and $(\pi(n-1), \pi(0))$ are traversed. Hence, when updating the pheromone matrix, the notation $(i, j) \in \pi$ from Algorithm 2.1, p.24, translates to $(i, j) \in \{(\pi(l), \pi(l+$

$1))|l \in [0, n-2]\} \cup \{(\pi(n-1), \pi(0))\}$.

The basic node×node ACO algorithm can be modified to work more efficiently when dealing with symmetric problem instances, e.g. when dealing with TSPs with Euclidean distances. If $\forall i, j : d_{ij} = d_{ji}$ holds, then the tour can be constructed in either direction with the same solution quality (this also holds for PTSP due to commutativity). Thus, when updating the pheromone values corresponding to a tour, it makes sense to do the update in both directions, i.e. if $(i, j) \in \pi$, then update with (i, j) as well as (j, i). This symmetric update is shown in Algorithm 3.2. Note that the single positive update with $\rho/2$ has been replaced by two updates with $\rho/4$, keeping the update amount constant. We use the symmetric update for all symmetric instances.

Algorithm 3.2 Symmetric update

```
1: for all $(i, j) \in \pi^+$ do
2:     $\tau_{ij} \mapsto \tau_{ij} + \rho/4$
3:     $\tau_{ji} \mapsto \tau_{ji} + \rho/4$
4: end for
5: for all $(i, j) \in \pi^*$ do
6:     $\tau_{ij} \mapsto \tau_{ij} + \rho/4$
7:     $\tau_{ji} \mapsto \tau_{ji} + \rho/4$
8: end for
```

As mentioned above, the heuristic information used previously [Bianchi et al., 2002a,b] for Ant Algorithms when solving PTSP instances is the TSP Heuristic. This heuristic is defined via the pairwise distances d_{ij} between customers i and j

$$\forall i \neq j : \eta_{ij} = \frac{1}{d_{ij}} \tag{3.6}$$

assuming that $\forall i \neq j : d_{ij} > 0$ holds. This heuristic prefers to visit customers located close to the current position, and the deterministic version of this greedy scheme, known as *nearest neighbor*, achieves good results for many instances of TSP. It makes sense to choose the next customer according to the distance because the distance traversed is immediately incorporated into the tourlength; hence, on a more abstract level, the decision where to go next is governed by the immediate impact on solution quality. Of course, this need not lead to optimal or even good solutions, but usually it is a good indicator which way to go and therefore beneficial to the ants when constructing a solution.

However, for the PTSP, the distance d_{ij} is only an approximation of the increase to expected tourlength, the quality of which worsens rapidly for decreasing probability values, and the nearest neighbor approach has been shown to behave poorly on PTSP instances in [Bertsimas and Howell, 1993]. In the following two Sections (3.3.2 and 3.3.3), we propose two heuristic guidance schemes first introduced in [Branke and Guntsch, 2003] which more accurately reflect the impact of a decision on solution quality.

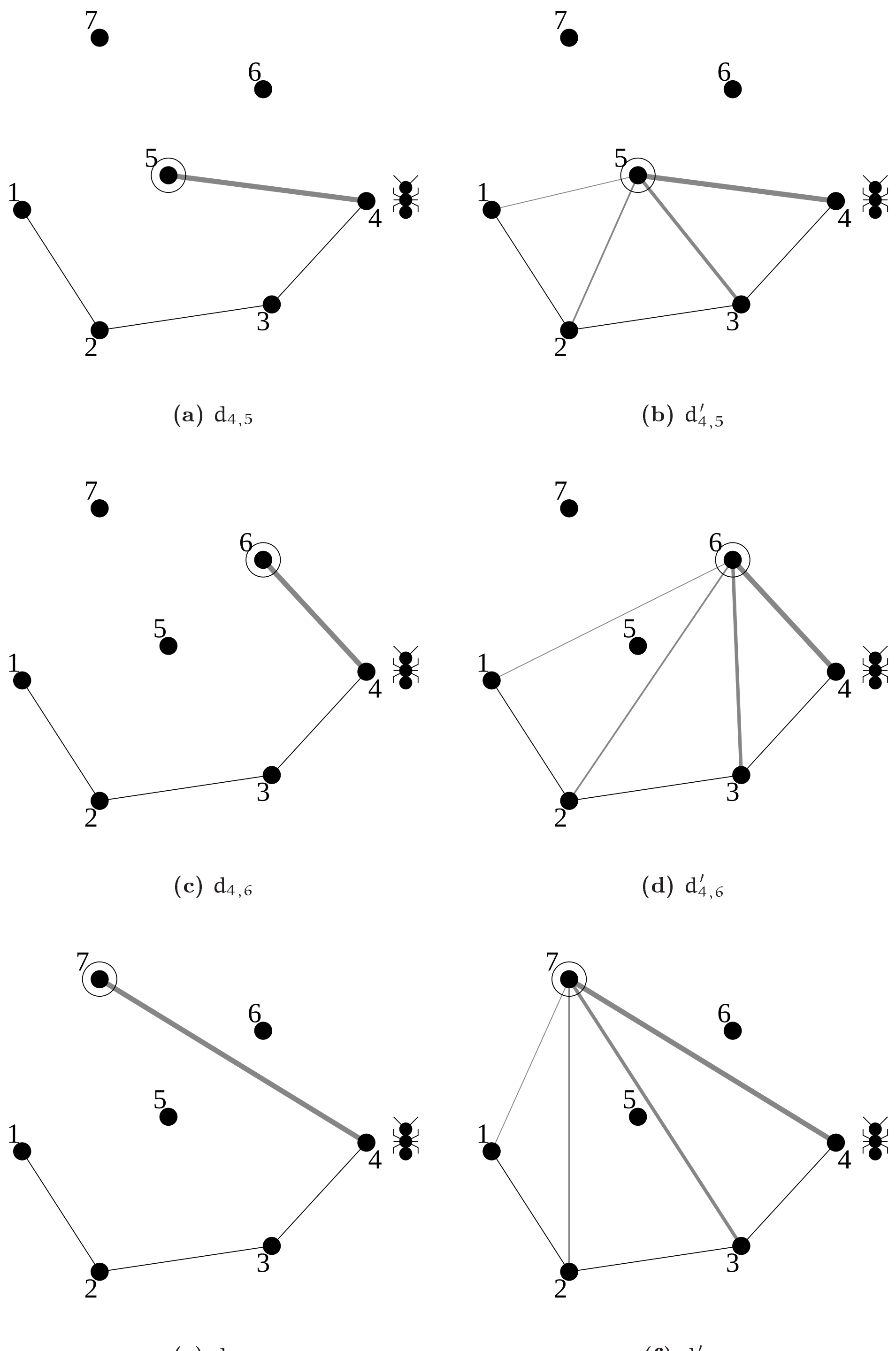
3.3.2 Depth-based Heuristic

As we have seen in Equation 3.2, when evaluating a solution of a PTSP instance, all distances must be considered, with the tour itself along with the probability values defining the weights of the individual distances. With the growing tourlength during the construction process, the estimation of the influence the next decision will have on the final solution quality becomes more and more accurate. All distances to the potentially next customer originating from any customer already integrated into the tour must be considered, since it is feasible that all customers between a customer i who is already planned and the target customer j are removed from the final instance. Formally, the distance d_{ij} is weighted by its likelihood, which is equal to the probability of customers i and j being included in the final instance and all customers in between being excluded. We define the probabilistic distance $d'_{ij}(\pi)$ depending on the partial tour π during the construction process as

$$d'_{ij} = \sum_{k=0}^{n-1-|S|} d_{\pi(k)j} \cdot p_{\pi(k)} \cdot p_j \cdot \prod_{h=k+1}^{n-1-|S|} (1 - p_{\pi(h)}) \tag{3.7}$$

Note that $\pi(n-1-|S|) = i$. This distance measure now encompasses the increase to the expected tourlength from the initial customer up to customer j; the rest of the tour beyond customer j is still unknown, and hence it cannot be used to judge the effect of choosing j. Figure 3.2 graphically illustrates normal and probabilistic distances for a partial tour.

Using d'_{ij} to define a PTSP Heuristic analogously to TSP has one drawback: the probability p_j of the target customer j being in the later realization is factored into the heuristic value, which is not very useful, since a tour visiting all the customers must be constructed, and decreasing the probability of going to a customer j based on its inclusion probability p_j will lead to j being included into the tour at a less opportune point, resulting in a higher expected tourlength. Therefore, we define a modified probabilistic distance d^P_{ij} as



(a) $d_{4,5}$ **(b)** $d'_{4,5}$

(c) $d_{4,6}$ **(d)** $d'_{4,6}$

(e) $d_{4,7}$ **(f)** $d'_{4,7}$

Figure 3.2: Normal (a),(c),(e) and probabilistic (b),(d),(f) distances between customers.

$$\begin{aligned} d^P_{ij} &= d'_{ij}/p_j \\ &= \sum_{k=0}^{n-1-|S|} d_{\pi(k)j} \cdot p_{\pi(k)} \cdot \prod_{h=k+1}^{n-1-|S|} (1 - p_{\pi(h)}) \end{aligned} \tag{3.8}$$

Using d^P_{ij} instead of d_{ij}, we define the Depth-based heuristic as

$$\forall i \neq j : \eta^P_{ij} = \frac{1}{d^P_{ij}} \tag{3.9}$$

Note that for homogeneous PTSP, the use of d'_{ij} instead of d^P_{ij} in Equation 3.9 would lead to identical results due to the construction of the probability distribution given in Equation 2.2, p.19, by which an ant makes local decisions.

At first glance, it seems that using the Depth-based Heuristic value entails an asymptotic increase in the time needed for solution generation in comparison to the TSP Heuristic, since computing d^P_{ij} by Equation 3.8 takes $O(n)$ time and must be performed $O(n^2)$ times for a total of $O(n^3)$ calculations per tour construction. The TSP Heuristic values are all computable in $O(1)$, leading to only $O(n^2)$ steps when constructing a tour. However, by maintaining a distance vector $D \in \mathbb{R}^n$ for storing the values of d^P_{ij}, a runtime of $O(n^2)$ per iteration is possible while using the Depth-based Heuristic. This is accomplished by modifying the construction process of an ant in ACO as shown in Algorithm 3.3 .

If the distance value $d^P_{ij}(t)$ used by the ant for decision t is known, and customer k is visited instead of customer j, then calculating the new value $d^P_{ij}(t+1)$ is done by considering two possibilities. Either customer k will not be included in the later instance, which corresponds to the term $(1-p_k)\cdot d^P_{ij}(t)$, or it will be included, in which case only the distance d_{ik} is relevant, resulting in the term $p_k \cdot d_{ik}$. The sum of these terms is the value for $d^P_{ij}(t+1)$. Note that the values for $d^P_{ij}(t)$ do not rise monotonously, as might at first glance be expected.

Each execution of the loop takes $O(n)$ time, and it is executed $\Theta(n)$ times. Due to the possibility of implementing this heuristic with only a negligible runtime penalty, the Depth-heuristic with limited depth as introduced in [Branke and Guntsch, 2003] is no longer pursued, since it not only requires a greater computational effort but is also less precise than using the full depth.

Algorithm 3.3 Fast implementation of Depth-based Heuristic

```
1: Initialize D to zero, i.e. $\forall j = 0, \ldots, n-1 : D_j = 0$
2: randomly choose start node $i \in_R S$
3: set $\pi_l(0) = i$
4: remove i from selection set $S \mapsto S \setminus \{i\}$
5: for decision $t = 1, \ldots, n-1$ do
6:     set $\forall j \in S : D_j = D_j \cdot (1 - p_i) + d_{ij} \cdot p_i$
7:     set $\forall j \in S : \eta^P_{ij} = 1/D_j$
8:     randomly draw $q \in_R [0, 1]$
9:     if $q \le q_0$ then
10:        choose $j = \arg\max_{h \in S} \tau^\alpha_{ih} \cdot (\eta^P_{ij})\beta$
11:    else
12:        choose j according to probability distribution
               $$p_{ij} = \frac{\tau^\alpha_{ij} \cdot (\eta^P_{ij})^\beta}{\sum_{h \in S} \tau^\alpha_{ih} \cdot (\eta^P_{ih})^\beta}$$
13:    end if
14:    remove j from selection set $S \mapsto S \setminus \{j\}$
15:    set $i \mapsto j$
16:    set $\pi_l(t) = i$
17: end for
```

3.3.3 Angle-based Heuristic

Another heuristic which attempts to give a better indication as to which customer to choose next is an Angle-based Heuristic . This heuristic is different from radial sort [Bertsimas and Howell, 1993], which sorts all nodes in a single circular pass. Instead, when deciding which customer to visit next, the ant takes into account the angle between the last traversed edge and the one implied by the choice of the next customer.

To explain why the angle between adjacent edges of a tour could be exploited heuristically, consider the placement of customers in the plane and respective tours shown in Figure 3.3. The tour on the left is a good TSP tour, while the tour on the right shows a good PTSP tour assuming e.g. a homogeneous value of $p = 0.5$ for all customers. The reason for this can be derived from Equation 3.2, which lets us compute the size of the coefficient for any distance d_{ij} between two customers i and j. For the example of $p_i = 0.5$, all distances of depth 1 have a coefficient of just over half the size of the distances of depth 0, which are the only ones used for TSP evaluation. The sum of distances of depth 1 are by far greater for the good TSP tour than for the good PTSP tour, and are not compensated by the shorter distances of depth 0. Since the distances at depths $\theta > 1$ are not significantly different, and their influence decreases exponentially, the expected tourlength for the right tour in Figure 3.3 is shorter than for the left tour.

Distances of depth 1 tend to be small when the tour has many sharp angles, since a sharp angle sends the ant back in the direction of the customer it just came from. Thus, letting the ant prefer sharp angles may guide it toward significantly better solutions. If the angle between two edges is γ, then we want the ants to make decisions influenced by $\cos\gamma$, which is defined for vectors u, v as $\cos\gamma = \langle u, v\rangle / (\|u\| \cdot \|v\|)$, with $\langle u, v\rangle$ being the scalar product of u and v, and $\|u\| = \langle u, u\rangle$. With $\cos\gamma = 1$ for u and v parallel, $\cos\gamma = 0$ for u perpendicular to v, and $\cos\gamma = -1$ for u and v in opposite directions, we construct

$$\eta^{\angle}_{ij} = \eta^{T}_{ij} \cdot \frac{1}{2}(1 - \frac{\langle u, v\rangle}{\|u\| \cdot \|v\|}). \tag{3.10}$$

It may be too extreme to practically deny any movement in the same direction as the previous move, for instance for very close customers, which is why a

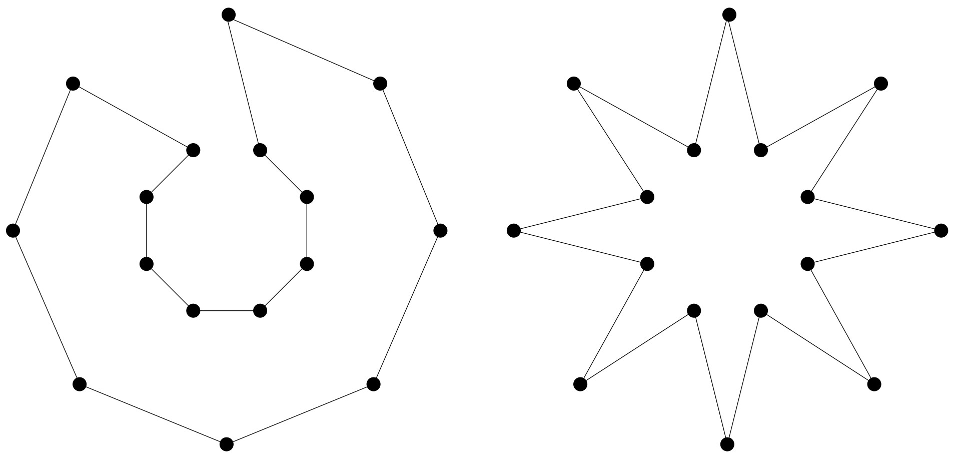

(a) Good TSP Tour **(b)** Good PTSP Tour

Figure 3.3: Comparison of the characteristics of TSP and PTSP tours.

linear combination with the TSP-heuristic of the form

$$\eta_{ij}^{\angle,c} = c \cdot \eta_{ij}^{\angle} + (1-c) \cdot \eta_{ij}^{T} = \eta_{ij}^{T} \cdot \left(1 - \frac{c}{2}\left(1 + \frac{\langle u, v \rangle}{\|u\| \cdot \|v\|}\right)\right) \tag{3.11}$$

with $c \in [0,1]$ could be appropriate. In previous work [Branke and Guntsch, 2003], the parameter c has been treated as a constant. However, for heterogeneous PTSP instances, it makes sense to adapt the parameter to the probability p_i of the customer the ant is currently located at. A customer i with a very low value p_i will probably not be part of a later realization, which is why including the customer between two other customers which lie close to one another, resulting in a sharp angle, is opportune. On the other hand, if a customer is almost certainly part of a later realization, then only the distance to its successor in the tour is relevant, and not the angle. We therefore study the effect of an adaptive c defined by

$$c = 1 - \frac{p_i}{2} \tag{3.12}$$

for heterogeneous problems.

When intensifying the influence of Angle-based Heuristic information by using a value of $\beta > 1$ in Equation 2.2, p.19, we apply this value only to the distances d_{ij}. The reason for applying a high β value is usually given by the fact that the individual heuristic values lie too closely together to effectively distinguish between good and bad choices. Since this is true only for the distance values and not the values of $\cos\gamma$, we choose to apply β only to the distance values used in Equation 3.11. In any case, it would be possible to achieve a similar effect as applying β to the entire equation by modifying c correspondingly.

Calculation of the value for $\eta_{ij}^{\angle,c}$ is possible with a constant computational effort. However, the constant is not very small, since a number of arithmetic operations must be carried out for determining each heuristic value. Therefore, it is more efficient to calculate all possible $O(n^3)$ (all incident edges must be considered) heuristic values during preprocessing and reference them later.

3.4 Empirical Evaluation

3.4.1 Test Setup

Currently, no designated benchmark instances for evaluating algorithms designed to solve the PTSP exist. Instead, it has been common practice to either generate distances and probabilities randomly [Laporte et al., 1994] or use instances from the [TSPLIB, 2003], which are supplemented with probabilities for the individual customers [Bianchi et al., 2002a,b]. Each of these approaches has its respective merits. Random instances with the same expected solution quality can be generated en masse, hence enabling some estimation of how good the algorithm performs on an unknown, random distribution of customers. Testing repeatedly on a known, fixed instance enables a better understanding of how the performance of the algorithm varies for the particular structure of that instance and can more easily serve as a basis for comparison with other algorithms.

In order to make our evaluation as sound as possible, we use two sets of test problems. The first set of problems uses the customer coordinates from the three Euclidean TSP instances `eil101`, `kroA200`, and `rd400` taken from the [TSPLIB, 2003], with $n = 101$, $n = 200$, and $n = 400$ customers respectively. Each set of customer coordinates is augmented with five different probability distributions for the occurrence of a customer: homogeneous probabilities of $p \in \{0.25, 0.5, 0.75\}$, and heterogeneous distributions, generated by choosing p_i randomly in the range $[0.3, 0.7]$ or $[0.1, 0.9]$ in such a way that the average probability for a customer to be included is exactly 0.5. Note that the heterogeneous probability distributions are completely different for each of the three customer location sets. Since the ACO algorithm is probabilistic, the resulting 15 PTSP instances are tested by running each algorithm 100 times using different random seeds and averaging the results.

For the second set of problems, 100 instances of 100 customer locations are randomly generated in the unit square $[(0,0),(1,1)] \subset \mathbb{R}^2$. This set is subsequently denoted as `rnd100`. Again, each problem is combined with the five different probability distributions described above, with each heterogeneous distribution being unique. This results in five groups (resulting from the different probability distributions) of 100 similar PTSP instances each. On these problem instances, each algorithm has been tested only once, and averages over all 100 problem instances in a group are reported.

The primary goal of this section is to demonstrate the general suitability of Ant Colony Optimization and the benefits of our proposed modifications for the PTSP. Therefore, practically no parameter tuning was performed. Instead, we use relatively standard settings for the TSP, which are $\alpha = 1$, $\beta = 5$ for a strong heuristic guidance, and $\rho = 0.001$. The number of iterations was set to 30000 for $n \in \{100, 101\}$, 40000 for $n = 200$, and 50000 $n = 400$, increasing to compensate for the higher complexity of the instances, which in turn induces a slower convergence process of the pheromone matrix. Note that since all instances have Euclidean distances and are therefore symmetric, we use the update rule described in Algorithm 3.2.

The pheromone values are initialized with $\tau_0 = 1$ in all cases. This leads to a longer initial search phase in which the influence of the pheromone is minimal, and the ants proceed probabilistically according to the heuristic information. Due to the update rule used in Algorithm 3.2, the sum of pheromone values in a row/column, which is initially $n - 1$ (since $\forall i \neq j : \tau_{ij} = \tau_0$), slowly approaches a value of 1, since for sum values greater than 1, more pheromone is evaporated than updated for every row/column.

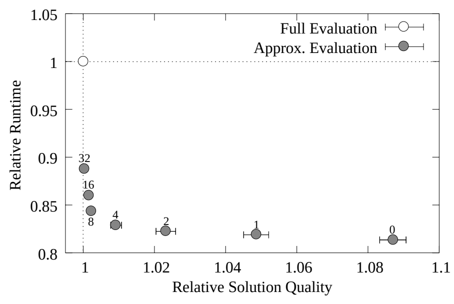
3.4.2 Approximate Evaluation

In order to determine the effect of using an approximate evaluation function of limited depth instead of the complete evaluation, we test evaluation depths of $\theta \in \{0, 1, 2, 4, 8, 16, 32, n-2\}$. Note that for $\theta = 0$ in homogeneous cases, the resulting tour-length corresponds to the TSP tour-length scaled by p^2, and $\theta = n - 2$ is equivalent to full evaluation in all cases. In order to avoid any side-effects due to specialized tour construction when comparing evaluation depths, we use the TSP-heuristic in the algorithm.

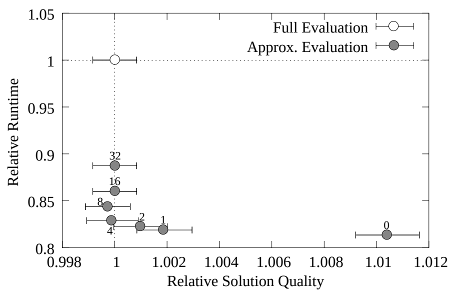
As expected, with increasing approximation depth, the ACO algorithm performs more and more like it is using the full evaluation function. The lower the approximation depth, the greater the likelihood of the algorithm to make wrong decisions, i.e. to promote one tour as the iteration's best while in fact a different tour is actually better. If this mistake is done often enough, the pheromone matrix will focus on a part of the search space which seems promising to the approximation function, but which is sub-optimal according to the complete evaluation, resulting in a worse solution quality.

The Figures 3.4 and 3.5 illustrate the relative runtime and solution quality for the ACO algorithm using approximate evaluation on the PTSP instances based on `eil101` after 30000 iterations and `rd400` after 50000 iterations respectively. The results for `rnd100` and `kroA200` are omitted here since they are virtually identical to `eil101` or lie between those for `eil101` and `rd400`, respectively. The basis for comparison is full evaluation in all cases, which by definition is located at $(1, 1)$. The savings in runtime achieved due to approximation lie between 11.27% and 18.65% on `eil101` and 17.65% to 23.87% on `rd400`, for approximation depths $\theta = 32$ to $\theta = 0$ and irrespective of the value of p.

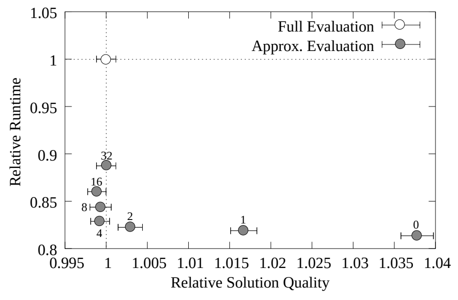
However, for small approximation depths, these savings in runtime come at



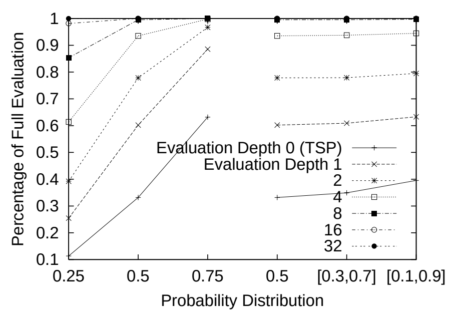
(a) $p = 0.25$



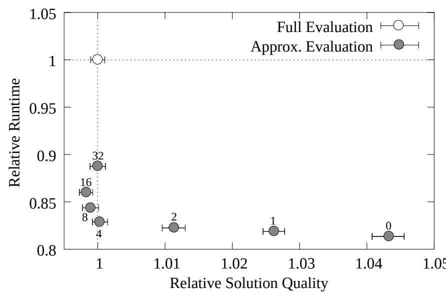
(b) $p = 0.75$



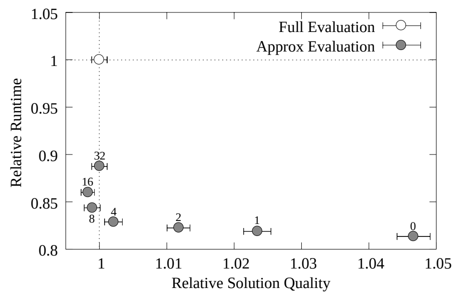
(c) $p = 0.5$



(d) Approximation Value

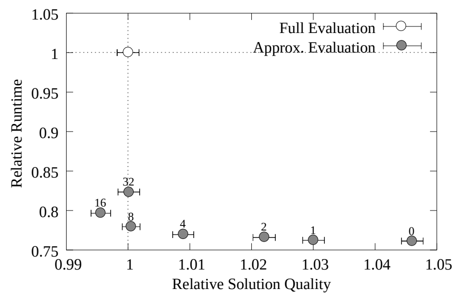


(e) $p \in [0.1, 0.9]$

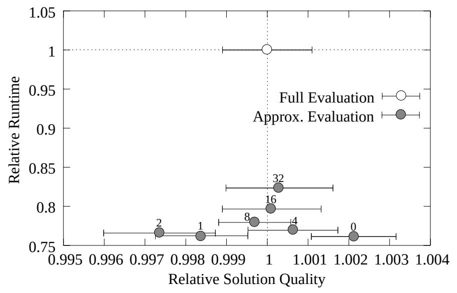


(f) $p \in [0.3, 0.7]$

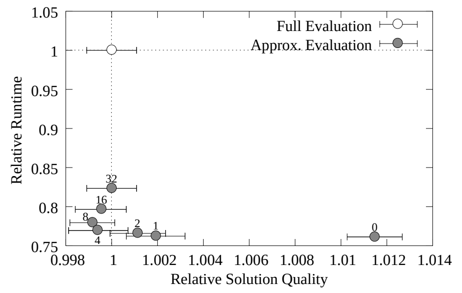
Figure 3.4: Relative runtime and solution quality of indicated approximation depths θ for 5 PTSP instances based on `eil101` in (a)-(c),(e),(f). The error bars show standard deviation. Subfigure (d) shows the percentage of the tour length according to full evaluation computed by the approximations.



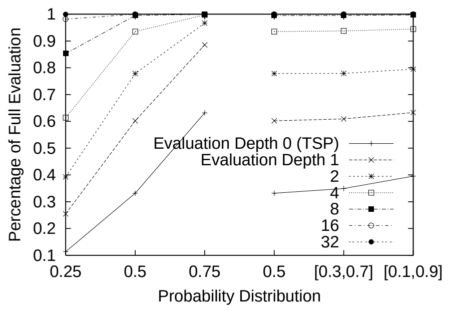
(a) $p = 0.25$



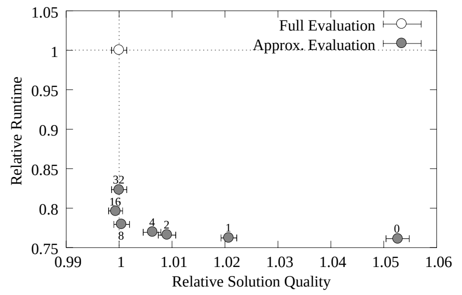
(b) $p = 0.75$



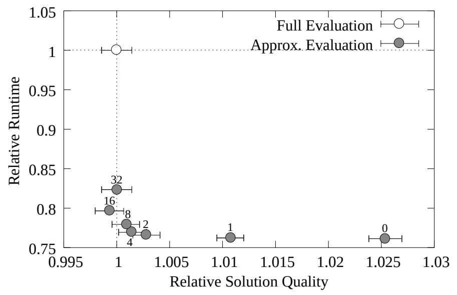
(c) $p = 0.5$



(d) Approximation Value



(e) $p \in [0.1, 0.9]$



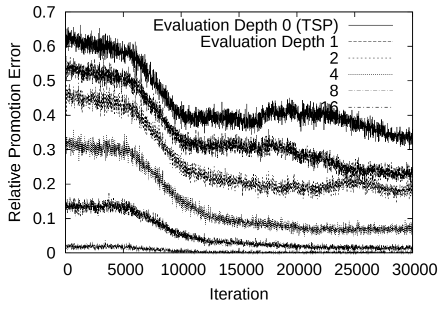
(f) $p \in [0.3, 0.7]$

Figure 3.5: Relative runtime and solution quality of indicated approximation depths θ for 5 PTSP instances based on rd400 in (a)-(c),(e),(f). The error bars show standard deviation. Subfigure (d) shows the percentage of the tour length according to full evaluation attained by the approximations.

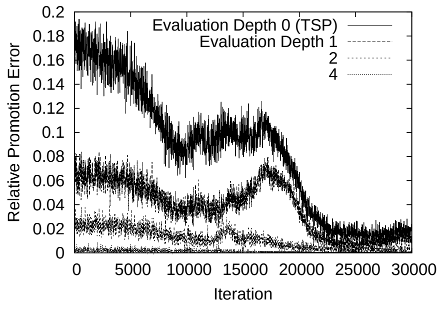
the expense of a worse solution quality. For the eil101 instances, a solution quality close or identical to that achieved by full evaluation is realized for $\theta \geq 8$ with $p = 0.25$, $\theta \geq 4$ for the scenarios $p = 0.5, p \in [0.3, 0.7], p \in [0.1, 0.9]$, and $\theta = 1$ for $p = 0.75$, where this approximation depth leads to a less than 0.2% worse relative solution quality than the full evaluation. The reason for this decrease in "necessary" approximation depth lies in the fact that edges of a greater depth have a steadily decreasing influence on the overall solution quality for rising values of p. The Subfigure 3.4(d) shows how large the value returned by the indicated approximation of the evaluation function is in comparison to the correct value from the full evaluation function. It seems that once a value of around 90% is reached by an approximation function, i.e. the discarded edges account for only 10% of the full solution quality, the degree of approximation is sufficient to choose the correct ant of an iteration as best. On a side note, the edges of higher depth seem to play a marginally less important role for heterogeneous problems, since the weights for a distance are computed as a product of values with a given global average, and this product is highest if all values are equal to the average.

For the results on the rd400 based instances shown in Figure 3.5, there are many similarities to eil101, but also some noteworthy difference. In comparing the individual approximation depths with one another and the full evaluation, the same depths θ as for eil101 are necessary to achieve a virtually identical solution quality as the full evaluation. In general, however, a given approximation depth θ performs significantly better on these larger instances with 400 customers than on the smaller ones with 101 customers, despite achieving only the same degree of approximation in terms of the relative portion of the solution quality it calculates, see Subfigure 3.5(d). For $p = 0.75$, a $\theta = 0$ approximation is already sufficient for getting within 0.25% of the solution quality found by the full evaluation.

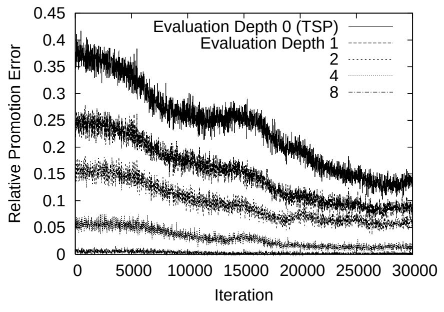
As we have mentioned earlier, the reason for the bad performance of the lower approximations is that they do not recognize the best ant of an iteration and instead let a different ant perform the update to the pheromone matrix and perhaps even replace the elitist ant. The Figures 3.6 and 3.7 show the relative promotion error for eil101 and rd400 respectively. As we can see, depending on the approximation depth θ, the algorithm quite often promotes the wrong ant for updating, with an initial probability of more than 60% for the instance eil101, $p = 0.25$, and $\theta = 0$, and a 44% average over the 30000 iterations runtime. The $\theta = 8$ approximation, which performs on par with full evaluation for this scenario, averages at just over 5% relative promotion error, with a



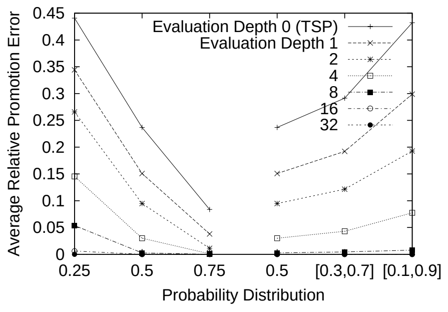
(a) $p = 0.25$



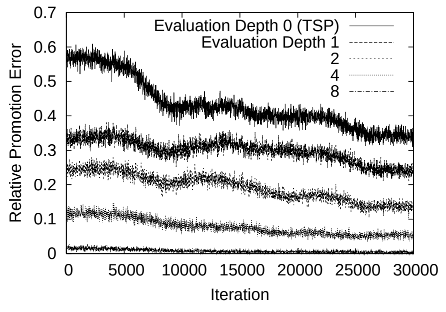
(b) $p = 0.75$



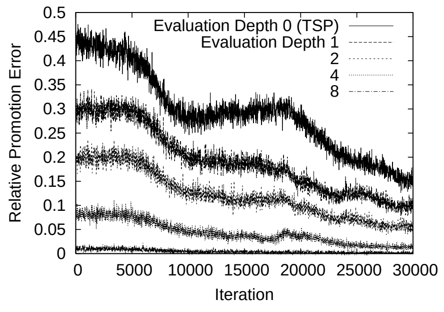
(c) $p = 0.5$



(d) avg. Relative Promotion Error

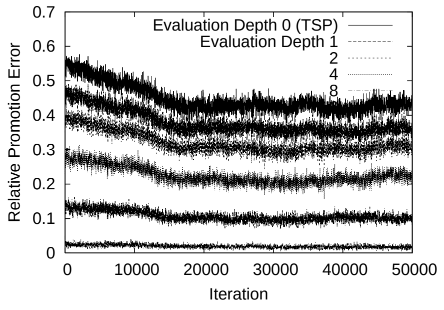


(e) $p \in [0.1, 0.9]$

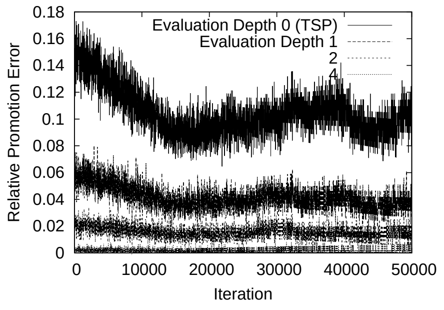


(f) $p \in [0.3, 0.7]$

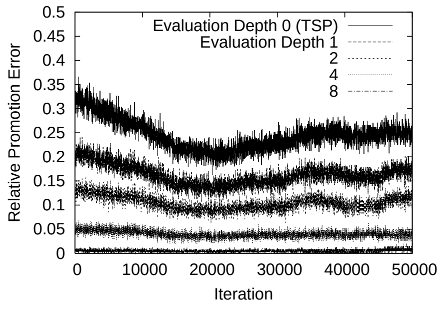
Figure 3.6: Relative promotion error of indicated approximation depths θ for 5 PTSP instances based on eil101 in (a)-(c),(e),(f). Subfigure (d) shows the average relative promotion error over 30000 iterations for the respective probability distributions. The oder of the curves from top to bottom is identical to the ordering in the legend.



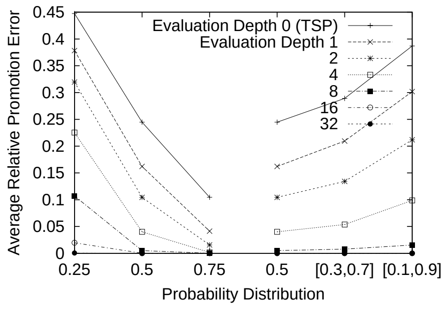
(a) $p = 0.25$



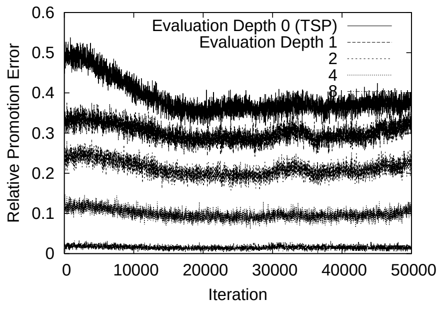
(b) $p = 0.75$



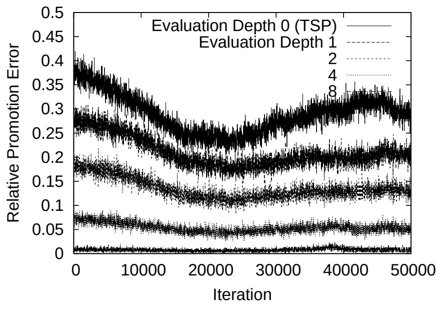
(c) $p = 0.5$



(d) avg. Relative Promotion Error



(e) $p \in [0.1, 0.9]$



(f) $p \in [0.3, 0.7]$

Figure 3.7: Relative promotion error of indicated approximation depths θ for 5 PTSP instances based on rd400 in (a)-(c),(e),(f). Subfigure (d) shows the average relative promotion error over 30000 iterations for the respective probability distributions. The oder of the curves from top to bottom is identical to the ordering in the legend.

maximum of around 15% at the start. The approximation depths deemed sufficient for the other scenarios, i.e. $\theta = 4$ for $p = 0.5, p \in [0.3, 0.7], p \in [0.1, 0.9]$ and $\theta = 1$ for $p = 0.75$, have an average relative promotion error of 3% and 3.8% respectively, with maximum values around 8%. The numbers are similar if somewhat higher for `rd400` with averages of 10.7%, 4% and 4.1% for the three classes above respectively. This is surprising, since it indicates that although the approximation is making more errors, i.e. promoting the wrong ant more often in `rd400` than in `eil101`, it still finds a better relative solution quality. Possible explanations could be that the tour promoted instead of the actually best tour is only marginally inferior, or that the approximated fitness landscape is easier to search, compensating for the promotion errors.

An interesting point is the development of the relative promotion error for an increasing heterogeneity of the problem instance. In contrast to the necessary degree of approximation in order to obtain a good solution, which is the same for $p = 0.5$, $p \in [0.3, 0.7]$, and $p \in [0.1, 0.9]$, the promotion error increases significantly over these three scenarios. The reason could be that the variance of the difference between approximated and full solution quality increases with heterogeneous instances, resulting in more erroneous promotions, while at the same time the approximated fitness landscape contains less local optima to trap the search process.

3.4.3 Improved Heuristic Guidance

For comparing the heuristics proposed in Subsections 3.3.2 and 3.3.3, we always use the full evaluation function in order to eliminate the possibility of wrong evaluations being the cause for poor performance. In order to gauge whether the new heuristic functions actually do offer superior guidance compared to the TSP Heuristic, our first evaluation concentrates on running the proposed heuristics in a greedy fashion on the `rnd100` distributions explained in Subsection 3.4.1 with homogeneous probabilities from 0 to 1 in increments of 0.02. The result is shown in Figure 3.8.

For each instance, the average of starting once from each possible customer was taken. Each point in the curve represents the average over all 100 random instances, i.e. of 10000 runs, of the relative performance compared to the greedy TSP Heuristic, also known as nearest neighbor, on that instance. The Depth-based Heuristic offers the highest potential for guiding ants toward better solutions, especially for low homogeneous values of p. However, when used with sensible parameter settings, the Angle-based Heuristic also outper-

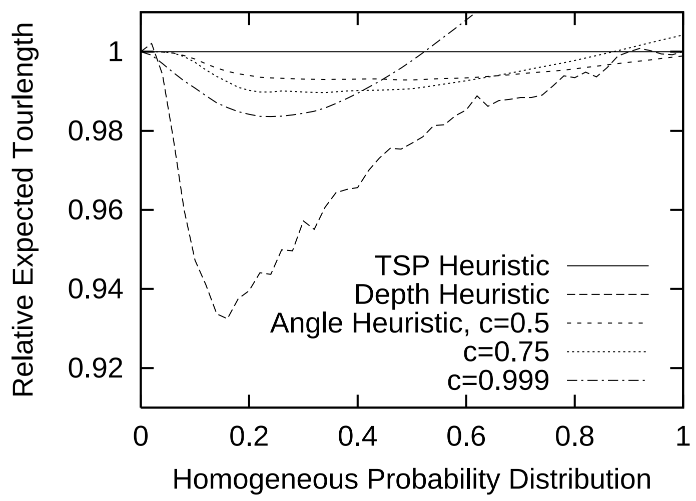


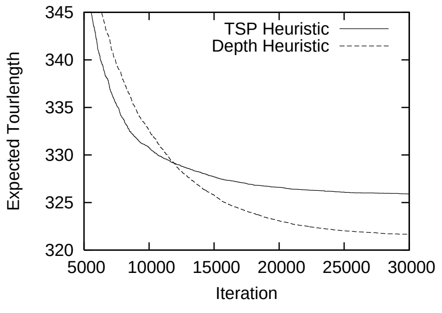
Figure 3.8: Comparison of average performance over 10000 runs of greedy variants of TSP, Depth-based, and Angle-based Heuristic for several parameters.

forms the TSP Heuristic, albeit either by only a small margin, or at the cost of a worse performance for higher values of p.

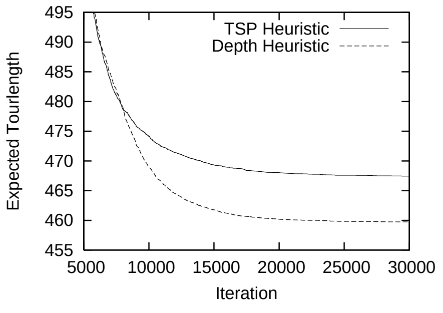
The following evaluations of ACO with the respective heuristic guidance schemes are based on the customer distribution `eil101` with the specified probability distributions. Unless otherwise stated, the results are practically the same for the other instances aside from the greater number of iterations needed for convergence.

The solution quality of ACO with the Depth-based Heuristic compared to ACO with the TSP Heuristic is shown in Figure 3.9.

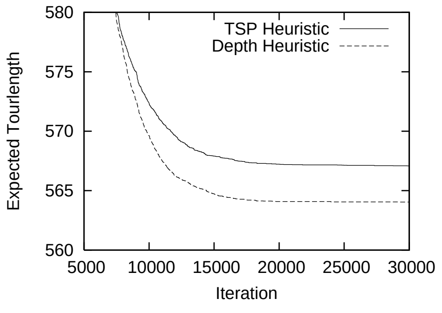
Starting with Figure 3.9(a), we see a typical phenomenon for the Depth-based Heuristic. Although the final solution quality is significantly better than for ACO with the TSP Heuristic, the convergence is slower. ACO with the Depth-based Heuristic needs roughly 12000 iterations, i.e. 40% of the runtime, before its solution quality becomes better than ACO with the TSP Heuristic. To a lesser extent, this effect also occurs for $p = 0.5$ and $p = 0.75$, but is not visible in Figures 3.9(b) and 3.9(c) due to the scaling used to show final solution quality. Figure 3.9(d) shows that the effect of slow convergence is intensified for heterogeneous instances, ACO with the Depth-based Heuristic requiring 20000 iterations before it performs better than TSP-guided ACO. This indicates that heterogeneous instances are perhaps more difficult to solve than homogeneous ones, since utilizing the information about the probability



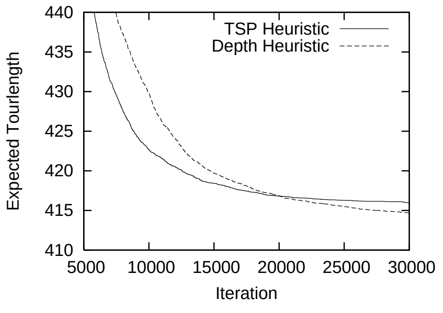
(a) $p = 0.25$



(b) $p = 0.5$



(c) $p = 0.75$



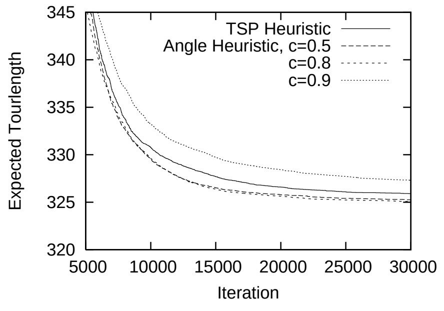
(d) $p \in [0.1, 0.9]$

Figure 3.9: Comparison of solution quality for ACO with Depth-based Heuristic vs. TSP Heuristic on `eil101`.

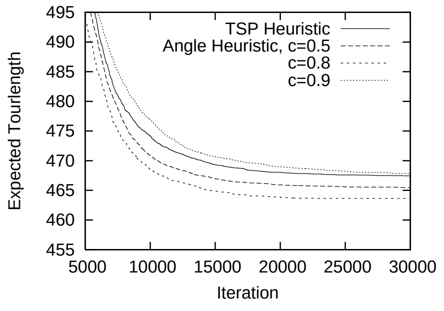
of a city being included gives a smaller advantage compared to the TSP than for the homogeneous instances.

For the Angle-based Heuristic, we test values of $c \in [0.5, 0.9]$ in increments of 0.05. For the sake of clarity, only the results for $c \in \{0.5, 0.8.0.9\}$ are presented in Figure 3.10; the results for the other parameters can be interpolated from these values.

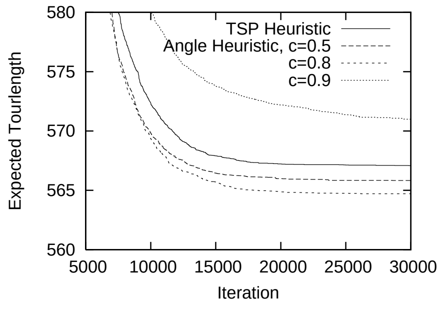
The first notable difference between the Angle-based and the Depth-based Heuristic is that the curve for the ACO with Angle-based heuristic and $c = 0.8$ is located beneath the curve for the ACO with the TSP Heuristic for every instance during the entire run. The parameter $c = 0.8$ performs best for all the homogeneous instances studied, and is beaten only marginally by the adaptive c (see Equation 3.12) on the heterogeneous instances. The other constant parameters $c = 0.5$ and $c = 0.9$ represent the respective end of the evaluated spectrum of parameters, and perform worse than $c = 0.8$ and for the



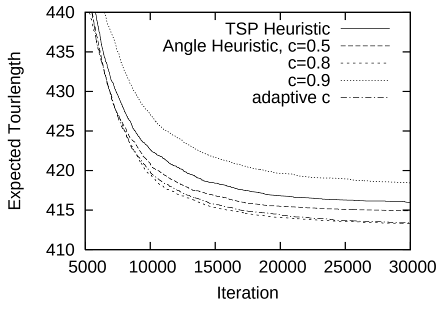
(a) $p = 0.25$



(b) $p = 0.5$



(c) $p = 0.75$



(d) $p \in [0.1, 0.9]$

Figure 3.10: Comparison of solution quality for ACO with Angle-based Heuristic vs. TSP Heuristic on `eil101`.

case of $c = 0.9$ even worse than the TSP Heuristic. It should be noted that all values $c \in [0.65, 0.8]$ perform well and are very close in terms of performance, making c a relatively robust parameter in this interval.

So far, we have neglected to mention that these improved heuristics, while resulting in a better solution quality in the same number of iterations, take longer to calculate than the TSP Heuristic, the values of which are constructed completely during preprocessing at the beginning of the algorithm. The values for the Angle-heuristic can also be calculated completely during preprocessing, requiring an $O(n^3)$ sized data structure since all incident edges must be considered. Accessing this data structure takes significantly longer than the $O(n^2)$ size data structure for the TSP Heuristic. Algorithm 3.3 gives a fast implementation for deriving the values for the Depth-based Heuristic, but the exponentiation with β must be done at runtime, which costs several multiplications. The values that have been derived from our implementation as time units per iteration for the three heuristics employed are shown in Table 3.1.

Table 3.1: Time units per iteration for ACO with specified heuristic.

TSP Heuristic	Depth-based Heuristic	Angle-based Heuristic
1	1.43335	1.24021

Since neither of the two new heuristics asymptotically increases the runtime for the ant decision process, the slowdown can effectively be bounded by the displayed constants. Figure 3.11 shows an appropriately scaled comparison between the ACO versions using the TSP, Depth-based, and Angle-based Heuristic on `eil101` with a homogeneous value of $p = 0.5$. As can be seen, the Depth-based Heuristic ultimately still outperforms the other two, even though no recalibration of the evaporation rate was undertaken to compensate for the fewer iterations available.

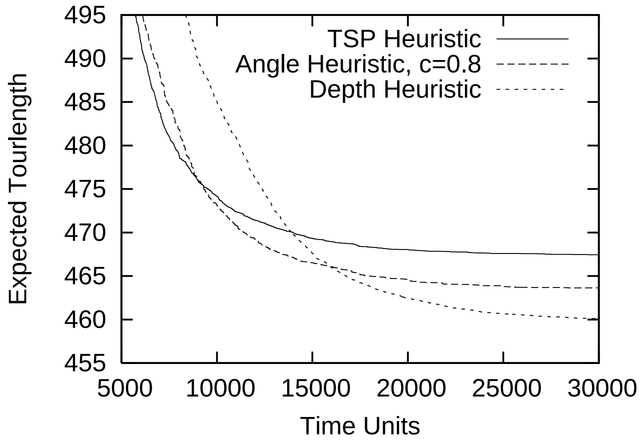


Figure 3.11: Comparison of ACO using the TSP, Depth-based, and Angle-based heuristic on `eil101` with $p = 0.5$.

The only apparent drawback of the Depth-based Heuristic is the long time it takes the ACO algorithm to find a good part of the solution space. A possible explanations for this behavior is that the heuristic values computed for the Depth-based Heuristic are too uniform to provide useful guidance at the start of the algorithm. Figure 3.12 shows the progression of the ratio between maximum and minimum probabilistic distance perceived by the ants during a tour construction for homogeneous PTSP instances based on `rnd100` with $p \in \{0.25, 0.5, 0.75, 1.0\}$, with $p = 1.0$ corresponding to a TSP instance, i.e. using the "normal" distance d_{ij} between customers i and j. Note that for $n = 100$, only 99 decisions are made, since the starting customer is chosen randomly, and for the final decision, the ratio is always 1 since there is only one customer left in S.

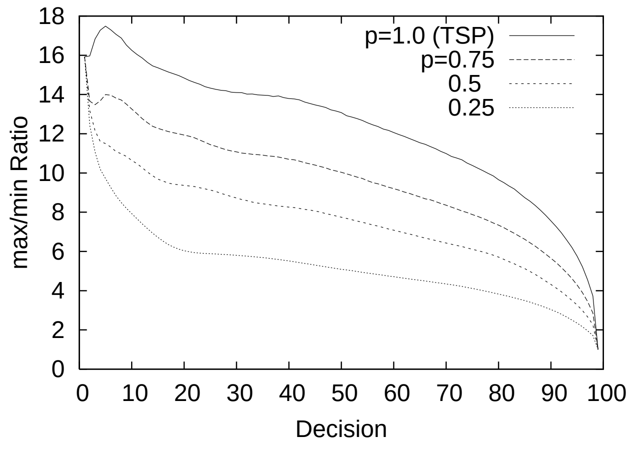


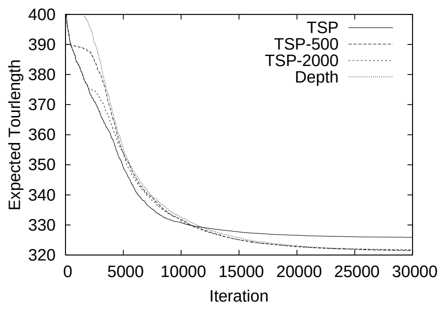
Figure 3.12: Development of the quotient between maximum and minimum probabilistic distance over the course of a tour construction for several homogeneous instances based on `rnd100`.

Figure 3.12 indicates that the distance values and therefore the heuristic values are located in a smaller interval for smaller, homogeneous values of p. The curves for the heterogeneous probability distributions lie in the vicinity of $p = 0.5$ and are not shown.

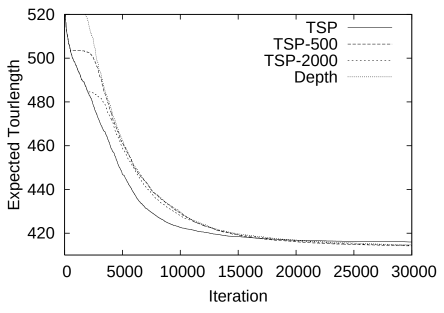
As a possible remedy for the slow early convergence, we let the ACO algorithm run with the TSP Heuristic during the initial phase of the algorithm, and then switch to the Depth-based Heuristic after a number of iterations t. Figure 3.13 shows the resulting solution quality for $t \in \{500, 2000\}$.

As we can see, immediately after the switch from the TSP to the Depth-based Heuristic, the solution quality stops improving and instead draws nearer to the curve of the solution quality when using only the Depth-based Heuristic. The final solution quality after 30000 iterations is virtually identical to using only the Depth-based heuristic.

If the Depth-based Heuristic cannot immediately improve upon tours constructed via the TSP Heuristic, it is perhaps more promising to use the Depth-based Heuristic from the start, but with a higher weight β in Equation 2.2, p.19. This also has the goal of making the weighted heuristic values η^{β} used during tour construction less uniform and therefore more capable of providing guidance. However, as Figure 3.14 shows, a quicker early convergence comes at the cost of a worse final solution quality.

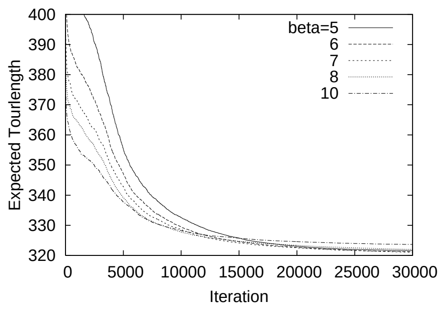


(a) $p = 0.25$

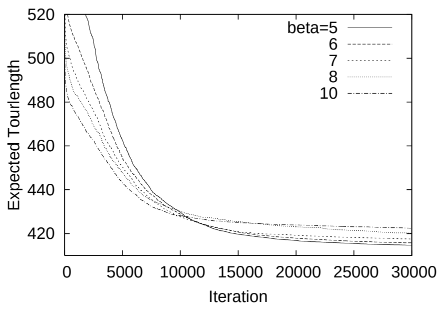


(b) $p \in [0.1, 0.9]$

Figure 3.13: Comparison of pure TSP and Depth-based Heuristic with two-phase variations, where the TSP Heuristic is run up to iteration $t \in \{500, 2000\}$ and then replaced by the Depth-based Heuristic, on `eil101`.



(a) $p = 0.25$



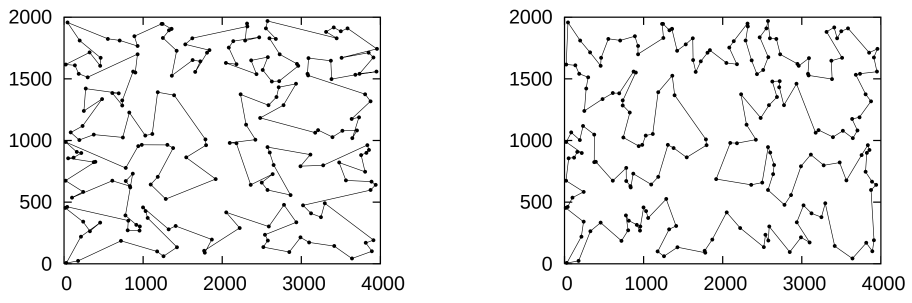
(b) $p \in [0.1, 0.9]$

Figure 3.14: Comparison of different heuristic weights β for the Depth-based Heuristic on `eil101`.

Ultimately, the two modifications for the Depth-based Heuristic with the aim of quicker early convergence and an implied better overall solution quality did not result in significantly better results, being mostly worse and only marginally better in some cases. Therefore, in the following subsection, we continue to employ the original Depth-based Heuristic as proposed in Subsection 3.3.2.

3.4.4 Comparison with Hilbertsorting and 1-Shift

Hilbertsorting, which assigns the index on the space-filling Hilbertcurve to each point in a two-dimensional plane, is a quick method for finding solutions to the TSP [Bartholdi and Platzman, 1982], and has also been applied to the PTSP [Bertsimas and Howell, 1993]. When combined with the 1-Shift heuristic (also in [Bertsimas and Howell, 1993]), which iteratively performs the best reinsertion of a node between two others in the tour until no further improvements can be attained, this method is currently one of the best known for finding solutions to PTSP instances. Figure 3.15 shows the respective paths resulting from Hilbertsorting, and from Hilbertsorting with subsequent application of the 1-Shift heuristic for kroA200 and a homogeneous value of $p = 0.5$. Note that the value of p only influences the 1-Shift heuristic.



(a) Hilbertsorting only (b) Hilbertsorting with 1-Shift

Figure 3.15: Path structure resulting from Hilbertsorting and subsequent 1-Shift on the kroA200 based PTSP instance with $p = 0.5$.

The resulting solution quality of applying Hilbertsorting with 1-Shift as well as the performance of the individual heuristic guidance schemes for ACO are displayed in Table 3.2; the best result for each instance is marked bold. For the ACO variants, the number in parentheses is the standard error, while for Hilbertsorting with 1-Shift, it represents the number of 1-Shifts that were performed until no further improvement was found.

For all but four of the test instances, ACO with the Depth-based heuristic achieves the best solution quality, by a significant margin in most cases. The strongly heterogeneous instances seem to be the most difficult ones for ACO, despite the improvements to the original Depth-based Heuristic in [Branke and Guntsch, 2003], without which the performance is roughly 5% ($p \in [0.3, 0.7]$)

Table 3.2: Comparison of results for PTSP instances.

PTSP instance	TSP-ACO	Depth-ACO	Angle-ACO $c = 0.8$	Hilbertsorting with 1-Shift
eil101 $p = 0.25$	325.903 ($\pm$0.162)	**322.023** ($\pm$0.228)	325.071 ($\pm$0.161)	322.233 (43)
eil101 $p = 0.5$	467.441 ($\pm$0.551)	**460.563** ($\pm$0.407)	463.570 ($\pm$0.602)	467.673 (25)
eil101 $p = 0.75$	567.076 ($\pm$0.474)	**564.036** ($\pm$0.546)	564.711 ($\pm$0.486)	570.529 (26)
eil101 $p \in [0.1, 0.9]$	415.978 ($\pm$0.435)	414.708 ($\pm$0.498)	413.301 ($\pm$0.440)	**399.697** (35)
eil101 $p \in [0.3, 0.7]$	450.469 ($\pm$0.515)	**442.225** ($\pm$0.271)	447.056 ($\pm$0.461)	448.203 (30)
kroA200 $p = 0.25$	17816.2 ($\pm$17.973)	**17574.6** ($\pm$10.854)	17719.7 ($\pm$12.785)	18517 (97)
kroA200 $p = 0.5$	23461.2 ($\pm$22.327)	**23327.8** ($\pm$22.849)	23341.5 ($\pm$19.599)	23999.8 (131)
kroA200 $p = 0.75$	27205.6 ($\pm$16.580)	**27126.1** ($\pm$18.076)	27179.9 ($\pm$19.759)	30945.1 (87)
kroA200 $p \in [0.1, 0.9]$	20910.4 ($\pm$23.89)	**20806.4** ($\pm$24.807)	20874.6 ($\pm$27.797)	21468.6 (102)
kroA200 $p \in [0.3, 0.7]$	22675.5 ($\pm$17.248)	**22569.8** ($\pm$20.591)	22610.4 ($\pm$16.654)	24956.3 (93)
rd400 $p = 0.25$	9744.52 ($\pm$17.208)	9327.92 ($\pm$18.063)	9505.89 ($\pm$13.974)	**9042.65** (149)
rd400 $p = 0.5$	12698.1 ($\pm$13.794)	**12257.2** ($\pm$8.713)	12463.14 ($\pm$13.325)	12305.7 (127)
rd400 $p = 0.75$	14619.1 ($\pm$16.059)	**14449.1** ($\pm$10.953)	14455.75 ($\pm$12.825)	14625.4 (116)
rd400 $p \in [0.1, 0.9]$	11547.1 ($\pm$16.602)	11304.2 ($\pm$11.973)	11363.01 ($\pm$13.306)	**10498.7** (156)
rd400 $p \in [0.3, 0.7]$	12370.1 ($\pm$17.664)	12034.6 ($\pm$12.152)	12190.81 ($\pm$13.158)	**11753.4** (140)
rnd100 $p = 0.25$	4.548 ($\pm$0.012)	**4.49** ($\pm$0.012)	4.535 ($\pm$0.012)	4.711 (20.545)
rnd100 $p = 0.5$	6.091 ($\pm$0.016)	**6.044** ($\pm$0.016)	6.076 ($\pm$0.016)	6.551 (17.395)
rnd100 $p = 0.75$	7.097 ($\pm$0.019)	**7.077** ($\pm$0.018)	7.093 ($\pm$0.018)	7.924 (15.215)
rnd100 $p \in [0.1, 0.9]$	5.35 ($\pm$0.019)	**5.308** ($\pm$0.018)	5.319 ($\pm$0.018)	5.902 (19.795)
rnd100 $p \in [0.3, 0.7]$	5.868 ($\pm$0.017)	**5.816** ($\pm$0.016)	5.849 ($\pm$0.016)	6.359 (18.25)

to 15% ($p \in [0.1, 0.9]$) worse. Also, the rd400 instance for both heterogeneous probability distributions and the low homogeneous value of $p = 0.25$ is solved better by Hilbertsorting with 1-Shift. For $p = 0.25$, the reason for this is that after 50000 iterations, ACO was still improving the average solution quality by a significant amount, indicating that it could have reached the same solution quality. Note that well over 100 1-Shift operations are performed after the initial Hilbertsorting for rd400, each of which takes $O(n^4)$ time since $n \cdot (n-1)$ evaluations must be performed to determine which customer to shift, and each evaluation takes $O(n^2)$ time. For the 100 customer instances, the runtime of Hilbertsorting is about equal to the ACO, while on rd400, it is more than 10 times as long, biasing the comparison toward Hilbertsorting with 1-Shift. Without the 1-Shift operation, Hilbertsorting is not competitive.

3.5 Summary and Outlook

We have introduced new ideas for improving the performance of Ant Colony Optimization (ACO) when applied to the Probabilistic Traveling Salesperson Problem (PTSP). First, an approximation of the evaluation function was proposed, motivated by the fact that the full evaluation takes $O(n^2)$ time while an approximation, with the potential for leading to a practically identical solution quality, is computable with $O(n)$ steps. Second, two new methods for deriving heuristic values were proposed with the aim of providing the ants with better guidance during tour construction. These modifications were evaluated empirically on a number of problem instances deemed representative for determining their performance.

The approximation of the evaluation function works very well, with approximation depths of 4 or 8 resulting in virtually the same solution quality with the full evaluation, which corresponds to an approximation depth of $n - 2$ for an instance with n customers. Since these comparatively low values are sufficient, further exploration in this area, e.g. for an adaptive approximation which dynamically changes the approximation depth in order to guarantee a relative promotion error of $\epsilon = 0$, was not undertaken. We believe that, although using the approximative evaluation for ACO is justified due to the large relative savings in runtime, algorithms with a solution construction process that takes $O(n)$ steps will benefit even more from this approximation, since an asymptotic decrease in runtime would be achieved, which is not the case for the ACO algorithm.

The comparison among the new heuristic guidance schemes and with the TSP

Heuristic ended in favor of the Depth-based Heuristic, which outperforms the other two both on a per-iteration basis as well as, to a lesser extent, on a per-second basis. This is not very surprising, since the Depth-based Heuristic more accurately takes into account the effect on solution quality that a choice will have than the TSP heuristic or the Angle-based Heuristic, which to a certain extent is an approximation of the Depth-heuristic limited to a depth of 1. Further attempts to improve the performance of the ACO algorithm with the Depth-based Heuristic did not succeed, and will be the subject of future work.

We feel that the Depth-based Heuristic still has some deficiencies, especially on heterogeneous instances. Nevertheless, without any parameter tuning, the ACO algorithm using this heuristic significantly outperforms one of the previously best known heuristic for PTSP, which is Hilbertsorting with 1-Shift, in all but four cases. For three of these cases, Hilbertsorting with 1-Shift is given 10 times as much runtime as the ACO algorithm for the optimization process. We feel confident that, with further improvements especially on heterogeneous instances which we plan to test in the future, the ACO algorithm using the Depth-based Heuristic will also beat Hilbertsorting with 1-Shift on these remaining instances.

Chapter 4

Dynamic Problems

4.1 Introduction to Dynamic Combinatorial Optimization Problems

In Chapter 3, we discussed modifications of the ACO algorithm to help it deal with a stochastic variant of the TSP called the Probabilistic TSP, in which an *a priori* solution must be constructed and applied later to a reduced instance with no further reoptimization. In contrast to the PTSP, where the goal is to find the solution with best expected quality, this chapter deals with the Dynamic TSP (DTSP), where customers are added and deleted during the optimization process, and the goal of the algorithm is to provide a good average solution quality over time [Guntsch et al., 2000]. Although only the DTSP is discussed, the notion of dynamic problems in which insertions or deletions occur to the set of elements comprising the instance can be applied to other static problems as well to receive the corresponding dynamic counterpart, e.g. the Dynamic Quadratic Assignment Problem, which we introduce in Chapter 5.

In contrast to our definition of DTSP, other authors have characterized separate stochastic variants of the TSP as Dynamic TSP. In [Psaraftis, 1988, 1995], DTSP is defined as the problem of finding an optimal routing policy for a salesman for a number of customers with a demand for service generated randomly by e.g. a Poisson Process. Since these demands become known to the salesman as they are generated and he is already en route, this problem cannot be solved meaningfully with a priori optimization. Another definition is supplied by [Eyckelhof and Snoek, 2002], where traffic jams, i.e. dynamic increases of a distance value, are introduced spontaneously along the currently best solution. Again, since the location of the traffic jam depends on the currently best solution of the algorithm which is probabilistic, the dynamic changes cannot be predicted and a priori optimization cannot be applied. Other problems exist where, like for the PTSP, all information is known in advance, e.g. the Time-Dependent TSP, where edge weights are a function of the time [Malandraki, 1990, Malandraki and Daskin, 1992]. These problems are not dynamic in a strict sense. In fact, due to the availability of all information from the start, they are static problems, and an optimal solution can be constructed a priori with no need to reoptimize.

As stated above, the Dynamic TSP we consider is characterized by the insertion and deletion of customers. Note that these changes to the problem instance do not occur while a solution is being built, but rather between individual solution constructions. The challenge of the problem does not lie in adapting a partially constructed tour to the changes in the instance, but rather to adapt completely finished tours to the modified instance. The central point of interest lies in how quickly an algorithm can move from a good solution on one instance to a good solution on the successor instance.

Some work has been done with the aim of classifying the performance of algorithms on dynamic problems as well as the characteristics of the dynamic changes in the context of Evolutionary Algorithms. In [Weicker, 2002a,b], a number of methods for describing how well an algorithm copes with dynamic changes are defined, including stability, which measures how much the solution quality deteriorates as the result of a change, and ϵ-reactivity, which is used to gauge how quickly an algorithm comes within a relative margin ϵ of the previous level of solution quality. Of course, the values which can be considered good for stability and reactivity depend on the nature of the changes to the problem instance. In [Branke, 1999, Branke and Schmeck, 2002], the changes are characterized by traits like severity, i.e. how large the change to the instance is, and frequency, i.e. how often such a change occurs.

Our approach and evaluation methods are similar to those introduced above. If $f(t)$ is the tourlength at time t, then measuring

$$F(T) = \int_0^T f(t)dt \tag{4.1}$$

for large values of T allows us to gauge the performance of three characteristics of an algorithm used to “solve” the DTSP:

1. How large is the deterioration of solution quality (i.e. how good are the first solutions found) immediately after a change?

2. How long does it take the algorithm to find solutions of comparable quality to the previous instance if both instances are similar?

3. Given a certain frequency and intensity of modifications and an approximately constant level of expected solution quality, how good is the level of solution quality maintained by the algorithm, and how does it develop over time?

The answer to the first question tells us how good the algorithm will perform in the worst case scenario of having to "realize" a tour immediately after a change to the problem instance. The second one gives some indication of the recovery time needed after a change before a good solution is once again available. Lastly, the third question asks whether the algorithm can be run perpetually in a given environment, or if a complete restart becomes unavoidable after some time.
Applications for this version of DTSP are present in scenarios where a constant evolution of the structure of the instance at hand takes place as well as uncertainty about the exact point in time when the optimization process must be stopped (and the best solution available realized) exists. This situation can arise in military as well as crisis management fields, where an outside threat forces the current solution to be realized.
Interestingly, tests on dynamically changing instances of the Shortest Path Problem have already been performed with real biological ants in [Beckers et al., 1992]. Here, the two short branches from the double bridge experiment (see Figure 2.1, p.15) were initially removed from the setup and added after 30 minutes. The ant *Linepithema humile* was unable to reoptimize on the changed instance, staying on the long path marked by the pheromones. However, the species *Lasius Niger*, which was tested in the same fashion, successfully reoriented the path to include both short branches. The strength and different intensities of pheromone trails combined with the ability of *Lasius Niger* to realize when moving almost perpendicular to the direction in which a food source lies (which is the case on the long branches in the double bridge setup) and consequently make a u-turn were identified as contributing factors for this ability to reoptimize.
The contents of this chapter are based upon previous work by the author in [Guntsch et al., 2000, Guntsch and Middendorf, 2000, Guntsch et al., 2001], where ACO is used for the DTSP. In Section 4.2, we cover the possible strategies for a standard ACO algorithm to react to a change in the problem instance. These consist of various methods for modifying the pheromone information

as well as a scheme for repairing the solution enforced by the elitist ant. To better explain what these pheromone modification strategies do, we introduce the concept of pheromone entropy in Section 4.3. An empirical evaluation with the aim of answering the three questions posed above is performed in Sections 4.4 and 4.5. We summarize our findings in Section 4.8, and also discuss possibilities for extensions and future work.

4.2 Reacting to a Change

In this section, we describe how to modify the ACO algorithm so that it exhibits a better performance for the DTSP. The basic algorithm used is Algorithm 3.1, p.34. In contrast to the PTSP, our modifications do not have the aim of improving the solution construction process, which for the Dynamic TSP is the same as for the regular, static TSP. Instead, we focus on ways to modify the pheromone information after a change has occurred to the problem instance, helping the ants in the reoptimization process. When modifying the pheromone information, which serves as a form of memory about the location of good solutions, we must decide how much of and where this information is still exploitable, and how much should be removed in order to enforce exploration.

In order to describe our methods for enforcing exploration, we must first describe the immediate effects a change to the problem instance has on the ACO algorithm. When a customer i is removed from a DTSP instance, the corresponding row i and column i in the pheromone matrix are equally removed, since they correspond to the pheromone information about when to go to customer i and where to leave from customer i respectively. For each customer i added, a new row i and column i are introduced into the matrix with the pheromone values initialized to τ_0. When a change, i.e. insertions and/or deletions, to a problem instance occurs, the ants will need to restructure the tour in order to leave out the deleted customers and include the newly inserted ones. However, the pheromone information on the outgoing edges from the "old" customers might be so strong as to practically deny the inclusion into the tour of the new customers at an opportune place, resulting in the inclusion at the very end of the construction process when no other choices remain. If for example the new customers are evenly distributed in a plane and distances are Euclidean, this will tend to result in a bad solution quality. Hence, it is necessary to reset the pheromone information on the outgoing edges of the old customers to some degree, which we call the reset value γ. Using $\gamma_i \in [0,1]$,

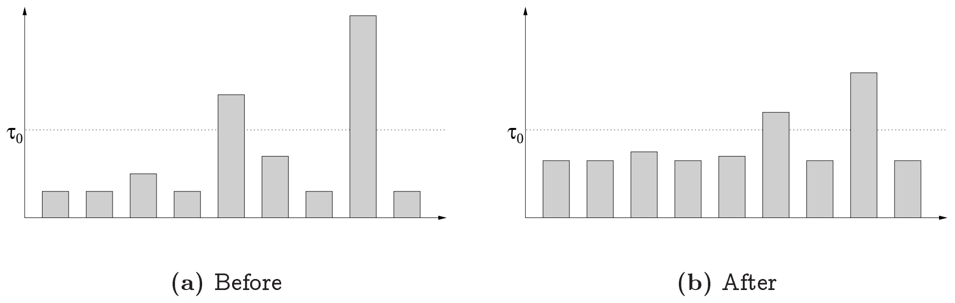


Figure 4.1: Pheromone values on outgoing edges from a customer i (a) before and (b) after a partial reset with $\gamma_i = 0.5$.

a customer i smoothes the pheromone values on outgoing edges to all other customers j via

$$\forall j \neq i : \tau_{ij} \mapsto (1 - \gamma_i) \cdot \tau_{ij} + \gamma_i \cdot \tau_0 \tag{4.2}$$

With the application of Equation 4.2, all pheromone values are reinitialized to a relative extent defined by γ_i, with $\gamma_i = 0$ denoting no change and $\gamma_i = 1$ a complete reset to the initial values. Resetting the entire pheromone matrix to initial values for static problem instances has also been proposed in [Gambardella et al., 1999a, Cordón et al., 2000] to counter stagnation of the search process. Figure 4.1 shows an example of the effect a reset with $\gamma_i = 0.5$ has on the pheromone values on the outgoing edges from a customer i.
When dealing with symmetric problems, it is beneficial to keep the pheromone matrix symmetric as well, see Subsection 3.3.1. If $\tau_{ij} = \tau_{ji}$ but customers i and j are assigned different respective reset values $\gamma_i \neq \gamma_j$, then this symmetry will be disrupted. Since the customers i and j effectively share the edge (i, j), it makes sense to use the average

$$\gamma_{ij} = \frac{\gamma_i + \gamma_j}{2} \tag{4.3}$$

to reset τ_{ij} as well as τ_{ji}.
Having a mechanism to reinforce exploration via a reset value, what remains is to define methods for assigning reset values to all customers in a problem instance once a change has taken place. We call these methods pheromone modification strategies. We separate two fundamental approaches for assigning reset values: global and local. A global assignment of the reset values implies that all customers i receive the same reset values $\gamma_i = \gamma$. This type of assignment can be carried out very fast, especially if γ is not computed but rather a parameter of the respective strategy. In this spirit, we introduce

the Restart-Strategy below in Subsection 4.2.1. However, instead of using a uniform value for the assignment of the reset values, it might also work well to reset more information in close proximity to the location of the insertions or deletions. Hence, a local assignment which takes this proximity into account might be useful. The η-Strategy and the τ-Strategy introduced in Subsections 4.2.2 and 4.2.3 respectively perform a localized assignment with different definitions of proximity to the changes in the problem instance. In 4.2.4, we show that it is also possible to combine global and local strategies. A different approach to the preservation of meaningful information is taken in Subsection 4.2.5, where we introduce a procedure for repairing the solution represented by the elitist ant so that it potentially represents a good solution to the changed instance and can continue to increase the search intensity on a promising portion of the solution space.

4.2.1 Restart-Strategy

The Restart-Strategy is comparatively simple, which is its strength as well as its weakness. When applied, it assigns to each customer the strategy-specific parameter $\lambda_R \in [0, 1]$ as the reset-values, i.e.

$$\forall i : \gamma_i = \lambda_R \tag{4.4}$$

This strategy has the advantage of requiring no computation for assigning the reset-values, and is therefore very fast. On the other hand, neither location nor intensity of the change to the problem instance are taken into account, resulting in an enforcement of exploration in areas of the instance where a good solution already existed. Thus, this strategy runs the risk of forcing the ants to find a previous good solution multiple times, depending on the exact value of λ_R. Since its proposal, this strategy has also been successfully employed for Dynamic Vehicle Routing Problems (DVRPs), see [Montemanni et al., 2003].

4.2.2 η-Strategy

In the η-Strategy, each customer is assigned a reset value proportionate to the closest inserted/deleted customer. For customer i, we define the distance to the old removed and newly added customers j via the heuristic value η_{ij}, which in turn is defined through the actual distance between customers i and

j. In order to avoid any imbalance due to scaling, we calculate the average

$$\eta_0 = \frac{1}{n \cdot (n-1)} \sum_{i} \sum_{j \neq i} \eta_{ij} \tag{4.5}$$

over all heuristic values. Then, using the heuristic values, we define the heuristic distance d^{η}_{ij} between an old customer i left in the instance and a customer $j \in C$, where C is the set of customers that were inserted or removed in the last change to the problem instance, as

$$d^{\eta}_{ij} = 1 - \frac{\eta_0}{\lambda_E \cdot \eta_{ij}} \tag{4.6}$$

with $\lambda_E \in (0, \infty)$. Since we are interested in the closest insertion or deletion, i.e. the maximum value of d^{η}_{ij} for all customers $j \in C$, we define

$$d^{\eta}_{i} = \max_{j \in C} d^{\eta}_{ij} \tag{4.7}$$

The value of d^{η}_{i} is always smaller than 1 by design, so by limiting the minimum value to 0, we receive the reset-value

$$\gamma_i = \max(0, d^{\eta}_{i}) \tag{4.8}$$

This assignment of reset points can be envisioned as a number of cones being created above all customers $j \in C$, with the maximum positive height of a cone above any of the old customers i indicating how high the reset-value γ_i for this customer is. The parameter $\lambda_E \in (0, \infty)$ defines how wide the cone is at its base; for $\lambda_E \to 0$, the cone approaches a width of 0, which is equivalent to not applying the strategy at all, while for $\lambda_E \to \infty$, all pheromone information is reset to τ_0. In Figure 4.2, the η-Strategy with $\lambda_E = 1$ is demonstrated on a 10×10 grid of customer locations, with a customer near the middle being removed.
Note that due to the definition of the heuristic distance directly via the heuristic values, the application of the η-Strategy would not be meaningful for a DTSP where the triangle-inequality does not hold for the distances between customers.

4.2.3 τ-Strategy

Similarly to the η-Strategy, the τ-Strategy also assigns reset-values according to the distance from the closest insertion or removal. However, instead of defining the distance by using the heuristic values, the pheromone values are

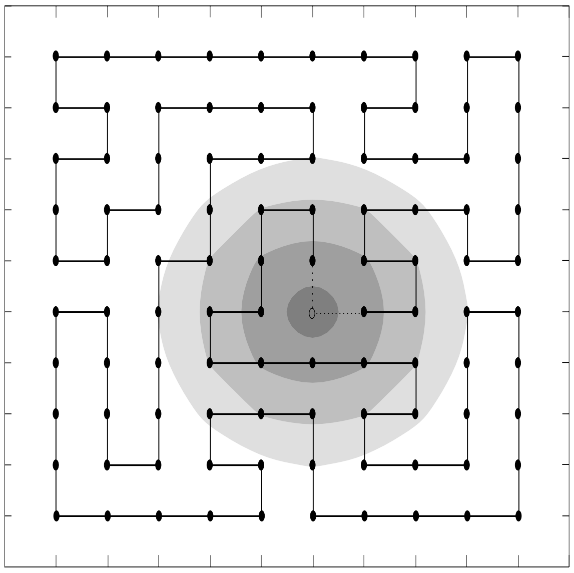


Figure 4.2: A 10×10 grid of customers with coordinates from (0,0) to (9,9) and an optimal path, reset-value distribution according to the η-Strategy upon removal of the customer at (5,4).

used. Specifically, the pheromone distance d^{τ}_{ij} between an old customer i and $j \in C$ is derived from how strong i and j are connected with pheromone values. The pheromone connection of two customers i and j for a given path P_{ij} from i to j is defined as the product of normalized pheromone values on the edges of P_{ij}. Hence, the pheromone distance d^{τ}_{ij} is defined as

$$d^{\tau}_{ij} = \lambda_T \cdot \max_{P_{ij}} \prod_{(u,v)\in P_{ij}} \frac{\tau_{uv}}{\tau_{\max}} \tag{4.9}$$

The maximum over all paths can be calculated efficiently using a modified spanning tree algorithm. To prevent any incompatibility due to absolute sizes, we scale all pheromone values by the maximum value achievable via the update rule. This is $\tau_{\max} = 1$ for asymmetric instances and $\tau_{\max} = 1/2$ for symmetric ones. Using the distance values provided by Equation 4.9, we determine the "closest" change analogously to the η-Strategy, that is

$$d^{\tau}_{i} = \max_{j\in C} d^{\tau}_{ij} \tag{4.10}$$

Since the strategy parameter $\lambda_T \in [0,\infty)$ is used for linear scaling of the pheromone distances, and all distances are greater than 0, distance values greater than 1 must be cut off to calculate the final reset value for a customer

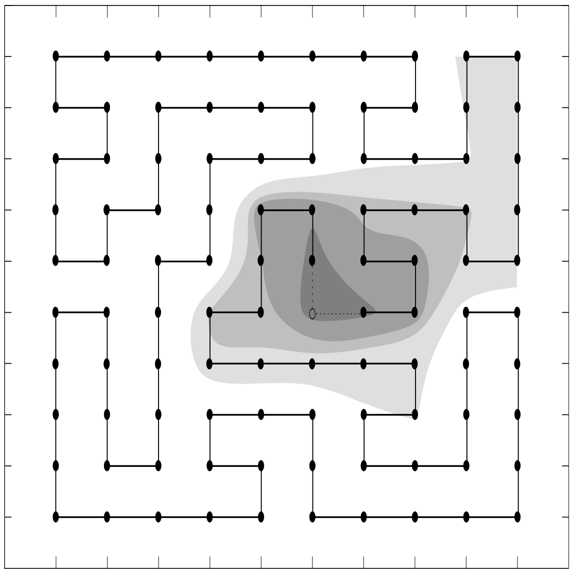


Figure 4.3: A 10×10 grid of customers with coordinates from (0,0) to (9,9) and an optimal path, reset-value distribution according to the τ-Strategy upon removal of the customer at (5,4).

i, i.e.

$$\gamma_i = \min(1, d_i^{\tau}) \tag{4.11}$$

The τ-Strategy can be applied meaningfully to both symmetric and asymmetric problem instances. However, recall that inserted customers have pheromone values set to τ_0, which can be on the order of magnitude of $1/n$, where n is the size of the instance (see Sections 4.4 and 4.5). This would result in a very limited range of values, and a linear scaling would result in a complete reset, which is why, for applying the τ-Strategy, we set the pheromone values from an inserted customer i to the two closest neighbors j_1 and j_2 to 1, i.e. $\tau_{ij_1} = 1$ and $\tau_{ij_2} = 1$ if $d_{ij_1} \leq d_{ij_2}$ and $\forall k \neq j_1 : d_{ij_2} \leq d_{ik}$. This setting is temporary, and all pheromone values of inserted customers are reset to τ_0 after the τ-Strategy has been applied. Figure 4.3 shows an application of the τ-Strategy for a single deletion with $\lambda_T = 1$.

4.2.4 Combinations

A combination of the global Restart-Strategy with one of the two more locally acting η- or τ-Strategies could be advantageous in a situation where strong local resetting near the inserted/deleted customers is necessary to incorporate a change while a lower global resetting is needed to maintain the flexibility for the ACO algorithm to change the best tour found more strongly if beneficial. This combination can be realized by having each of the two strategies involved distribute reset-values according to their respective scheme and then choosing for each customer i the maximum of the two reset-values determined by the two strategies. Formally, if the first strategy distributes γ_{i1} and the second γ_{i2}, then $\gamma_i = \max\{\gamma_{i1}, \gamma_{i2}\}$.

4.2.5 Repairing the Elitist Ant

Whenever a change to the instance that the algorithm is running on occurs, the elitist ant, which enforces the best solution found so far, no longer represents a valid solution. Consequently, it must be dropped and a new elitist ant is determined after the first iteration of ants has worked on the changed instance. The possible loss of information from dropping the old best solution can be alleviated somewhat by modifying the former best tour so that it once again yields a valid and presumably good solution to the changed instance. We use two greedy heuristics for performing this modification:

- all customers that were deleted from the instance are also deleted from the old best tour, effectively connecting their respective predecessors and successors,
- the customers that were added are inserted individually into the tour at the place where they cause the minimum increase in length.

The tour derived from this process is the new tour of the elitist ant. We call this method KeepElitist. Clearly, this modification can be combined with the pheromone modification strategies explained above.

4.3 Entropy

During the later empirical evaluation, we apply the pheromone modification strategies proposed in Section 4.2. Since these strategies reset the pheromone values to some extent, we are interested in how much pheromone is actually reset, i.e. how much "freedom" the ants actually have when constructing a tour, especially after a change. A well known measure for the amount of freedom in a probability distribution $p_1, \dots, p_n$ is its entropy H, defined in [Shannon, 1948] as

$$H = -\sum_{i=1}^{n} p_i \cdot \log(p_i) \tag{4.12}$$

Note that $H \in [0, \log(n)]$. By design we always have $\forall i : \sum_{j=0}^{n-1} \tau_{ij} = 1$ for the empirical evaluation in this section. This intuitively allows us to use the values in the pheromone matrix to compute what we call the pheromone entropy H^τ. The pheromone entropy of the matrix is calculated as the average over the scaled pheromone entropies $H_i^\tau \in [0, 1]$ in the individual rows, where the scaling is used to make H_i^τ independent on n. Formally,

$$\begin{aligned} H_i^\tau &= \frac{1}{\log(n)} \sum_{j \neq i} -\tau_{ij} \cdot \log(\tau_{ij} \qquad (4.13) \\ H^\tau &= \frac{1}{n} \sum_{i=0}^{n-1} H_i^\tau \\ &= \frac{1}{n \cdot \log(n)} \sum_{i=0}^{n-1} \sum_{j \neq i} -\tau_{ij} \cdot \log(\tau_{ij}) \end{aligned}$$

The pheromone entropy is highest, i.e. $H^\tau = 1$, when the pheromone values are equally distributed, e.g. when all matrix elements are set to initial values. A lower value for H^τ signifies less freedom of decision for the ants when constructing a tour, with $H^\tau = 0$ indicating that exactly one pheromone value in each row is 1, and all others 0, which would mean complete convergence. The actual entropy in the decision process as perceived by the ants was introduced by [Merkle et al., 2000a, 2002], while pheromone entropy in the above form has previously been used in [Guntsch and Middendorf, 2000, Guntsch et al., 2001]. Note that the pheromone entropy does not correspond exactly to the entropy of the probability distributions perceived during tour construction for two reasons. First, since we utilize the information about the distance between two customers during tour construction, we use Equation 2.2, p.19, which includes the heuristic values, instead of Equation 2.1, p.18. Second, when the probability distribution over the selection set S is constructed, some choices will usually be blocked, resulting in a bias. Nonetheless, we use the entropy in the pheromone matrix, since it is independent of the actual tour an ant constructs, and it allows us to more accurately gauge the effects of the pheromone modification strategies, which work solely on the values in the pheromone matrix.

4.4 Evaluation of a Single Insertion or Deletion

In this section, we empirically evaluate the performance of the respective pheromone modification strategies for scenarios in which only a single insertion or deletion is conducted. Later, in Section 4.5, scenarios with a continuous insertion and deletion are studied.

4.4.1 Test Setup

We conduct only a single insertion or deletion of one customer at two different points in time on the Euclidean instance `eil101` from the [TSPLIB, 2003], yielding four test scenarios:

del250: delete a customer after 250 iterations

del500: delete a customer after 500 iterations

ins250: reinsert a customer after 250 iterations

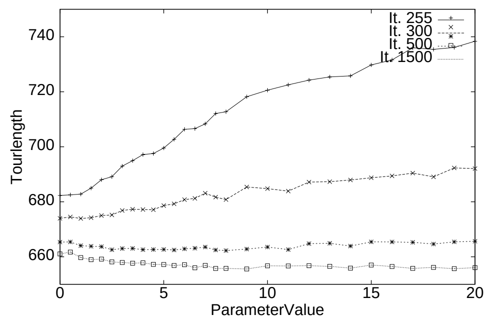
ins500: reinsert a customer after 500 iterations

For the two insertion scenarios, a customer is removed before the start of the algorithm and later reinserted. For all `eil101` scenarios, we run the algorithm for 1500 iterations. In order to remove any bias due to the location of the customers inserted or deleted, the results presented in the following Subsection 4.4.2 for the individual scenarios are averages over the respective insertion or deletion of all 101 customers comprising the instance. Using these four scenarios, we investigate the immediate effect of the respective strategies on solution quality. To this end, we also explore the influence of the parameter setting of each individual strategy.
The ACO algorithm uses the parameters $m = 10$ ants, $\alpha = 1$, $\beta = 5$, $q_0 = 0.5$, and $\rho = 0.01$. We also performed all tests with $\beta = 1$ and $q_0 \in \{0.0, 0.9\}$, with equivalent or worse performance. The elitist ant was dropped when the insertion/deletion occurred, and redetermined in the first iteration thereafter. As an initial pheromone value, we use $\tau_0 = 1/(n-1)$, which, using the update rule defined in Algorithm 3.2, p.35, keeps the row/column sum of pheromone values at a constant value of 1, which is necessary for calculating the pheromone entropy H^τ via Equation **??**.

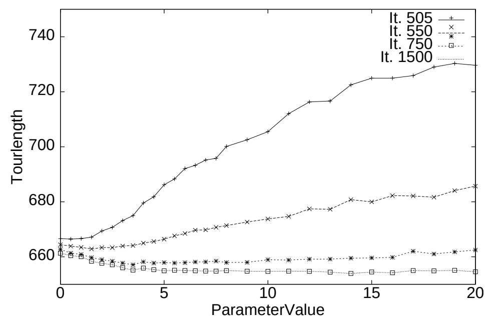
4.4.2 Results

In this subsection we analyze the use of the three main strategies, i.e. the Restart-, η- and τ-Strategy, on the scenarios del250, del500, ins250 and ins500. Since the results for deletion and insertion were quite similar, we will use only the respective deletion scenarios to support our arguments in the following.

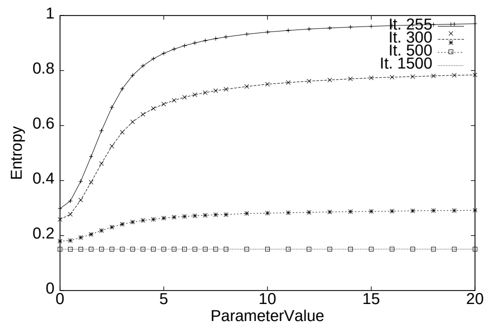
For all three strategies, we tested the influence of the λ parameters (i.e. λ_R, λ_E, λ_T) which influence the height of the reset-values. The Figures 4.4, 4.5, and 4.6 all show the influence of the strategy parameter λ on solution quality and entropy at 4 discrete points, each corresponding to one of the curves, which are located 5, 50, and 250 iterations after the deletion has occurred, and the final result after 1500 iterations.



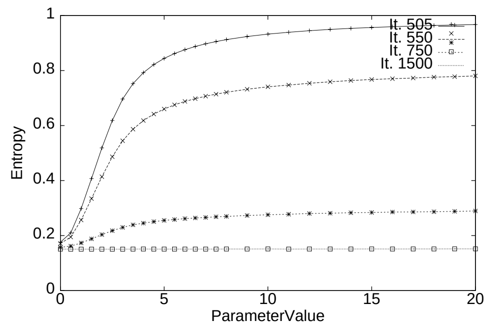
(a) del250, best solution



(b) del500, best solution



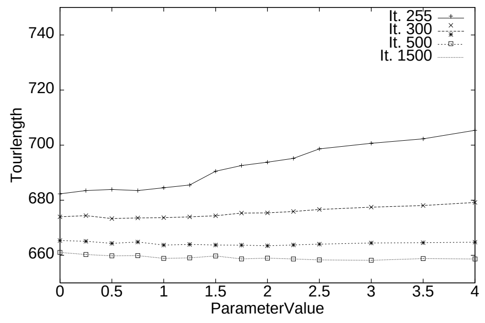
(c) del250, entropy



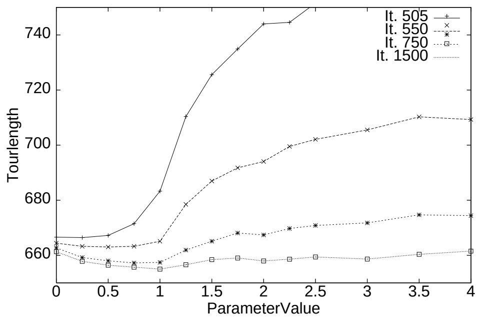
(d) del500, entropy

Figure 4.4: Best results and entropy of η-Strategy with $\lambda_E \in [0, 20]$ on del250, del500.

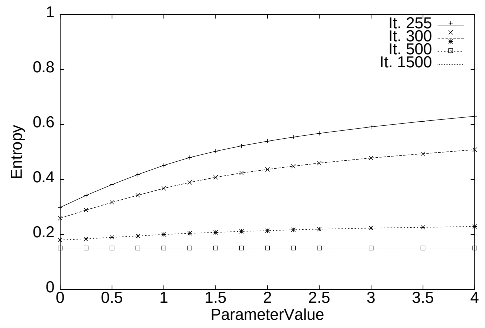
Figure 4.4 shows the results obtained by the η-Strategy for parameter-values of $\lambda_E \in [0, 20]$ in 0.5 to 1.0 increments. As can be seen, a higher value of λ_E entails a worse starting solution and a better final solution, the latter however holding only for $\lambda \leq 8$, after which no significant difference between the final tourlengths exists. This suggests that "good" values for λ_E are not too large. The respective entropy-curves asymptotically approach their maximum value of 1 after the turning point around $\lambda_E = 2$. In conformity with the development of the best solution, the difference in entropy for $\lambda_E > 8$ is only marginal.



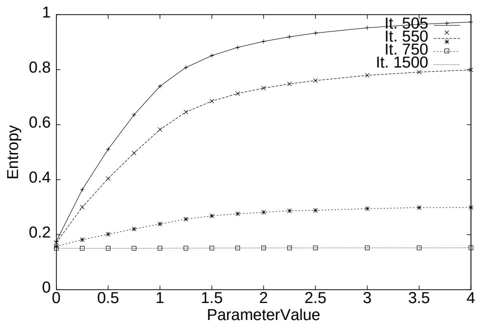
(a) del250, best solution



(b) del500, best solution



(c) del250, entropy



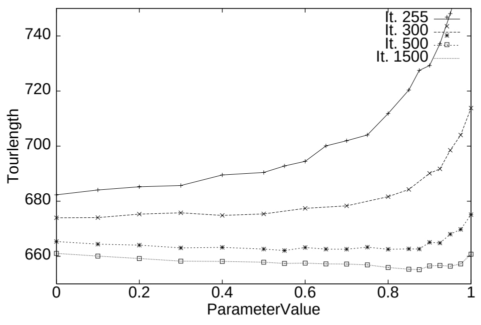
(d) del500, entropy

Figure 4.5: Best results and entropy of τ-Strategy with $\lambda_T \in [0, 4]$ on del250, del500.

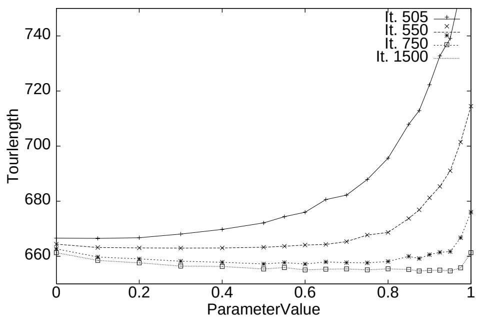
The results for the τ-Strategy with $\lambda_T \in [0, 4]$, in increments of 0.25 to 0.5, are shown in Figure 4.5. The results for a deletion after 250 iterations are somewhat similar to those for the η-Strategy [1], but a deletion after 500 iterations leads to different results. This is due to the mechanism of pheromone-based distances that is used by the τ-Strategy, which leads to $\frac{\tau_{ij}}{\tau_{max}} \to 1$ and there-

[1] This is the only place where a case of insertion differs from deletion, since when inserting with the τ-Strategy, the value τ_{max} is used, which is substantially higher than regular values in the pheromone-matrix at lower iterations. Therefore, the results for ins250 look somewhat more like those for del500 than those for del250.

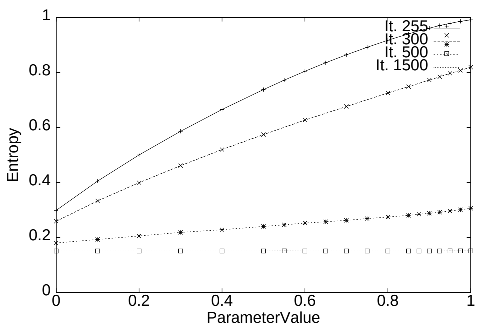
fore $d^{\tau}_{ij} \to 1$ for advanced iterations. It seems that here the τ-Strategy is quite efficient when $\lambda_T \approx 1$. In this case not too much pheromone is reset so that the results are good even at iteration 505, but enough to be comparable to a total reinitialization (see Figure 4.6) after 1500 iterations.



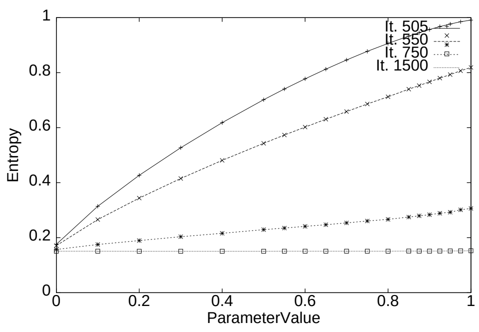
(a) del250, best solution



(b) del500, best solution



(c) del250, entropy



(d) del500, entropy

Figure 4.6: Best results and entropy of Restart-Strategy with $\lambda_R \in [0, 1]$ on del250, del500.

The results for the Restart-Strategy are shown in Figure 4.6, with $\lambda_R \in [0, 1]$ in 0.025 to 0.1 increments. Similarly to the η-Strategy, the two cases del250 and del500 look almost identical , with a slightly higher entropy- and worse solution-level for del250. For $\lambda_R \leq 0.5$ the solution quality is roughly the same as before the change 50 iterations after the deletion has occurred. For higher λ_R the trade-off of worse solution quality in the beginning for a better solution quality in the end holds up to $\lambda_r \approx 0.9$, after which virtually all information is reset and the ant algorithm needs more time to "rediscover" a good solution.

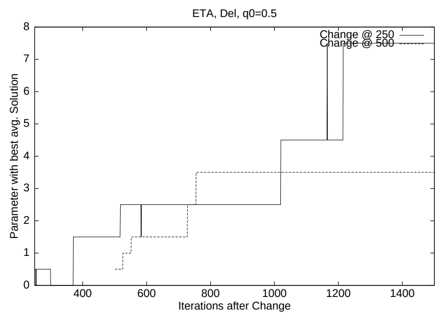
Now, the goal is to find an optimal parameter for each of the strategies and compare their performance. However, determining which parameter is best is not possible without knowledge of how many iterations after the change the best found solution is needed. Figure 4.7 (a)-(c) shows which parameter for each individual strategy resulted in the best average solution over the indicated number of iterations after the deletion. If we assume that the probability for needing the new best solution at a specific iteration is equal for all iterations between the change and iteration 1500, then we only need the rightmost value in each of these subfigures to determine the best λ-setting (i.e. for del250 $\lambda_E = 7.5$, $\lambda_T = 1.0$, and $\lambda_R = 0.85$).
Subfigure 4.7 (d) shows the curve for each strategy with their respective optimal parameter for del250. As can be seen, the τ-Strategy performs best immediately after the change while the η- and Restart-strategy sacrifice good immediate performance for a better solution quality toward the end. This fact also results in the latter two strategies having a slightly better average solution quality over the 1250 iterations considered (η-Strategy : 660.396 and Restart-Strategy : 660.316 vs. τ-Strategy : 661.674).

4.5 Evaluation of Continuous Insertion and Deletion

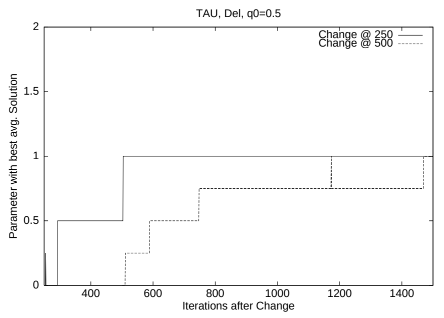
After having analyzed in detail the principal effect on solution quality that the various individual pheromone modification strategies proposed in Section 4.2 have upon insertion or deletion of a customer, this section deals with the performance on a perpetually evolving instance.

4.6 Test Setup

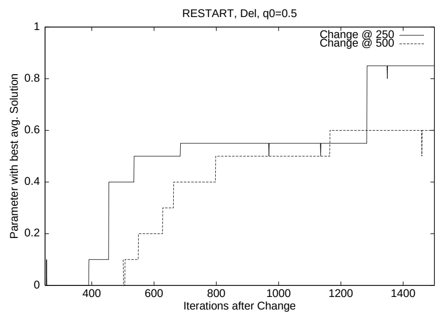
For this empirical evaluation, we use subsets of the Euclidean instance rd400 from the [TSPLIB, 2003]. Specifically, 200 random customers are taken away from the 400 making up the problem instance to form a spare pool of customers before the start of the algorithm, leaving the instance with 200 customers. During the run of the algorithm the actual problem instance is changed every t iterations by exchanging c customers between the actual instance and the spare pool, i.e. c customers are deleted from the actual instance and the same number of customers from the spare pool are inserted. When deciding which customers to delete, the first customer j is chosen at random and all other customers i according to a probability distribution defined by η_{ij}^{p}, with p being a parameter that determines the relative influence of the distances



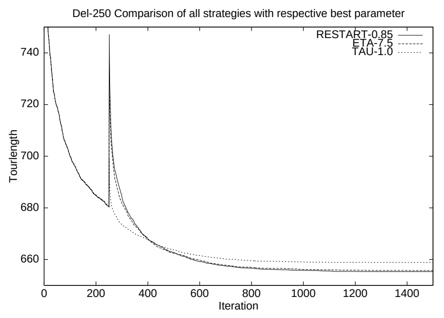
(a) η-Strategy



(b) τ-Strategy



(c) Restart-Strategy



(d) "best" parameters

Figure 4.7: Figures (a)-(c) show which parameter had the best average solution how many iterations after the deletion, (d) compares the three strategies for their respective best parameter λ for del250.

between i and j. The customers that are inserted are chosen analogously from the spare pool. An example is shown in Figure 4.8.

We test all combinations of parameter values $c \in \{1, 5, 25\}$, $t \in \{50, 200, 750\}$, and $p \in \{0.0, 2.0\}$, yielding a total of 18 different scenarios. Note that for $c = 1$, the parameter p has no effect as only one customer is removed/inserted, and that for $p = 0.0$, all customers are chosen with equal probability. For each configuration (c, t, p), 10 test runs of 8999 iterations are performed (in iteration 9000, the next change would occur for all tested t), each starting with a different random subset of 200 customers. All results that are used as a basis for comparison are averages over these 10 runs. Only the results during iterations 3000-8999 are used to measure the performance of the applied strategies, since the behavior of the ACO algorithm during the first iterations is not represen-

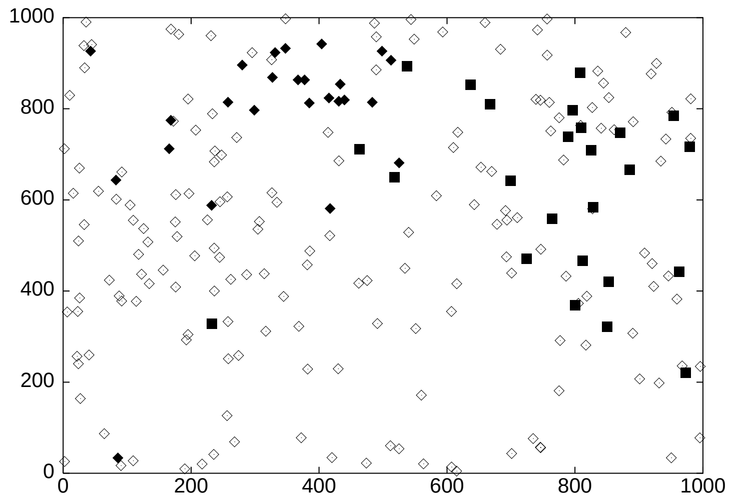


Figure 4.8: 200 customer subset of rd400 with $c = 25$ deleted and inserted customers determined with $p = 2$.

tative for the latter stages. On the basis of this test setup, we evaluate the "longterm" ability of the ACO algorithm to adapt to a periodically changing instance.

The parameter values for the ACO algorithm used in the tests are $m = 10$ ants, $\alpha = 1$, $\beta = 5$, $q_0 = 0.9$, $\rho = 0.05$, and $\tau_0 = 1/(n-1)$. For the pheromone modification strategies, we test the parameters $\lambda_R \in \{0.25, 0.5, 0.75, 1.0\}$ for the Restart-Strategy, $\lambda_E \in \{0.5, 1.0, 2.0, 5.0\}$ for the η-Strategy, and $\lambda_T \in \{0.5, 1.0, 1.5, 2.0\}$ for the τ-Strategy. A parameter value of 0.0, which is equivalent for all strategies and corresponds to not applying the strategy at all, is also tested. Furthermore, we combine the Restart-Strategy with $\lambda_R \in \{0.25, 0.5\}$ with the η- and τ-Strategies, using their respective parameter-values above, in order to determine if such a combination can yield better results than the "pure" strategies by itself. Finally, all of the above settings are tested with and without keeping a modified elite ant as described in Section 4.2.5 after the exchange of customers.

4.7 Results

The results of the continuous insertion and deletion over 6000 iterations is shown in Figures 4.9 for $p = 0$ and 4.10 for $p = 2$. In these figures, the average solution quality over the 6000 iterations in relation to the performance of the other strategies for each configuration is displayed, i.e. a lighter shade indicates a good performance of the strategy-parameter combination for the implied configuration in relation to all other combinations, and a darker shade stands for a bad performance.

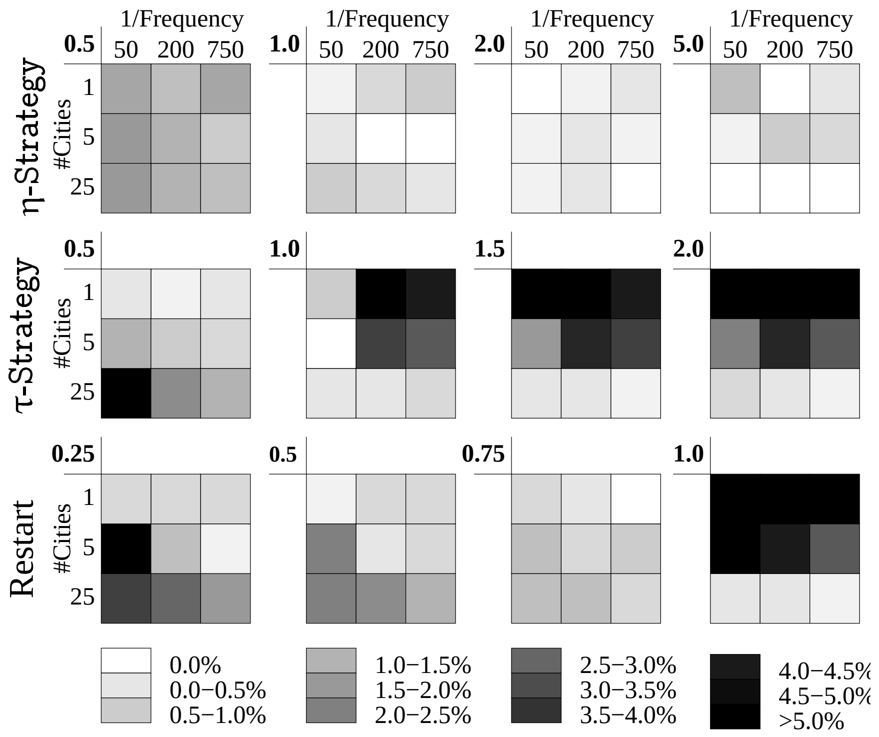


Figure 4.9: Relative performance of Restart-, η- and τ-strategy for $p = 0$ and different values of c and t: loss in quality of the best found solution averaged over iterations 3000-8999 compared to the best performing variant.

Judging from "average darkness", the best overall strategy is the the η-Strategy with a parameter $\lambda_E = 2.0$, especially for a high degree of proximity for the customers inserted and removed. The τ-Strategy with $\lambda_T = 1.0$ provides good to very good solutions when changes occur quickly, i.e. for $t = 50$. The Restart-Strategy, when given enough time and not confronted with changes that are too severe, is also able to achieve good solutions for $\lambda_R = 0.75$. A complete restart, i.e. using the Restart-Strategy with $\lambda_R = 1.0$, is only comparable to the other strategies for the cases where many customers are exchanged, even beating some of the other strategies when they do not reset enough information. This would likely increase if even more customers were transferred as the changed problems would become almost independent of one another.

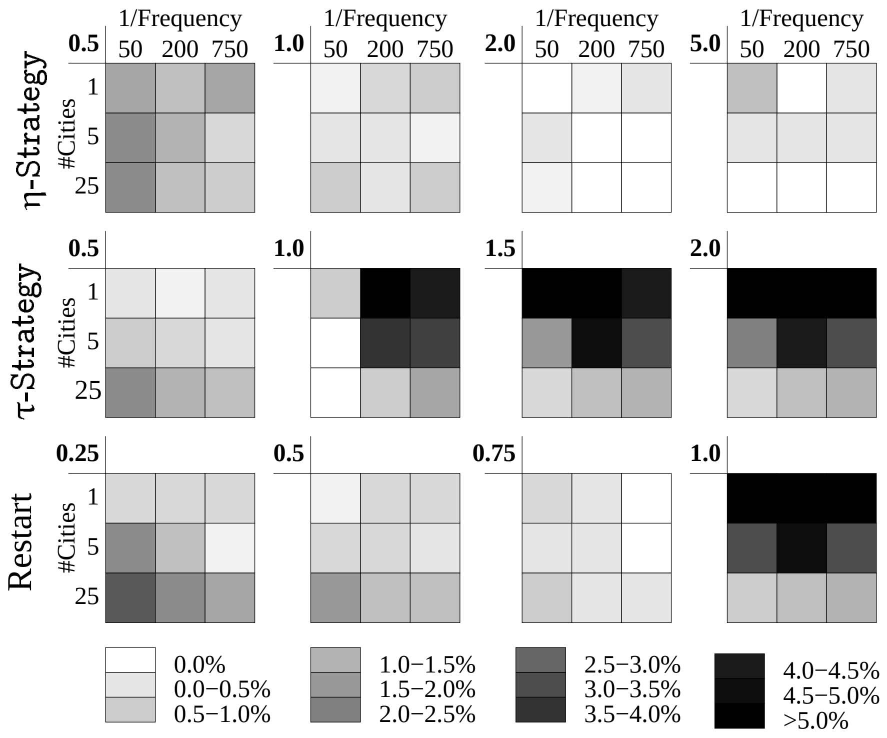
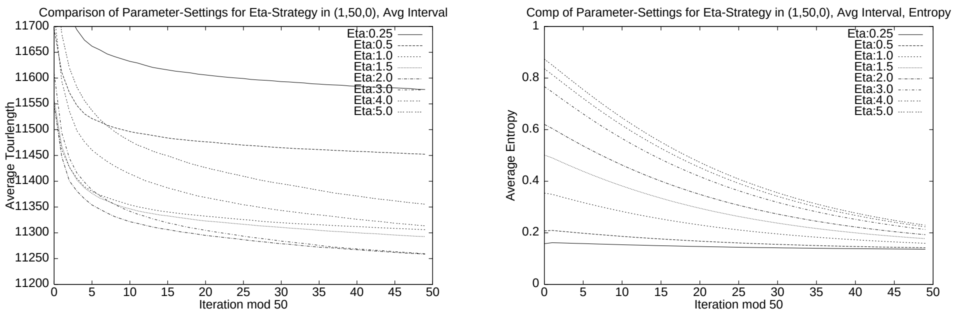


Figure 4.10: Relative performance of Restart-, η- and τ-strategy for $p = 2$ and different values of c and t: loss in quality of the best found solution averaged over iterations 3000-8999 compared to the best performing variant.

As for the influence of the proximity-value p, it seems that the difference in the solution-quality achieved by the individual strategies becomes less for $p = 2$ compared to $p = 0$. For the local strategies, a stronger proximity of the exchanged customers is beneficial because a cluster of customers being inserted or deleted will cause a distribution of reset-values that is not as much dependent on the number of customers comprising the cluster as on their position in the graph or their degree of connectivity in the pheromone matrix. Therefore, although the transfer of customers might be large, the local confinement of this change makes it easier to incorporate for the local strategies. The Restart-Strategy, however, also benefits from a higher degree of proximity. This is probably due to the fact that the misleading pheromone information is more centralized than for the case of equal distribution, and therefore easier to deal with for the ACO algorithm.

In the following, we use one of the configurations to highlight the differences between the separate pheromone modification strategies. The Figures 4.11, 4.12, and 4.13 show a detailed view of the optimization behavior for the individual strategies and its dependency on their respective λ-parameters (λ_E, λ_T, λ_R) for the case of $(c, t, p) = (1, 50, 0)$, i.e. frequently occurring small changes.



(a) Solution Quality **(b)** Entropy

Figure 4.11: Average performance of the η-Strategy for different parameter values on the configuration $(c, t, p) = (1, 50, 0)$ over the t interval. Setting $\lambda_E = 0$ performs poorly, the values being out of range and hence not shown.

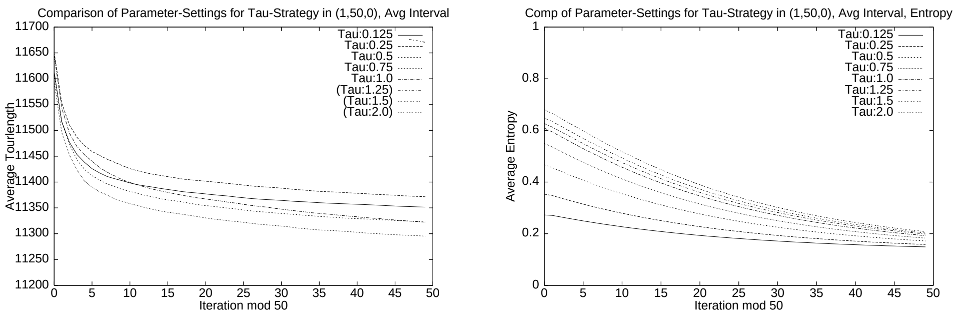
The η-Strategy only slowly becomes worse in terms of solution quality, despite resetting a lot of pheromone for high values of λ_E, as is indicated by the entropy-curves in Figure 4.11. This implies that a large part of the information left intact by the η-Strategy is beneficial in quickly finding a new good path.

In contrast, the τ-Strategy depicted in Figure 4.12 shows a significant loss of performance for $\lambda_T > 1$, even though the entropy curve indicates that the increase in reset information is only moderate.

For the Restart-Strategy, not resetting enough as well as resetting too much information has a negative effect on the entailed solution quality.

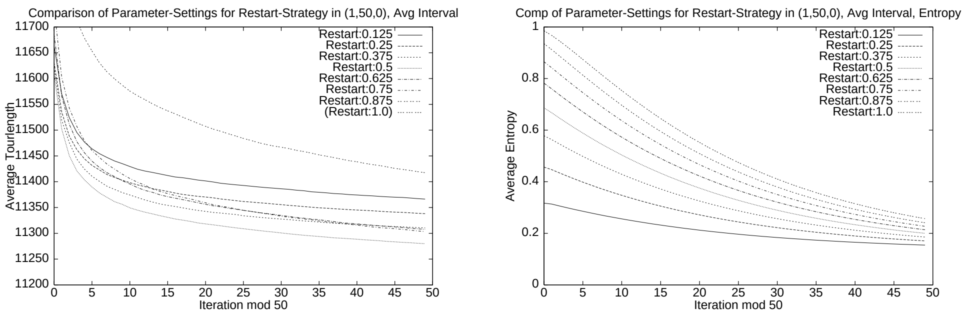
For all strategies, the biggest performance gain occurs when going from performing no resetting at all, which means using a parameter-value of 0.0, to performing even just a little, i.e. setting $\lambda_E = 0.25$, $\lambda_T = 0.125$, or $\lambda_R = 0.125$. The curves for the Restart-Strategy also show that the difference from resetting almost all pheromone information to actually resetting all of it is enormous in terms of solution quality when using this strategy.

As mentioned in the Test Setup, we are also interested in the performance of combinations of the local η- and τ-Strategies with the Restart-Strategy. For



(a) Solution Quality **(b)** Entropy

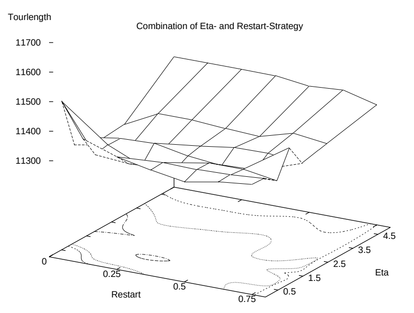
Figure 4.12: Average performance of the τ-Strategy for different parameter values on the configuration $(c, t, p) = (1, 50, 0)$ over the t interval. Setting $\lambda_T = 0$ performs poorly, the values being out of range and hence not shown.



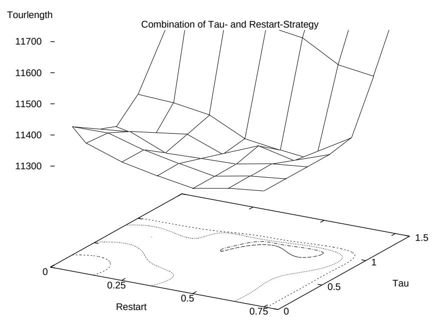
(a) Solution Quality **(b)** Entropy

Figure 4.13: Average performance of the Restart-Strategy for different parameter values on the configuration $(c, t, p) = (1, 50, 0)$ over the t interval. Setting $\lambda_R = 0$ performs poorly, the values being out of range and hence not shown.

some cases, this combination provides better solutions than any of the strategies could achieve by itself. An example of this is once again the configuration $(c, t, p) = (1, 50, 0)$ shown in Figure 4.14, for which we include additional parameter-tests to make a more precise analysis. The contour lines for the combination of the η- and Restart-Strategy show that there are two areas in which good performance was achieved, one of them a true combination with $\lambda_E = 1$ and $\lambda_R = 0.25$, and the other one, which is better in terms of solution



(a) η-Restart Combination

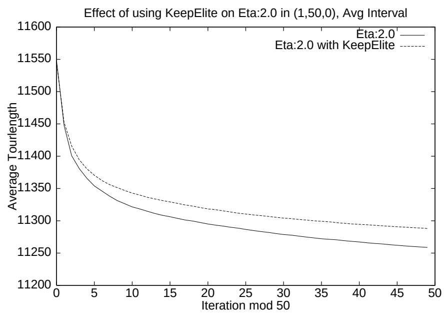


(b) τ-Restart Combination

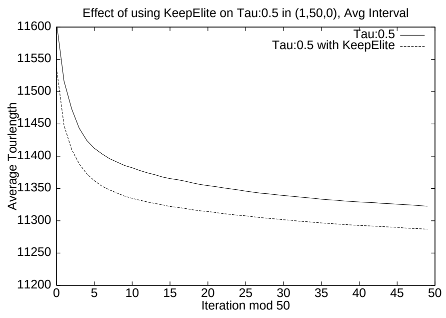
Figure 4.14: Combinations of the local η- and τ-Strategies with the Restart-Strategy for the configuration $(c,t,p) = (1,50,0)$.

quality as well as larger, with $\lambda_E \in \{2,3\}$ and $\lambda_R = 0$. This suggests that the η-Strategy does not benefit much, if at all, from being combined with Restart. For the τ-Strategy on the other hand we see a promising area located around a combination of medium λ_T and λ_R values, specifically for $\lambda_T = 1.0$ and $\lambda_R \in \{0.5, 0.675\}$, and also for $\lambda_T = 0.75$ and $\lambda_R = 0.375$. Thus the combination of the τ- and Restart-Strategy performs better than either strategy by itself, and also better than the η-Strategy for this configuration, justifying its application.

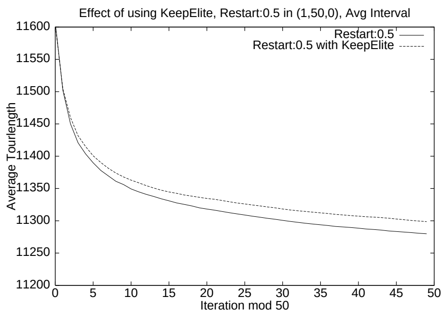
Finally, we combine the KeepElitist method with the individual strategies as well as the combinations of the η- and τ-Strategies with the Restart-Strategy. Figure 4.15 shows how this modified the average behavior of the pure strategies with their respectively best λ-parameter on configuration $(c,t,p) = (1,50,0)$. As can be observed, for the η- and Restart-Strategy the combination on average entailed a worse solution, while for the τ-Strategy the effect on average was an improvement. Overall, the heuristic of keeping a modified elite ant was beneficial only when the number of customers c that was inserted and deleted was not too large and when the time for adapting to the problem t was small. If too many customers were exchanged, then the heuristic would no longer provide a good solution, and the ants would find a better solution in the first iteration after the change. This case is not dangerous, since keeping the modified elitist ant would be the same as not keeping it; only the "new" elitist ant would update the pheromone matrix. The second case in which keeping an elitist ant does not entail better solutions is when the interval t between



(a) η-Strategy



(b) τ-Strategy



(c) Restart-Strategy

Figure 4.15: Combination of the individual single strategies with Keep-Elitist for the configuration $(c, t, p) = (1, 50, 0)$.

changes is long enough to permit the algorithm to adapt very well to the new instance, and the guidance provided by an early good solution leads toward stagnation in the end. This case is potentially dangerous, as the elitist ant survives the first generation(s) and influences the pheromone matrix, thereby restricting the search space to a region that is perhaps not very promising.

4.8 Summary and Outlook

In this chapter, we have introduced the Dynamic TSP, and analyzed the performance of the ACO algorithm for this optimization problem. Specifically, we proposed three pheromone modification strategies to partially reinitialize the pheromone values, depending on the strategy parameter and, for the local strategies, the distance from the dynamic changes in the problem instance.

Furthermore, combining these strategies and keeping a repaired elitist ant were also studied.

The main focus in evaluating these strategies was the average performance over time, with the other two points of interest, namely the loss in solution quality immediately after a change and the rate of recovery, also being discussed briefly. Since the average solution quality is closely correlated to these latter two points, this focus is justified.

Overall, the η-Strategy shows the most consistent good performance for the scenarios considered, in single as well as continuous change environments. This is not very surprising, since this strategy assigns reset values according to the Euclidean distance, which works very well for the Euclidean TSP instances we chose for empirical evaluation. The τ-Strategy, which also works locally, has a much harder time in resetting the relevant information if changes are large or infrequent. This implies that the reset point distribution undertaken by the τ-Strategy is not as effective as that of the η-Strategy. As mentioned above, one reason for this inadequacy seems to be the linear scaling of the reset values, which quickly leads to an "all or nothing" approach for $\lambda_T > 1$ in Equation 4.9. Some preliminary testing with a modified τ-Strategy, called τ'-Strategy, uses the Equation

$$d_{ij}^{\tau'} = \max_{P_{ij}} \prod_{(u,v)\in P_{ij}} \left(\frac{\tau_{uv}}{\tau_{\max}}\right)^{\frac{1}{\lambda_T}} \tag{4.14}$$

for calculating the distances. A value of $\lambda_T = 1$ makes the τ- and τ'-Strategies identical, but for higher values, the τ'-Strategy does a better job of distributing medium reset values to greater Euclidean distances along the path, which is the original intention of this strategy. Preliminary results with this new form of distribution are promising, and we plan to investigate these further in the future.

Once it was established that resetting information performs better than simply letting the ACO algorithm continue with a modified problem instance, the global Restart-Strategy became the main competitor for the local η- and τ-Strategies. The strategy of uniformly resetting pheromone values to some extent performs surprisingly well, considering that unaffected parts of the instance are reset just as strong as those in the immediate vicinity of a change. This might actually be one of the reasons why the Restart-Strategy performs well: it also reoptimizes old parts of the tour after a partial reset, perhaps enabling it to escape a local but not global optimum in this part of the instance. This explanation is also supported by the good performance of the

(τ,Restart)-Strategy combination, which performs better than either of the single strategies and at times better than the otherwise dominating η-Strategy. Another advantage of the Restart-Strategy is that it is immediately applicable to any problem which is dynamic in the same sense as DTSP, since no problem-specific knowledge is needed.

Further future work for evaluating the three single strategies proposed also includes asymmetrical TSP instances. For these, it is necessary to modify the distances used by the η-Strategy, since the triangle inequality need no longer hold, and the shortest path must be determined for the distance values. It is likely that the η-Strategy will no longer perform as well in these scenarios as it was for the Euclidean TSP instances. If an adequate neighborhood can be constructed, it is also possible to apply the DTSP strategies to other, non-TSP problems. For the Dynamic Vehicle Routing Problem, this has already been performed in [Montemanni et al., 2003]. We plan to assess the applicability to other problem classes in the future.

Keeping a repaired elitist ant was good for a marginally better solution in some configurations for the τ- and Restart-Strategies, when changes were small and frequent. The reason for this might be that a single good solution can become quite bad after being repaired to encompass the changes to the problem instance. Instead of only repairing the solution of the elitist ant, it could be beneficial to maintain a population of good solutions from past iterations and apply the repair mechanism to each of them, thereby increasing the likelihood of a good solution being created. This trail of thought is the inspiration for a new type of ACO called Population-based ACO (PACO), which we introduce in Chapter 5.

Chapter 5

Population-based Ant Colony Optimization

5.1 Motivation

In this chapter, we introduce a new type of Ant Algorithm, the Population-based ACO (PACO), which is inspired in part by the DTSP in Chapter 4. This algorithm maintains and updates a population of solutions instead of a pheromone matrix. We evaluate the performance of the PACO on static as well as dynamic problem instances.

The normal ACO algorithm, as described in Chapter 2, updates pheromone values after each iteration and uses evaporation, i.e. multiplication with a factor $(1 - \rho) < 1$, to gradually reduce the influence of previous updates. This implies a strict chronological hierarchy in the importance of updates, meaning that in any given iteration, the largest influence on the behavior of the ants emanates from the updates in the previous iteration, the second largest from the updates two iterations ago, and so forth. However, during the run of an algorithm, these nuances between a series of updates is often not very important. Rather, it is important that new information influences the behavior of the ants to a large degree, regardless of when exactly during the past few iterations it was found, and that old information has only little or no influence during the tour construction. This is especially true in dynamic problem instances which change over time.

In this chapter, we propose to maintain a population of solutions from which the pheromone matrix is derived, and to perform all updates only on the population instead of the matrix. The idea is that every solution in the population is reflected by a corresponding update in the pheromone matrix, which is undertaken when the solution enters the population. This update does not diminish over the following iterations, i.e. no evaporation takes place. Instead, when a solution is removed from the population, a negative update is performed along those pheromone values which previously received a positive update when the solution entered the population. The population size has a constant upper bound, which means that whenever a solution enters a (full) population, another must leave it. Effectively, this results in a one step discretization of the evaporation process: for a number of iterations, a given solution represented by its update to the pheromone matrix has a constant influence on the ants' construction process, and no influence thereafter.

We expect several advantages from this new update scheme. The pheromone values no longer converge to a point where further exploration is impossible, similarly to the *MAX-MIN* Ant System, see [Stützle and Hoos, 1997, 2000]. The pheromone matrix is also more versatile, since its values can change completely in the time it takes to replace the solutions in a population. The discrete nature of the pheromone matrix makes it easier to understand its structure at any time during the run of the algorithm. The updates to the pheromone matrix, consisting of n additions and n subtractions, i.e. $O(n)$ operations, are also faster than for the evaporation case, where each matrix element must be multiplied with $(1-\rho)$, which takes $O(n^2)$ steps for problems which use a quadratic pheromone matrix. Note that other methods with an $O(n)$ update time also exist, e.g. updating by an amount proportional to the quality of the current solution in comparison to the average solution quality over the past few iterations, as proposed in [Maniezzo, 1999].

Another advantage we perceive is that the PACO algorithm is specifically designed to be able to handle dynamic problem instances via a repair function for the individual solutions in the population. This repair function, which we called KeepElitist for the DTSP (see Subsection 4.2.5), modifies a solution for the instance prior to the change, attempting to use the evaluation function in such a way that the modifications result in a valid, high-quality solution. If successful, it implies that the dynamic counterpart of any static problem can be handled by PACO if the problem admits the application of a repair function in the spirit of the KeepElitist strategy.

This chapter is based on previous work by the author in [Guntsch and Middendorf, 2002b,a]. Section 5.2 gives a formal implementation of how the pheromone matrix is derived from the population, and includes different ways of updating the population. In Section 5.3, we test the PACO on static TSPs to ascertain the performance of the new update scheme. Afterwards, in Section

5.4, the PACO algorithm is tested in the environment that originally inspired it, which is Dynamic Combinatorial Optimization Problems (see Chapter 4). At the end of the chapter, we provide a summary and an outlook on future work.

5.2 Implementation

The difference between PACO and standard ACO lies in the memory and update mechanism; the solution construction remains identical to previously given implementations in Chapters 2 and 3.

5.2.1 Population of Solutions

The PACO algorithm starts with an empty population $P = \emptyset$ and a matrix with initialized pheromone values, i.e. $\forall i, j : \tau_{ij} = \tau_0$. At the end of an iteration, the best solution of the iteration π^+ is added to the population, and, in case this exceeds the maximum size k, i.e. if after the update $|P| = k+1$, a solution $\sigma \in P$ is removed. When a solution π is added to P, the pheromone matrix is updated with a value of Δ along the solution, and, conversely, an update of $-\Delta$ is performed along the elements implied by a solution σ that is removed from P. Formally, PACO behaves as indicated in Algorithm 5.1.
It is also possible to derive the complete pheromone matrix from the population, since the following equality holds for the pheromone values at all times:

$$\tau_{ij} = \tau_0 + \Delta \cdot |\{\pi \in P | (i, j) \in \pi\}| \tag{5.1}$$

We denote the maximum possible value an element of the pheromone value can attain in the PACO update scheme as

$$\tau_{max} = \tau_0 + \Delta \cdot k \tag{5.2}$$

if the population size is bounded by k. Hence, all pheromone values are located in the interval $[\tau_0, \tau_{max}]$. If τ_{max} is used as a parameter of the algorithm instead of Δ, we can derive

$$\Delta = \frac{\tau_{max} - \tau_0}{k} \tag{5.3}$$

Note that the actual value used for τ_0 is arbitrary, since τ_{max} could simply be scaled in accordance to achieve the same ratio and hence have the same influence on the decision process. For reasons of clarity, we will continue to use a pheromone initialization τ_0 which results in a row/column sum of 1 in

Algorithm 5.1 PACO algorithm

```
1: initialize pheromone values $\tau_{ij} \mapsto \tau_0$
2: initialize population $P \mapsto \emptyset$
3: repeat
4:    for each ant $i \in \{1, \dots, m\}$ do
5:       initialize selection set $S \mapsto \{0, 1, \dots, n-1\}$
6:       let ant $i$ construct solution $\pi_i$
7:    end for
8:    determine best solution of iteration $\pi^+$
9:    add $\pi^+$ to population $P \mapsto P \cup \{\pi^+\}$
10:   for all $(i, j) \in \pi^+$ do
11:      $\tau_{ij} \mapsto \tau_{ij} + \Delta$
12:   end for
13:   if $|P| > k$ then
14:      remove solution $\sigma$ from population $P \mapsto P \setminus \sigma$
15:      for all $(i, j) \in \sigma$ do
16:         $\tau_{ij} \mapsto \tau_{ij} - \Delta$
17:      end for
18:   end if
19: until condition for termination met
```

the pheromone matrix. As for the regular, evaporation based algorithm, the update rule can also be modified to perform symmetric updates analogously to Algorithm 3.2, p.35, which is used e.g. for Euclidean TSP instances.

So far, we have not described according to which criteria a solution σ is chosen to be removed from the population. The following subsections deal with possible strategies for this removal.

5.2.2 Age-based Strategy

The easiest method for updating the population, which is also the original one used in [Guntsch and Middendorf, 2002b], is an Age-based Strategy in which the eldest solution is removed if the population becomes too large. This strategy insures that each solution has an influence on the construction process over exactly k iterations, after which it is removed from P.

5.2.3 Quality-based Strategy

Instead of keeping the best solution of each of the past k respective iterations in the population, another obvious strategy is to store the best k solutions found over all past iterations. In this Quality-based Strategy, when the population limit is exceeded, the worst solution in the population at that time, including the solution that has just been added, is removed. Specifically, if the best solution of an iteration π^+ is worse than all solutions in the population, then P effectively remains unchanged.

5.2.4 Probabilistic Quality-based Strategy

A disadvantage of the (deterministic) Quality-based Strategy is the possibility that after some iterations, P might consist of k copies of the best solution found so far, which will happen if the best solution is found in k separate iterations. In this case, the ants would be focused on a very small portion of the search space, thus significantly hindering further exploration. To ameliorate this deficiency of the Quality-based Strategy, we introduce a variation called the Prob-based Strategy, which chooses which element from P to remove probabilistically based on the quality of the solutions. Let $f(\pi)$ denote the solution value associated with a solution π and let lower values of f indicate a higher solution quality. With $P = \{\pi_1, \ldots, \pi_{k+1}\}$, we define a probability distribution over the solutions π_i, $i = 1, \ldots, k+1$:

$$
\begin{aligned}
p_i &= \frac{x_i}{\sum_{j=1}^{k+1} x_j} \qquad \text{with} \\
x_i &= f(\pi_i) - \min_{j \in [1,k+1]} f(\pi_j) + avg(\pi) \qquad \text{and} \\
avg(\pi) &= \frac{1}{k+1} \sum_{j=1}^{k+1} f(\pi_j) - \min_{j=1,\ldots,k+1} f(\pi_j)
\end{aligned}
$$

Figure 5.1 illustrates the transformation from the quality of a solution to its probability of being removed from the population. As can be seen, any solution from the population might be removed, and there is a strong preference to remove bad solutions. As with the Quality-based Strategy, if $\pi_j = \pi^+ \in P$ is chosen for removal, P received no effective update.

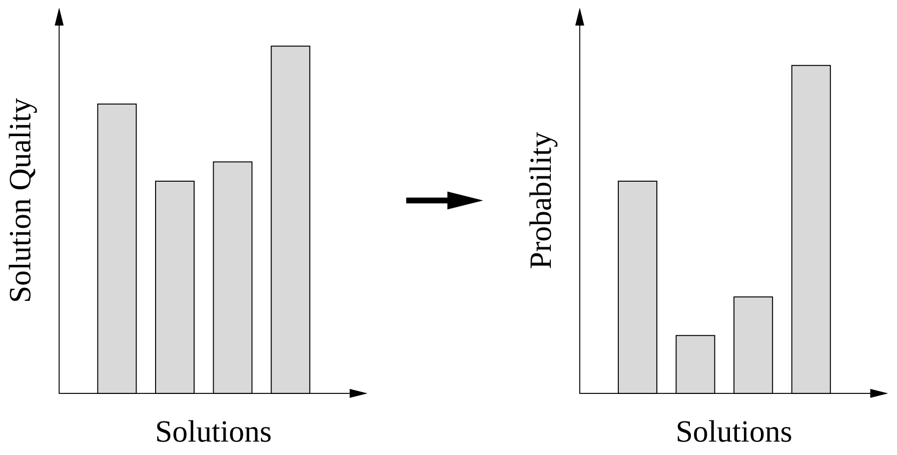


Figure 5.1: Illustration of the transformation from solution quality to probability of elimination from the population.

5.2.5 Age&Prob Combination

It is possible to combine two of the above strategies, the Age-based and the Prob-based one, to form a variation which closely resembles the Prob-based Strategy. In the Age&Prob-based Strategy, we apply the Prob-based Strategy to the population before the solution that would exceed the population limit is added. This removes the possibility of the solution that has just been added being immediately removed again, and therefore guarantees that the new solution will have an influence on the algorithm's behavior for at least one iteration before it can be removed.

5.2.6 Elitism

Finally, we propose a method for employing elitism in the PACO update scheme, which could be necessary since, depending on which strategy is used for the removal of solutions from the population, it is possible that the best solution found is lost after a number of iterations. We define π_e as the elitist solution, i.e. as the globally best solution found by the algorithm so far during this run, which is added to the population when elitism is employed. The elitist solution is not subject to the removal strategies listed above; instead, it can only be replaced by a better solution. When an elitist ant is used by the PACO algorithm, it can be assigned an explicit weight $w_e \in [0, 1]$, which means that the elitist ant updates with a value $\Delta_e = w_e \cdot \tau_{\max}$ and the k normal members of the population each update with $\Delta = (1 - w_e) \cdot \tau_{\max}/k$.

5.3 Evaluation on Static Problems

The aim of this section is to establish the update scheme used in PACO as a viable alternative to other, currently employed methods. To do this, we compare the performance of PACO to that of the standard ACO and to an implementation of the *MAX-MIN* AS to show that the performance achieved by PACO is at least on the same level as that of the other two mentioned algorithms. Note that we do not claim that our algorithm actually works better, which would require some form of proof that the optimal parameter set for each algorithm was tested. Rather, we run each of the algorithms with a number of sensible parameter settings which enable us to draw conclusions about the characteristics of PACO. Before we explain our test setup, we introduce a *MAX-MIN* Ant Algorithm and show how it differs from the standard ACO algorithm. Also, we give a thorough definition of the Quadratic Assignment Problem, which we use as one of our test problems later on.

5.3.1 A MAX-MIN Ant Algorithm

Since the pheromone values in the PACO algorithm are bounded, we want to compare our algorithm with another version of Ant Algorithm which enforces minimum and maximum pheromone values. We call this algorithm *MAX-MIN* ACO, and it is based on the *MAX-MIN* Ant System introduced in [Stützle and Hoos, 1997, 2000]. In our implementation, the *MAX-MIN* ACO algorithm maintains a pheromone matrix exactly like the standard evaporation based ACO algorithm. However, instead of using the immediate pheromone values τ_{ij} to construct the probability according to Equation 2.2, p.19, when constructing a solution, the initial pheromone value τ_0 scaled by a parameter $\varphi > 0$ is added:

$$p_{ij} = \frac{(\tau_{ij} + \varphi \cdot \tau_0)^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum\limits_{h \in S} (\tau_{ih} + \varphi \cdot \tau_0)^{\alpha} \cdot \eta_{ih}^{\beta}} \tag{5.4}$$

The modified pheromone values used by the ants lie in the interval $[\varphi \cdot \tau_0, 1 + \varphi \cdot \tau_0]$, high settings for φ encouraging more exploration and less exploitation to take place.

5.3.2 An ACO Algorithm for the QAP

The Quadratic Assignment Problem (QAP), described in [Koopmans and Beckman, 1957], is an optimization problem in which facilities must be assigned to locations in such a way that the sum of distance-weighted flows over all pairs of facilities is minimized. Formally, given a flow matrix $F = [f_{ij}]$ and a distance matrix $D = [d_{ij}]$, each containing real values, the goal is to find the permutation π which minimizes

$$f(\pi) = \sum_{i} \sum_{j \neq i} f_{ij} \cdot d_{\pi(i)\pi(j)} \tag{5.5}$$

This equation takes $O(n^2)$ steps to evaluate.
As we mentioned in Chapter 2, a node$\times$node representation of the pheromone matrix is inadequate for the QAP, since facilities are assigned to locations and no relationship exists between facilities independent of their locations. Therefore, it makes sense for the algorithm to learn which location a facility was assigned to, leading to a facility$\times$location or, more generally, node$\times$place representation of the pheromone matrix. Algorithm 5.2 shows the solution construction for such problems in which no restrictions exist for the assignment order. Problems where such restrictions exist are usually scheduling problems, where heuristic information is only available if jobs are scheduled consecutively starting at the front. In such cases, instead of choosing a random place, the algorithm should process them from the start and always proceed to the following one.
After the construction process, the solution π contains all allocations of facilities to locations, i.e. the facility j is assigned to place $\pi(j)$. The notation $(i, j) \in \pi$ from Algorithm 2.1, p.24, therefore means $(i, j) \in \{(l, \pi(l)) | l \in [0, n-1]\}$.

5.3.3 Test Setup

Our tests focus on the comparison between the PACO algorithm using the Age-based Strategy and the standard ACO algorithm. We also run our tests with the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant Algorithm described above. As test problems, we use the TSP, which is a well suited problem to test modifications of the ACO metaheuristic since Ant Algorithms are known to perform well on this problem. On the other hand, we also wish to gauge the effect of the limited pheromone information available to the ant algorithm, particularly when no heuristic guidance is used or available. Therefore, we use the Quadratic Assignment Problem (QAP) as a second problem. Specifically, the performance on the following TSP and QAP instances is analyzed:

- Instances from the [TSPLIB, 2003]:

Algorithm 5.2 Solution construction by ant l for node$\times$place problems

```
1: Initialize set of unassigned places $X = \{0, \dots, n-1\}$
2: while $X \neq \emptyset$ do
3:   randomly draw unassigned place $i \in_R X$
4:   update unassigned places $X \mapsto X \setminus \{i\}$
5:   randomly draw $q \in_R [0, 1]$
6:   if $q \leq q_0$ then
7:     choose $j = \arg\max_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ij}^{\beta}$
8:   else
9:     choose $j$ according to probability distribution
         $p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in S} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}}$
10:  end if
11:  remove $j$ from selection set $S \mapsto S \setminus \{j\}$
12:  set $\pi_l(j) = i$
13: end while
```

  - `eil101`, 101 cities, symmetric (Euclidean)
  - `kroA100`, 100 cities, symmetric (Euclidean)
  - `d198`, 198 cities, symmetric (Euclidean)
  - `kro124p`, 100 cities, asymmetric
  - `ftv170`, 170 cities, asymmetric

- Instances from the [QAPLIB, 2003]:
  - `wil50`, size 50, symmetric distances
  - `wil100`, size 100, symmetric distances
  - `tai100a`, size 100, asymmetric
  - `tai100b`, size 100, asymmetric

For the standard ACO algorithm, we couple evaporation rates of $\rho = 0.1, 0.05, 0.02, 0.01, 0.005$ with a maximum number of iterations $t = 2000$, 5000, 10000, 20000, 50000, respectively. These runtimes allow for a form of convergence, meaning no further improvement to solution quality if the algorithm were to continue running, in practically all cases.

As stated above, we use only the Age-based Strategy for deciding which solution to remove from the population when using PACO. The reason for this is that PACO behaves most like the other algorithms when this strategy is used, and our focus is on determining how the discretization and higher mobility of the pheromone values affects algorithmic behavior rather than testing the population update strategies at this point. The different update strategies are compared later in Section 5.4.

The other parameters for the PACO algorithm that are evaluated are $\tau_{max} \in \{1, 3, 10\}$ (note that for the symmetric TSP instances, the actual maximum pheromone value is $\tau_{max}/2$ because of symmetric updating) and population sizes $k = 1, 5, 25$. Since no convergence occurs due to the nature of the update strategy in PACO, all combinations are run for $t = 50000$ iterations.

Similarly to PACO, the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant Algorithm also does not converge and is therefore run for $t = 50000$ iterations. To perform a meaningful comparison, we set the scaling factor φ to such values that the ratio between maximum and minimum attainable pheromone values is the same as for PACO, i.e. we set $\varphi \in \{0.1, 0.3, 1\}$ to achieve $(\varphi \cdot \tau_0)^{-1} = \tau_{max}/\tau_0$. For evaporation rates, we use a spectrum of values $\rho \in \{0.01, 0.05, 0.25\}$.

The other variable parameters tested are $q_0 \in \{0, 0.5, 0.9\}$ and an elitism weight $w_e \in \{0, 0.25, 0.5, 0.75, 1\}$. The elitism weight $w_e \leq 1$ is simply the relative amount of updating performed by the elitist ant, leaving $(1 - w_e)$ for the best ant of the iteration or the population respectively. We introduce this additional parameter to better understand how large an influence elitism should have, particularly for PACO with Age-based removal.

The constant parameters for all three ACO algorithms are $m = 10$ ants per iteration, $\alpha = 1$, and, for the TSP, $\beta = 5$ and $\tau_0 = 1/(n-1)$. For QAP, where we intentionally make no heuristic information available, we effectively have $\beta = 0$ and, since QAP has a job$\times$place encoded matrix where the diagonal is not set to 0, an initial pheromone value of $\tau_0 = 1/n$. The results in the following section are the averages over 10 different random seeds for each combination.

5.3.4 Comparison of PACO with standard ACO

In order to determine whether the PACO algorithm is a competitive alternative to the standard one, we rank the different parameter combinations (210 altogether, 75 for the standard and 135 for the Population-based ACO algorithm) according to their average performance on the individual problem instances. From these instance specific ranks, we take the averages for the two problem classes to generate one class-specific rank for each parameter combination. Of course, this method cannot be used for a detailed comparison of solution quality, but the statement "a better average rank implies at least as good as a performance on average" holds, enabling us to ascertain whether the PACO algorithm is competitive. Tables 5.1 and 5.2 show the average ranks of the best respective parameter combination for the standard ACO and for PACO after a different number of iterations and for different values of ρ. The number of iterations used in the tables are $t_0 = t/100$, $t_1 = t/50$, $t_2 = t/25$, $t_3 = t/10$, $t_4 = t/5$, $t_5 = t/2.5$, and $t_6 = t$.

Table 5.1: Average ranks for standard ACO and PACO on TSP

		t_0	t_1	t_2	t_3	t_4	t_5	t_6
$\rho = 0.1$	Standard	27	34.8	50.6	57	77	89.4	101.6
	PACO	9	17.8	16.8	19.4	20.2	16.2	14
$\rho = 0.05$	Standard	47.4	50.6	49.4	54.2	71.8	85.2	92.6
	PACO	18.8	18.0	19.6	19.2	16.6	14.4	11.6
$\rho = 0.02$	Standard	65.2	64.4	47.8	43.2	53	62.2	77.4
	PACO	17.8	19.8	19	17	15.2	11.8	13.6
$\rho = 0.01$	Standard	69.2	68.2	48	43.4	55.8	61	69.8
	PACO	18.6	18.4	16	15.4	11.6	14	14.6
$\rho = 0.005$	Standard	75.2	66.2	48.6	51.2	58.2	64.2	71.6
	PACO	17.2	15.8	14.8	12.2	13.6	15	18.8

For both tables, with the exception of using $\rho = 0.005$ at time t_5 for QAP, the best combination of parameters for the PACO algorithm always achieves a better average rank than the standard ACO algorithm for any t_i, indicating that it's performance is indeed competitive.

The average rank (for all t_i) over all parameter combinations is 66.5 for TSP and 53.2 for QAP. The tables show that the average rank of the best parameter combination is significantly lower for both the standard ACO and the PACO algorithm on QAP than on TSP. This seems to indicate that more of a distinction between good and bad parameter sets can be made for QAP, whereas for the TSP the different parameter combinations perform closer to one another in terms of quality. Also the use of a heuristic for TSP might result in a more similar behavior of the different parameter combinations.

Table 5.2: Average ranks for standard ACO and PACO algorithm on QAP

		t_0	t_1	t_2	t_3	t_4	t_5	t_6
$\rho = 0.1$	Standard	27.4	25.8	14.4	4.6	2.4	6.6	25.8
	PACO	4	4.2	3	3.6	3.6	4.2	5.2
$\rho = 0.05$	Standard	38.2	26.2	17	7	5	8.6	27.4
	PACO	2.8	3.4	2.6	3.2	5.2	5.8	5.2
$\rho = 0.02$	Standard	40.8	41.6	28.2	15.8	8	7	17
	PACO	3.4	2.6	2.6	4.8	5.8	5.8	6.2
$\rho = 0.01$	Standard	46.8	44.6	31.8	19.6	8.2	7	18
	PACO	2.6	2.6	3.6	5.2	5.6	6.2	7.4
$\rho = 0.005$	Standard	44.8	41.8	31.2	14	9	3	14.8
	PACO	2.2	4.4	5	5.6	6.4	7.8	7.2

Another effect is that for TSP, the best parameter combination seems to be more instance-specific than for QAP.

Note that for any row in Tables 5.1 and 5.2, the combination of parameters that is best does not necessarily stay the same. Instead, it undergoes a characteristic development depending on at what time t_i best performance is sought. Tables 5.3 and 5.4 give examples for the parameter-development of the best combinations that takes place in such a row for the TSP and QAP respectively. For the standard ACO algorithm, the typical development toward parameters that focus more on exploration than on exploitation when given ample time can be seen with declining q_0 and w_e, for TSP as well as QAP.

For the PACO algorithm, the situation is less clear. For TSP, all best combinations from Table 5.1 use population size $k = 1$, which means that the only guidance for the ants of a given iteration are the solutions of the elite ant and of the best working ant from the previous iteration, and the heuristic. It seems that the ability to quickly explore the neighborhood of a promising solution (last iteration's best), combined with a drive toward the best solution found so far and a strong heuristic influence to prevent choices that seem drastically bad, works best. Of the best five PACO combinations for the scenario of Table 5.3, four use population size $k = 1$ and one $k = 5$. An elite weight $w_e \in \{0.25, 0.5, 0.75\}$ is also used by all these combinations, suggesting that some degree of enforcement of the best solution found so far is beneficial. These traits are combined with $q_0 = 0.5$ and at first $\tau_{max} = 3$, replaced by

Table 5.3: Parameter-development over time for the best respective combinations on TSP (top 3 rows for standard ACO algorithm, bottom four for PACO), $\rho = 0.01$.

$\rho = 0.01$	200	400	800	2000	4000	8000	20000
q_0	0.5	0.5	0	0	0	0	0
w_e	0.75	0.75	0.75	0.5	0.5	0.25	0.25
$\tau_{\max}$	1	1	1	1	1	1	1
q_0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
w_e	0.25	0.25	0.25	0.25	0.25	0.25	0.25
$\tau_{\max}$	3	1	1	1	1	1	1
k	1	1	1	1	1	1	1

Table 5.4: Parameter-development over time for the best respective combinations on QAP (top 3 rows for standard ACO algorithm, bottom four for PACO), $\rho = 0.01$.

$\rho = 0.01$	200	400	800	2000	4000	8000	20000
q_0	0.9	0.5	0.5	0	0	0	0
w_e	1	0.75	1	0.75	0.5	0.25	0.25
$\tau_{\max}$	1	1	1	1	1	1	1
q_0	0.5	0.9	0.9	0.9	0.9	0.9	0.9
w_e	0.75	0.5	0.5	0.5	0.5	0.75	0.75
$\tau_{\max}$	10	3	3	3	3	3	10
k	1	5	5	5	5	25	1

$\tau_{\max} = 1$ after t_1. The parameter settings $q_0 = 0.9$ and $\tau_{\max} = 10$ achieve a similar effect in the PACO algorithm, both determining how likely it is that an ant will choose the best (q_0) or one of the best ($\tau_{\max}$) choices according to pheromone and heuristic information. The medium q_0 value and low $\tau_{\max}$ indicate that deviating often from the best path found so far produces good results.

For QAP, the parameter combinations which performed well are somewhat different from those for TSP. The only similarity is the strong guidance provided by an elite ant with $w_e \in \{0.5, 0.75\}$. However, both a higher q_0 and a higher

$\tau_{\max}$ value suggest that it is beneficial to stay closer to the best assignment(s) encoded in the pheromone matrix. Also, the larger population size of $k = 5$ and once even $k = 25$ point out that a greater diversity in solutions encoded into the matrix is helpful. The probable reason for this is the absence of a heuristic to guide the ants when a strong pheromone trail is not available.

5.3.5 Comparison of PACO with MAX-MIN Ant Algorithm

In order to compare the PACO algorithm with the *MAX-MIN* Ant Algorithm proposed in Subsection 5.3.1, we also use a rank-based comparison as above in Subsection 5.3.4 (altogether 84 different parameter combinations, 45 for the Max-Min and 39 for the PACO algorithm).

Table 5.5: Average ranks for Max-Min and PACO algorithm on TSP

	500	1000	2000	5000	10000	20000	50000
Max-Min	20.6	14.6	21.6	14	22.3	17.3	17
PACO	2.6	4.6	11.6	11.3	10.6	11.3	12.6

Table 5.6: Average ranks for Max-Min and PACO algorithm on QAP

	500	1000	2000	5000	10000	20000	50000
Max-Min	18.5	17.5	14	13	11	8.5	8
PACO	12	13.5	13	9	4.5	7.5	10

Tables 5.5 and 5.6 show the average rank of the best parameter combination for the PACO and *MAX-MIN* Ant Algorithm for TSP and QAP respectively. The main observation than can be made is that for all cases but one (QAP after 50000 iterations), the average rank of the PACO algorithm is lower than that of the *MAX-MIN* Ant Algorithm, indicating that the former's performance is at least as good as the latter's. This suggests that the solution quality which is achieved by the PACO algorithm is not only due to the implicit minimum and maximum pheromone values, but also to the quick adaptation a small explicit population of solutions is able to perform.

5.4 Evaluation on Dynamic Problems

Having established that the PACO algorithm works at least as well as the standard ACO meta-heuristic on some static problem instances in Section 5.3, we turn our attention to the dynamic problem instances which originally inspired the modified update scheme in PACO. When a change occurs to a problem instance, PACO reacts by repairing all solutions in the population with the mechanism introduced as the KeepElitist strategy in Subsection 4.2.5. The new pheromone matrix is derived from the repaired population via Equation 5.1. In addition to the DTSP introduced in Chapter 4, we also test the PACO algorithm on a dynamic variant of the QAP, which we explain below.

5.4.1 Dynamic Quadratic Assignment Problem

In the Dynamic Quadratic Assignment Problem (DQAP), the locations that facilities are assigned to can spontaneously change their position. When a change occurs to a DQAP instance, the following repair mechanism is applied to the current solutions. The facilities corresponding to the altered locations are reassigned sequentially to the new positions in such a fashion that the deterioration in solution quality resulting from each individual assignment is minimized. For the DQAP, this process is more computationally expensive than the repair mechanism for the DTSP, requiring $O(n)$ steps for a reassignment for the DQAP compared to a constant number of calculations for the DTSP.

5.4.2 Test Setup

The DTSP test setup is practically identical to the one used in Section 4.5 for analyzing the pheromone modification strategies proposed for an evaporation based ACO algorithm. From the TSP instance `rd400` with 400 customers taken from the [TSPLIB, 2003], 200 random customers are removed to form a spare pool before the algorithm starts. While the algorithm is running, $k \in \{1, 5, 25\}$ random customers are exchanged between the instance and the spare pool every $t \in \{50, 200, 750\}$ iterations. Note that we do not examine the difference between exchanging completely random and more clustered groups as we did in Section 4.5. Thus, there are 9 DTSP scenarios (c, t) which we analyze.
For the DQAP, we take the instance `tai150b` from the [QAPLIB, 2003] which consists of 150 facilities and locations, and proceed similarly as for the DTSP.

75 random locations are removed before the start of the algorithm and from a spare pool, and while the algorithm is running, $k \in \{1, 5, 15\}$ random locations are exchanged between the instance and the spare pool every $t \in \{50, 200, 750\}$ iterations, resulting in 9 scenarios as for the DTSP. Note that since the spare pool is of size 75 and not 200 as for the DTSP instance, we limit the change severity to 15 locations instead of 25. For all runs, the first 75 facilities in `tai150b` were used.

The parameters of the PACO algorithm are chosen in accordance with the results in Section 5.3 and some preliminary testing. For both the DTSP and the DQAP, $q_0 = 0.9$ and $\alpha = 1$ was used for the decision rule, while $\beta = 5$ is used exclusively for the DTSP since no heuristic is utilized for the DQAP. This implies another different setting, namely the maximum pheromone value, which is set to $\tau_{\max} = 1$ for the DTSP and $\tau_{\max} = 5$ for the DQAP, which requires more pheromone induced guidance due to the absence of heuristic values. We explore all the proposed strategies for the removal of solutions from the population, i.e. Age-based, Quality-based, Prob-based, and Age&Prob-based, with population limits of $k \in \{1, 3, 6, 10\}$. Note that for $k = 1$, Age-based and Age&Prob-based removal are identical. We also evaluate the effect of elitism by having one of the individuals in the population represent the best found solution, leaving $k - 1$ solutions to be managed by the chosen strategy. We also analyze the effect of elitism. However, in order to make the comparison between the strategies using elitism and not using elitism fair, we subtract 1 from the maximum population size k when employing elitism to reflect the additional solution. Otherwise, the algorithms with elitism would have a larger population and therefore a greater chance of having a good repaired solution in the population after a change. By proceeding in this fashion, when $k = 1$, using elitism for any strategy is the same as using Quality-based removal without elitism. The elitist weight is $1/k$, meaning that the elitist solution has the same influence on the pheromone matrix as the regular solutions.

For each of the scenarios (c, t), 25 random instances were created. The results in the following section are based on the integral performance, i.e. the average achieved solution quality, over 9000 iterations, averaged over the 25 random seeds. For comparison, we measure the performance of a strategy which does not repair the solutions in the population but rather restarts the algorithm each time a change occurs. For this, we use PACO with the best performing configuration on static TSP and QAP, which is Age-based removal with $k = 1$.

5.4.3 Results

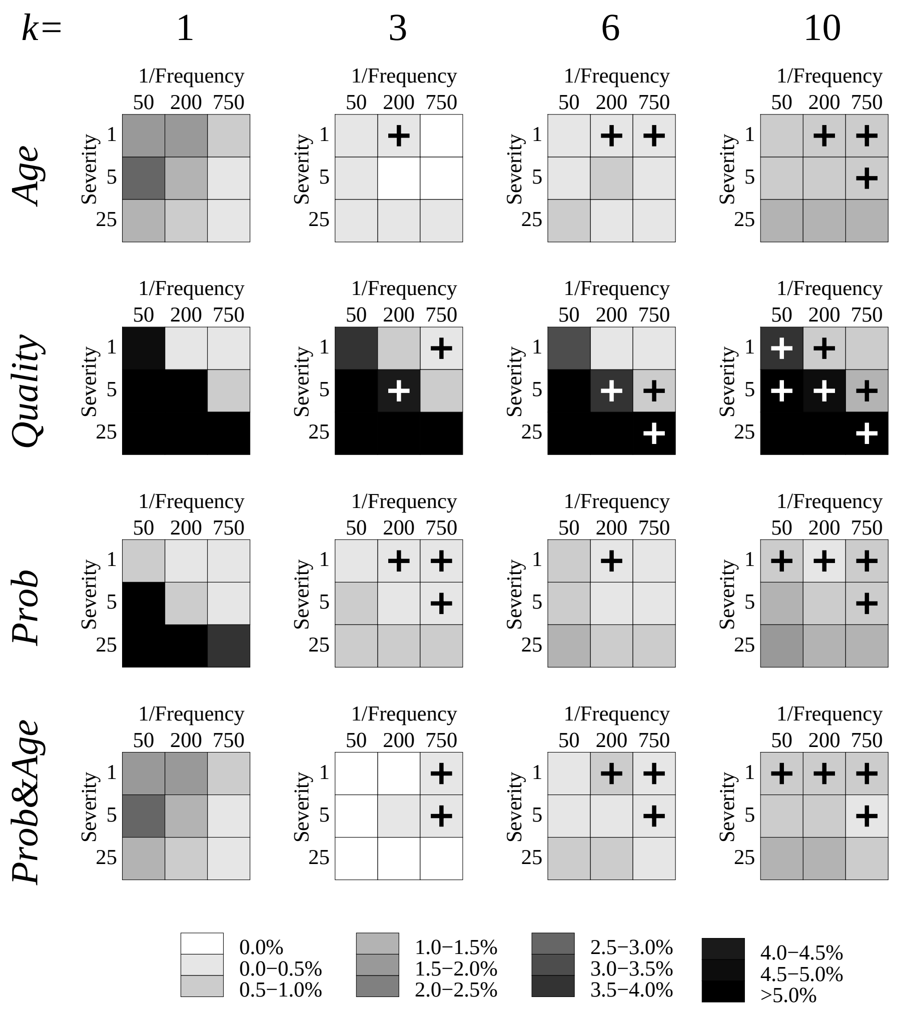


Figure 5.2: Relative deviation from the respective best solution by the individual configurations for TSP is shown. A "+" indicates that the use of elitism leads to better performance; otherwise, not using elitism performs better.

The Figures 5.2 and 5.3 show the resulting quality of the PACO algorithm for the DTSP and DQAP respectively. All scenarios and parameter combinations proposed in the test setup in Subsection 5.4.2 used with the repair of the solutions in the population are covered. The interpretation of the figures is the same as for the previous Figures 4.9 and 4.10 in Chapter 4. Basically, the lighter a square is in these two figures, the better the relative performance of

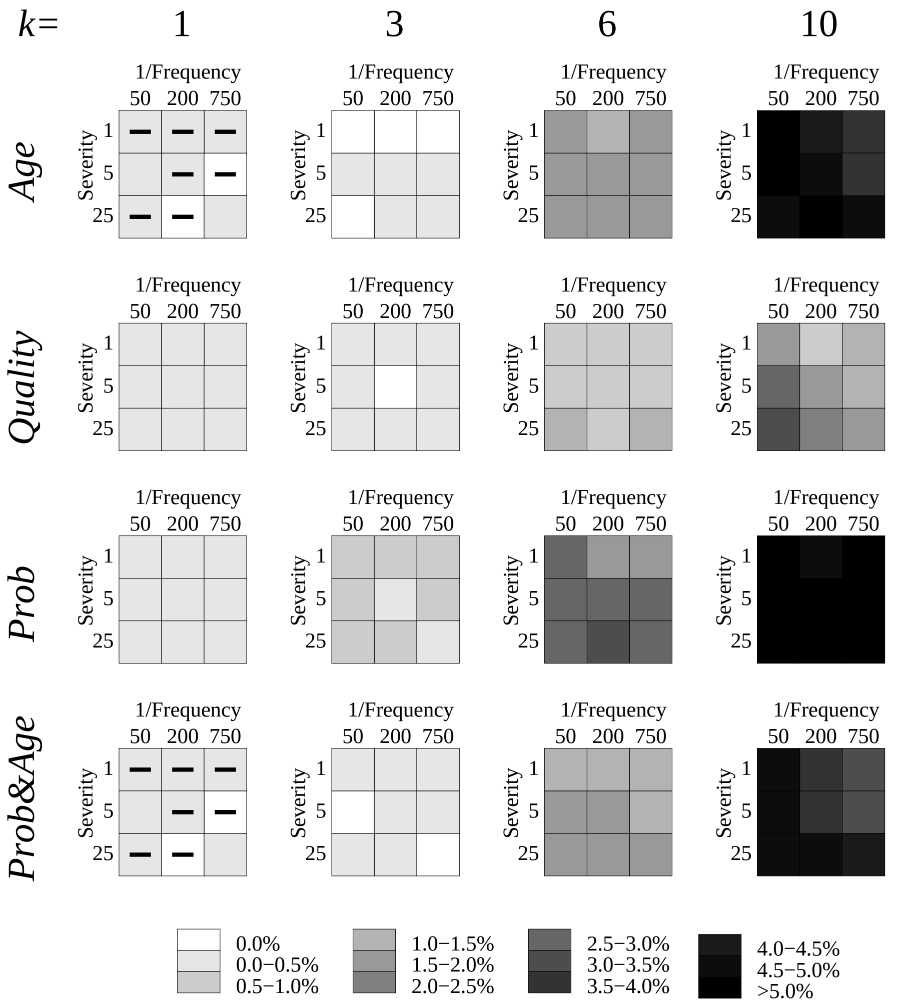


Figure 5.3: Relative deviation from the respective best solution by the individual configurations for QAP is shown. A "-" indicates that not using elitism leads to better performance better; otherwise, using elitism performs better.

the indicated configuration on the corresponding scenario. The best configuration always has a white square, and any configuration that performs more than 5% worse than the respective best is colored completely black. In addition to the shading, some of the squares in Figure 5.2 exhibit a "+" sign. This is used when employing elitism in combination with the indicated configuration leads to a better solution than the omission of elitism, which for DTSP performs better in the majority of cases, i.e. all those not marked with "+". A similar approach was taken in Figure 5.3. However, for DQAP, in most cases the use of elitism leads to better solutions, and hence a "-" is shown when the omission of elitism performs better.

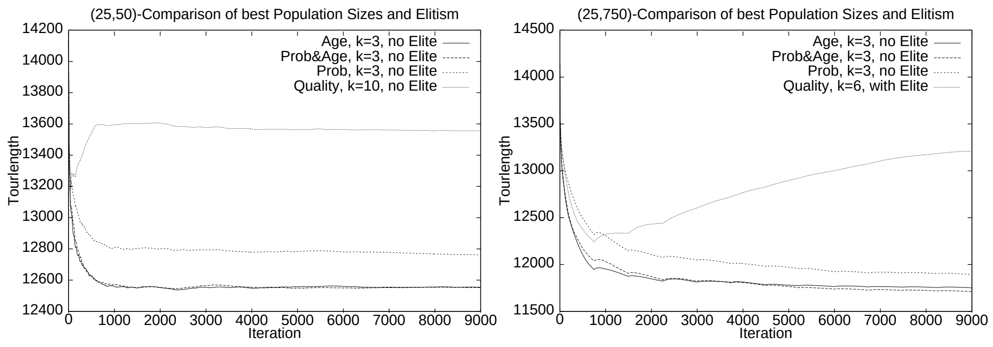


Figure 5.4: Comparison of strategies in (25,50) and (25,750) scenarios for TSP (integral performance, i.e. average over all iterations from the first to the actual).

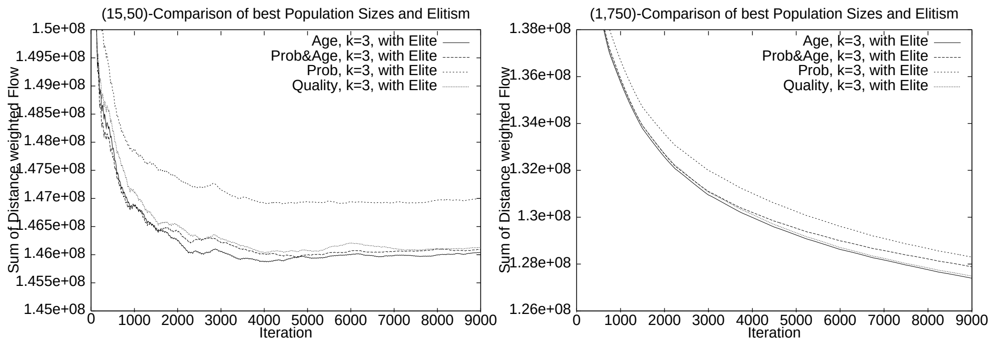


Figure 5.5: Comparison of strategies in (15,50) and (1,750) scenarios for QAP (integral performance, i.e. average over all iterations from the first to the actual).

As mentioned above, for DTSP, most configurations perform better when not combined with elitism. Specifically, the best solution for each scenario is found by a configuration that does not employ elitism. It should be noted that even when the use of elitism creates a better solution, the improvement is typically less than 0.5%. The cases in which configurations benefit most from using elitism are when only small changes occur to the problem, thus making it likely that the modified best solution would still represent a very good solution, and when population size is not too small and there is enough time to do exploration beside exploitation of the elite solution in the population.

The two best configurations however, which are Age-based and Prob&Age-based update with $k = 3$, exhibit best performance when not using elitism. The Quality-based Strategy performs poorly for most of the scenarios and population sizes, especially when frequent and severe changes occur. Indeed, in these cases, the Quality-based strategy performs increasingly bad over time, with the often and extensively repaired previous best solutions causing a focus on an area of the search space with bad solution quality, see Figure 5.4. The reason for this behavior is probably due to the combination of strong and usually good heuristic values with the deterministic fashion of holding only the best solutions, which effectively obstructs further exploration.

For DQAP, the behavior of the individual configurations is different from DTSP in some ways. Here, all configurations with $k > 1$ benefit from elitism, and even some with $k = 1$ (effectively turning them into a Quality-based update strategy). Judging from the good performance of the configurations with $k = 1$ and the Quality-based Strategy for larger sizes of k as well, it seems that the ants require a strong and not too variable guidance when looking for good solutions, with the quick convergence behavior of a small population size outweighing the better handling of changes to the problem instance by a larger population. For small populations, the difference between the individual configurations is not very large, which can be seen for 2 representative scenarios in Figure 5.5. Recall that these curves show integral performance, which explains why the values are still falling noticeably at the end of 9000 iterations. The only strategy which performs significantly worse than the others is Prob-based removal, which has neither the flexibility exhibited by the Age-based update in exploring new areas of the search space nor the persistence of the Quality-based Strategy in exhaustively exploiting the surroundings of a good solution.

So far we have only examined how the dynamic strategies compare to one another. The easiest method for dealing with any dynamic problem is of course to simply restart the algorithm. If this method proves more successful than integrating the changes into an ongoing optimization process, the integration is superfluous. Figure 5.6 shows the integral performance for the PACO algorithm using the Age-based population update and $k = 1$ (which is one of the best configurations for using restart).

One can see that restarting performs vastly inferior to integrating the changes, for DTSP as well as DQAP. However, restarting starts becoming competitive in DTSP when the changes to the instance become too great and/or when the change interval is long enough. For DQAP, the shades are darker than

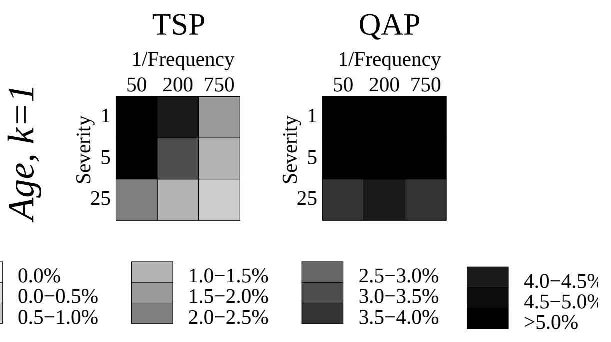


Figure 5.6: Solution quality when restarting PACO on DTSP and DQAP, compared to the best performing configuration which incorporates the changes.

for DTSP, indicating that the necessary change severity and duration between changes must be even larger in order for restarting to perform on the same scale.

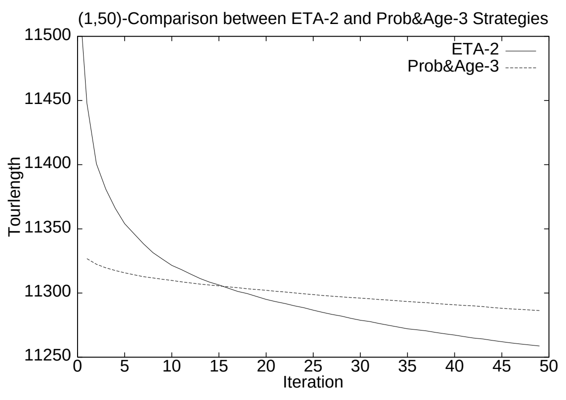


Figure 5.7: η-Strategy with $\lambda_E = 2$ compared to Prob&Age with $k = 3$.

In Chapter 4, we introduced modifications for the standard, evaporation based ACO meta-heuristic to enhance the performance on the DTSP. We use the $(c, t) = (1, 50)$ scenario given in Figure 5.7 to compare the two approaches. The best evaporation based ACO strategy that was studied in Section 4.7 for this scenario is the η-Strategy with $\lambda_E = 2$; for PACO, we use Prob&Age with $k = 3$. The figure shows the characteristic difference between the strategies for the evaporation based ACO and PACO in general. There is a comparatively large loss of solution quality by the η-strategy after a change, followed by fast improvement, while the Prob&Age-strategy shows small solution deterioration and slow improvement. The integral performance by the Prob&Age-strategy is only slightly worse than that of the η-strategy, which, considering the η-strategy is DTSP-specific, is quite promising.

5.5 Summary and Outlook

The Population-based Ant Colony Optimization (PACO) meta-heuristic is a new type of Ant Algorithm, which maintains a population of solutions from which pheromone information can be derived rather than an explicit pheromone matrix. We have shown for a number of static instances from the TSP and QAP problem classes that PACO performs at least as well as the regular ACO, and also at least as well as a *MAX-MIN* Ant Algorithm. For the TSP, where good heuristic information exists, a very small population is necessary for the algorithm to perform well, often consisting of only a single solution. For the QAP, the best-performing population size increases to 5, which is plausible since no heuristic information is available, and the algorithm must rely on previous samples of the solution space. Regardless, the maximum population value, which was 25 in our tests, performed significantly worse than the smaller populations in all test instances, indicating that for PACO, the population should always be relatively small, especially when heuristic information is present. The comparison with the *MAX-MIN* Ant Algorithm showed that not only does the PACO algorithm benefit from minimum or maximum pheromone values, but also from the versatility of the population, which can move to new areas of the solution space more quickly than the conventional ACO.
On the dynamic problem instances, again the small populations outperformed the larger ones, even though the large populations have a higher potential for good solutions to be present in the repaired population after a change. The repair mechanism itself is successful, performing significantly better than a restart of the algorithm with an empty population. For the DTSP, the high quality heuristic information makes keeping an explicit elitist ant superfluous, and the Quality-based Strategy for maintaining the population also performs badly. For the DQAP, the opposite is true. Since no heuristic information is available, it is essential to maintain the elitist solution, either separately or through the Quality-based update strategy.
The PACO algorithm holds great potential for future work. Since the PACO algorithm maintains a population of solutions, it can work together with other algorithms that manage such a population, like Genetic Algorithms (GA) [Holland, 1975, Goldberg, 1989], Population Based Incremental Learning (PBIL)

[Baluja, 1994, Baluja and Caruana, 1995], or Asynchronous Teams (A-Team) [Murthy, 1992, Souza, 1993]. First steps in this direction, specifically using PACO as a crossover operator for a GA on the TSP, have been undertaken by [Branke et al., 2003] with great success, and we believe that further work could highlight the similarities and differences of these population driven approaches.

Another area of research for PACO is the implementation in hardware. Due to the discrete nature of the population and the induced pheromone values, PACO is better suited for a hardware implementation, where floating point numbers are notoriously expensive in terms of computational complexity. Indeed, with the hardware implementation given in [Diessel et al., 2002], the PACO algorithm with a constant population size and working without heuristic information can be implemented in such a way that the generation of an entire solution takes only $O(n)$ steps. Together with the parallel generation and evaluation of solutions, this makes the hardware implementation several orders of magnitude faster than current software implementations on a sequential PC.

Another field where PACO has an advantage over the normal ACO due to its structure is Multi-Criteria Optimization (MCO), where the search process is focused on a number of solutions, the so-called Pareto-front, instead of just one. We analyze the applicability of PACO on MCO in detail in Chapter 6.

Chapter 6

Multi-Criteria Optimization

6.1 Optimizing more than one Objective

Most work on combinatorial optimization problems deals with the optimization of a single objective. For many real-world problems optimizing only a single criterion is a valid approach, especially when either no other objectives exist or their impact on solution quality is negligible. However, for many optimization problems, a number of objectives exist, perhaps even conflicting ones, that should all be optimized. Some examples for this are portfolio optimization, where one attempts to simultaneously minimize the risk and maximize the fiscal return, bridge construction, where a good design is characterized by low total mass and high stiffness, and some machine scheduling problems, where both the total tardiness over all jobs and the sum of setup costs between consecutive jobs should be minimized. Once more than one objective needs to be considered, there will usually no longer exist a single globally optimal solution, but rather a number of solutions, none of which is better than another in all objectives, i.e. none of which is dominated by another solution. These solutions make up the Pareto-optimal set of solutions, and the aim of Multi-Criteria Optimization (MCO) is to find or approximate all solutions in this set.

Numerous deterministic methods for solving MCO problems have been suggested. One of the most intuitive is to reduce the multiple optimization objectives to a single objective by assigning weights $w_i > 0$ to the individual objective functions f_i and finding the solution π which minimizes $F(\pi) = \sum_i w_i \cdot f_i(\pi)$. This technique requires the user to choose weights that reflect his preferences, and, if the global optimum is found, leads to a (single) solution from the set of Pareto-optimal solutions. A drawback to this method is its inability to cope with problems that have a set of Pareto-optimal solutions which are concave in the objective space, see [Das and Dennis, 1997]. In Goal Programming [Ignizio, 1976, Schniederjans, 1995], only one objective is minimized while the others are constrained to remain above a given lower bound or beneath an upper bound. Again, only a single solution is generated. If the objectives can be ordered by importance, then using Multilevel Programming [Bracken and McGill, 1973, Candler and Norton, 1977] is possible. At first, all solutions which are optimal with respect to the most important objective are sought. Then, In this set, all solutions optimal according to the second criterion are identified, and the reduced set is analyzed with respect to the third criterion, and so forth. The algorithm ends either before all objectives have been considered and only one solution is left, or after the last objective has been processed. All of the methods mentioned above can be used to generate multiple solutions via multiple passed by changing the parameters, goals and optimization objective, and hierarchy respectively. However, this is a very tedious process, and, depending on the shape of the surface defined by the set of Pareto-optimal solutions, will not be able to find all solutions.

Evolutionary Algorithms (EA) have also been used extensively for solving MCO problems, see [Deb, 2001, Coello et al., 2002] for an overview. Since EAs maintain a population of solutions from which new solutions are derived, they are an intuitive choice for this type of problem, where a set of solutions is sought. Most MCO problems to which EAs are applied are real-valued, which is a problem domain that is currently inaccessible to Ant Colony Optimization.

Recently, Ant Colony Optimization (ACO) has also been used for finding solution to MCO problems where the solution is either represented by a binary vector or a permutation. In most of the earlier work, it is assumed that the optimization criteria can be weighted by importance, similarly to the Multilevel Programming approach. [Mariano and Morales, 1999] propose a multi colony ACO approach where for each objective there exists one colony of ants. They study a problem where every objective is influenced only by parts of a solution: an ant from colony i receives a (partial) solution from the ants of colony $i - 1$ and then tries to improve or extend this solution with respect to criterion i. A final solution that has passed through all colonies is allowed to update the pheromone information if it is part of the current non-dominated front of solutions found by the algorithm. [Gambardella et al., 1999b] introduce an ant algorithm for a bi-criterion vehicle routing problem where they also use one ant colony for each criterion. Criterion 1 (the number of vehicles) is considered to be more important than criterion 2 (the total travel time of the tours). The two colonies share a common global best solution which is used for pheromone update in both colonies. Colony 1 tries to find a solution with one vehicle less than the currently best solution, while colony 2 tries to improve this solution with respect to criterion 2 under the restriction that the number of vehicles needed does not increase. Whenever colony 1 finds a new global best solution both colonies start anew (with the new global best solution). In [Gagné et al., 2000], a multi-criterion approach for solving a single machine total tardiness problem with changeover costs and two additional criteria is tested. In their approach the changeover costs are considered to be most important, and although heuristic values that take all criteria into account are used by the ants, the amount of pheromone that an ant adds to the pheromone matrix depends solely on the changeover costs of the solution. A similar approach is used in [Gravel et al., 2002] for a four criterion industrial scheduling problem.

In [Doerner et al., 2001c,b], a method for solving a transportation problem is proposed where the aim is to minimize the total costs by searching for solutions that minimize two different criteria. The general approach is to use two colonies of ants where each colony concentrates on a different criterion by using different heuristics. In [Doerner et al., 2001c] one criterion is considered to be the main criterion. Every k iterations, the master population which minimizes the main criterion updates its pheromone information according to the good solutions found in the slave population, which minimizes the minor criterion. However, no information flow occurs from the slave to the master colony. In [Doerner et al., 2001b], both criteria are considered to be of equal importance. The size of both populations is adapted so that the colony that finds the better solution with respect to costs becomes larger. Information exchange between the colonies is done by so called spy ants which base their decisions on the pheromone matrices in both colonies.

The only ACO approaches so far that aim to cover the Pareto-front of a multi-objective optimization problem have been proposed in [Doerner et al., 2001a, Doerner et al.] and in [Iredi et al., 2001]. In [Doerner et al., 2001a, Doerner et al.], a portfolio optimization problem with more than two criteria is studied. For each criterion, a separate pheromone matrix is used. Instead of a population of ants for each criterion each ant assigns weights to the pheromone information for all criteria according to a random weight vector when constructing a solution. Pheromone update is done by ants that found the best or the second best solution with respect to one criterion. A problem with this approach is that solutions in the non-dominated front that are not among the best with respect to a single criterion do not update the pheromone information. In [Iredi et al., 2001], an approach to solve bi-criterion optimization problems with a multiple colony ant algorithm were the colonies specialize to find good solutions in different parts of the front of non-dominated solutions is analyzed. Cooperation between the colonies is established by allowing only ants with solutions in the global front of non-dominated solutions to update the pheromone information. Two methods for pheromone update in the colonies are proposed. In the update by origin method an ant updates only in its own colony. For the other method the sequence of solutions along the non-dominated front is split into p parts of equal size. Ants that have found solutions in the ith part update in colony i, $i \in [1, p]$. This update method is called update by region in the non-dominated front. It is shown that cooperation between the colonies results in good solutions being found along the whole Pareto front. Heterogeneous colonies are used where the ants have different preferences between the criteria when constructing a solution. For the Single Machine Total Tardiness with Setup Costs Problem (SMTTSCP), two pheromone matrices are used: $M = (\tau_{ij})$ for the total tardiness criterion, where τ_{ij} is the desirability that job j be positioned on place i of the schedule, and $M' = (\tau'_{ij})$ for the changeover cost criterion, where τ'_{ij} gives information about scheduling job j immediately after job i.

In this Chapter, we introduce a Population-based Ant Colony Optimization (PACO, see Chapter 5 for an introduction) approach to solve MCO problems. The main idea is that the population of solutions used to construct new solutions is chosen from the set of all non-dominated solutions found so far, which acts as a super-population. The aim is to find a set of different solutions which covers the Pareto-optimal Front. One advantage of the proposed algorithm is that it can be applied to problems with more than two criteria and is not biased to solutions that are the best for one criterion. Also, the algorithm can be applied to any MCO problem where it is possible to run a single-objective ACO on the individual criteria. Some of the ideas we present are published in [Guntsch and Middendorf, 2003].

In Section 6.2, we give a formal definition of dominance and the Pareto-optimal Front. Afterwards, in Section 6.3, we describe how to modify the PACO algorithm so that it can solve MCO problems with an arbitrary number of objectives. Section 6.4 contains the test setup, including a detailed description of the test problem and instances, how heuristic information is derived, and how the pheromone matrices are employed. We summarize our results and comment on possible future work at the end of the chapter in Section 6.5.

6.2 Pareto-optimality

Formally, MCO deals with finding solutions π which meet the $(c_g + c_h)$ constraints defined via

$$\forall i \in [1, c_g] : g_i(\pi) \geq 0 \quad (6.1)$$

$$\forall i \in [1, c_h] : h_i(\pi) = 0 \quad (6.2)$$

and optimize the vector-function

$$f(\pi) = [f_1(\pi), \ldots, f_\xi(\pi)]^T \quad (6.3)$$

In this case, there are ξ separate optimization criteria. In contrast to single criterion optimization where it is always possible to say that a solution π is better than, as good as, or worse than another solution σ, ranking solutions is not always possible once two or more criteria exist. A solution π can only be considered better than a solution σ if it is at least as good as σ in all criteria and truly better in at least one. In this case, we say π dominates σ, denoted by $\pi \succ \sigma$. Formally, if wlog smaller values for all criteria $i = 1, \ldots, \xi$ are better, we define

$$\pi \succ \sigma :\Leftrightarrow (\forall i \in [1, \xi] : f_i(\pi) \leq f_i(\sigma)) \wedge (\exists i \in [1, \xi] : f_i(\pi) < f_i(\sigma)) \quad (6.4)$$

The Pareto-optimal Front of solutions is defined as the subset of solutions $\mathcal{F} = \{\pi_1, \ldots, \pi_l\}$ within the set of all feasible solution $\mathcal{G}$, for which

$$\forall \pi \in \mathcal{F}, \forall \sigma \in \mathcal{G} : \sigma \not\succ \pi \quad (6.5)$$

holds. This idea for optimality in MCO was originally proposed in [Edgeworth, 1881] and later generalized in [Pareto, 1896], from where the name is derived. Figure 6.1 gives an example of a set of feasible solutions $\mathcal{G}$ in 6.1(a) and the corresponding Pareto-optimal Front $\mathcal{F} \subset \mathcal{G}$ in 6.1(b) for a scheduling problem where the total tardiness of all jobs and the sum of setup costs between consecutive jobs should be minimized. This problem will be explained in depth in Subsection 6.4.1.

6.3 PACO for Multi-Criteria Optimization Problems

In this section we introduce a PACO approach for finding solutions in multi-criteria environments. First, we discuss how the solutions used to derive the pheromone information are selected. Afterwards, new methods for ants to make decisions based on pheromone and heuristic information originating from different criteria are introduced.

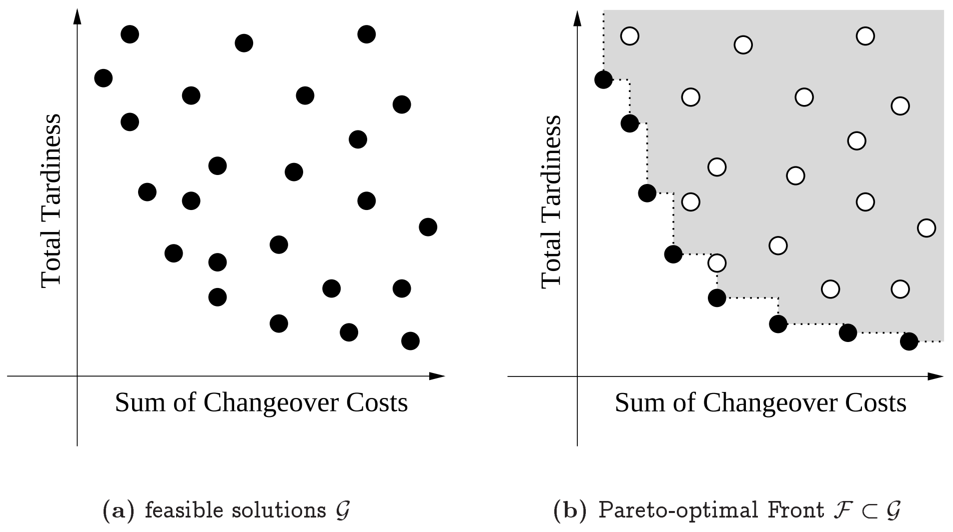


Figure 6.1: Set of feasible solutions $\mathcal{G}$ in (a) and the corresponding Pareto-optimal Front $\mathcal{F} \subset \mathcal{G}$ in (b).

6.3.1 Population of Solutions

Normally, the PACO algorithm maintains a population P of solutions which is reflected completely in the pheromone matrix, i.e. all solutions $\pi \in P$ have performed an update on the corresponding pheromone values, see Algorithm 5.1, p.90, for details. One of the characteristics of PACO which we discussed in Chapter 5 is that for the combinatorial optimization problems studied, the population sizes that perform best are relatively small, ranging from 1 to 5 solutions. The size of the set of Pareto-optimal solutions is typically bounded by a much higher number, which discourages using the entire Pareto-front as the "active" population which implies the pheromone matrix.

Let Q denote the set of non-dominated solutions that have been found after a number of iterations by the PACO algorithm. This set will act as the super-population for PACO, from which the active population $P \subseteq Q$ is derived to construct the pheromone information for the ants to work with. First, the algorithm chooses one starting solution π from Q at random. Then, the $k-1$ solutions in Q which are closest to π with respect to some distance measure are determined (if $|Q| \geq k-1$), where k is the maximum active population size. Distance is defined simply by the sum of absolute differences in solution quality over all criteria. Together, these k solutions form the population $P = \{\pi_1, \ldots, \pi_k\}$, with $\pi_1 = \pi$. After a solution has been constructed by an ant, the set Q is updated. After m ants have constructed a solution a new population P is chosen. Algorithm 6.1 shows a high-level description of the process.

Algorithm 6.1 basic procedure for PACO on MCO problems

```
1: initialize $P = \emptyset, Q = \emptyset$
2: repeat
3:     derive pheromone information from $P$
4:     for each ant $i \in \{1, \dots, m\}$ do
5:         initialize selection set $S \mapsto \{0, 1, \dots, n-1\}$
6:         let ant $i$ construct solution $\pi_i$
7:         if $\nexists \sigma \in Q : \sigma \succ \pi_i$ then
8:             $Q = Q \cup \{\pi_i\}$
9:         end if
10:    end for
11:    randomly choose $\pi \in_R Q$
12:    set $P \rightarrow \{\pi\}$
13:    while $|P| < k$ do
14:        find $\sigma^+ = \arg\min_{\sigma \in Q \setminus P} \sum_j |f_j(\pi) - f_j(\sigma)|$
15:        set $P = P \cup \{\sigma^+\}$
16:    end while
17: until condition for termination met
```

After the algorithm has terminated, Q is output as the result.

6.3.2 Generating the Probability Distribution

For mono-criterial permutation problems, we have discussed the node×node pheromone matrix representation, as used e.g. for the TSP, and the node×place representation, which is employed for the QAP or scheduling problems. Multi-criterial optimization problems can contain both types of criteria, and hence require both types of matrices. Specifically, the PACO for MCO problems maintains one pheromone Matrix which is interpreted in a node×node fashion and another as node×place.

Using these two matrices, it is possible to construct the individual probability distributions p_{ij}^l for the criteria $l = 1, \dots, \xi$ in a straightforward fashion via Equation 2.2, p.19. Proceeding in this fashion allows the usage of the heuristic information available for the respective single criterion, which in many cases helps the ants build good solutions quickly. What remains after the individual probability distributions p_{ij}^l have been computed is to aggregate them in a meaningful fashion. The basic idea for accomplishing this aggregation is to assign a weight $w_l \in (0, 1)$ with $l \in [1, \xi]$ to each of the probability distributions p_{ij}^l which reflect to what extent the search for a new solution in the non-dominated front should go in the direction of the implied criterion.

The population P is used to determine the weights $w_l = w_l(P)$ for each criterion $l = 1, \ldots, \xi$. The general idea is to give a criterion a high weight if the solutions in P are good in this criterion compared to all solutions in Q. Formally, to compute these weights we assign each solution $\pi \in P$ a reverse rank $r_l(\pi) \in [0, |Q| - 1]$ for each criterion l. By reverse rank we mean that $r_l(\pi) = 0$ is worst and $r_l(\pi) = |Q| - 1$ is best. Let $f_l(\pi)$ denote the quality of solution π with respect to criterion l, where, for minimization problems, lesser values of $f_l(\pi)$ indicate a better solution. Then

$$r_l(\pi) = |Q| - |\{\sigma \in Q | f_l(\sigma) < f_l(\pi)\}| - 1 \tag{6.6}$$

and, using this reverse rank, we define the weights $w_l(\pi)$ for the individual solutions via

$$w_l(\pi) = \frac{r_l(\pi)}{\sum_j r_j(\pi)} \tag{6.7}$$

Finally, from the individual solution weights, we calculate the combined weight for the population $P = \{\pi_1, \ldots, \pi_k\}$ by aggregating the weights of all solutions in P with respect to the criterion l

$$w_l = w_l(P) = \frac{1}{k} \sum_{i=1}^{k} w_l(\pi_i) \tag{6.8}$$

Note that

$$\sum_{l=1}^{\xi} w_l = 1 \tag{6.9}$$

for any population P. What remains now is to define exactly how these weights should influence the aggregation of the probability distributions p_{ij}^l. We propose the following two possibilities.

Weighted Sum

Since the weights for the individual criteria are calculated in such a way that their sum equals one, a straightforward method to aggregate the individual probability distributions is by calculating the weighted sum

$$p_{ij}^{\Sigma} = \sum_{l=1}^{\xi} w_l \cdot p_{ij}^l \tag{6.10}$$

The values p_{ij}^{Σ} are themselves a probability distribution that need not be scaled any further. Using this method, we expect the search process to be guided mostly by a single criterion at the edges of the Pareto-optimal Front and by the average composition at the inner parts.

Weighted Product

This method is an alternative to calculating the weighted sum, which is inspired by the idea that a very low probability for a given choice should not be averaged out by other high probabilities, but rather result in an overall low probability as well. This is accomplished by calculating a weighted product

$$p_{ij}^{\star} = \prod_{l=1}^{\xi} (p_{ij}^{l})^{w_l} \tag{6.11}$$

This distribution of values must be scaled by the inverse sum over all values to ensure that it becomes a probability distribution. Thus, for the weighted product, the values

$$p_{ij}^{\Pi} = \frac{p_{ij}^{\star}}{\sum_{j \in S} p_{ij}^{\star}} \tag{6.12}$$

give the final probability distribution.

6.4 Empirical Evaluation

In this section, we describe the tests performed to evaluate the performance of PACO on a specific MCO permutation problem, the Single Machine Total Tardiness with Setup Costs Problem (SMTTSCP). After defining this problem, we explain which ACO pheromone evaluation and heuristics to use in order to achieve good performance for the single criteria, i.e. minimizing total tardiness and the sum of setup costs, followed by the actual test setup and results.

6.4.1 An ACO Algorithm for the SMTTSCP

The Single Machine Total Tardiness with Setup Costs Problem (SMTTSCP) consists of scheduling n non-interruptible jobs on a single machine, starting with the next job in the schedule as soon as the previous is finished. Each of these jobs has a processing time p_j, indicating how long it takes to process this job on the machine, and a due date d_j, which is the time until which the job must be finished if no penalty, i.e. cost, is to be incurred. The total tardiness $\mathcal{T}(\pi)$ of a schedule π is defined as the sum over all individual tardinesses of jobs $j = 1, \dots, n$

$$\mathcal{T}(\pi) = \sum_{j=1}^{n} \max(C_j - d_j, 0) \tag{6.13}$$

where C_j is the completion time of job j:

$$C_j = \sum_{i:\pi(i) \leq \pi(j)} p_i \tag{6.14}$$

For this problem, a node$\times$place pheromone matrix representation is usually employed, where τ_{ij} gives information about the expected benefit of assigning job j to place i in the schedule. Instead of using only the pheromone value τ_{ij}, [Merkle and Middendorf, 2000] study using the sum over all pheromone values up to and including i

$$\tau'_{ij} = \sum_{l=1}^{i} \tau_{lj} \tag{6.15}$$

as pheromone information, since this more accurately reflects that a job which was not chosen despite a high pheromone value should be scheduled soon afterwards. A weighted combination of this summation evaluation and regular evaluation is studied for the Single Machine Total Weighted Tardiness Problem in [Merkle et al., 2000b] and shown to be superior to regular evaluation. Therefore, instead of the normal value τ_{ij}, we use

$$\tau^{\star}_{ij} = c \cdot x_i \cdot \tau_{ij} + (1 - c) \cdot y_i \cdot \sum_{l=1}^{i} \gamma^{i-l} \tau_{lj} \tag{6.16}$$

in Equation 2.2, p.19. The parameters of $\tau^{\star}_{ij}$ are c, which determines the relative influence of the weighted summation evaluation, γ, which indicates the weight of previous pheromone values at places $l < i$, and x_i and y_i, which are used for scaling, which is necessary since the value provided by the weighted summation evaluation can be a lot larger than the standard pheromone value. Specifically, the scaling values are $x_i = \sum_{h \in S} \sum_{l=1}^{i} \gamma^{i-l} \cdot \tau_{lh}$ and $y_i = \sum_{h \in S} \tau_{ih}$.

A very good heuristic function proposed in [Merkle and Middendorf, 2000] is also available for minimizing total tardiness, defined by

$$\eta_{ij} = \frac{1}{\max(T + p_j, d_j) - T} \tag{6.17}$$

where T is the total processing time of all jobs already scheduled, i.e. $T = \sum_{l=0}^{i-1} p_l$.
In addition to the processing times and due dates which are used to calculate the total tardiness of a schedule, a cost matrix $[c_{ij}]$ with $\forall i, j : c_{ij} \neq 0$ also exists which contains the setup costs for scheduling job j immediately after job i. The problem of minimizing the sum of these setup costs is virtually identical to finding the shortest tour for an asymmetric TSP, except that the tour is not a cycle and therefore ends at the last customer. Specifically, the cost $\mathcal{C}(\pi)$ of a schedule π of the jobs $j = 1, \ldots, n$ is defined as

$$\mathcal{C}(\pi) = \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \tag{6.18}$$

Due to this similarity, we use the standard approach as for a TSP for minimizing the sum of setup costs. Specifically, the pheromone matrix is represented in a node×node fashion, and the inverse cost is used as heuristic information, i.e.

$$\eta_{ij} = \frac{1}{c_{ij}} \tag{6.19}$$

Due to the fact that the first job scheduled incurs no setup cost, choosing which job to schedule first is not arbitrary (in contrast to the TSP). Since it is not possible to memorize which job to place first in a node×node encoded pheromone matrix, we insert a dummy job into all SMTTSCP instances, which has no processing time, no due date, and implies no setup cost for any following job. This dummy job is always scheduled first. We increase the dimension of the pheromone matrices by 1 to include the dummy job, and can thereby have information evolve in the node×node pheromone matrix about which “real” job to schedule first.

6.4.2 Test Setup

The SMTTSCP test instances we use are derived from [Iredi et al., 2001]. Each of the two bi-criterial problem instances consists of one due date and one changeover cost criterion. From these two instances, a four-criterial instance with two due date and two changeover criteria is constructed. For the first bi-criterial instance, called instance A from here on, the changeover costs between the jobs are chosen randomly from the interval $[1, 100]$, while for the other one, dubbed instance B, interval $[50, 100]$ is used. The processing times and due dates are chosen according to an often employed scheme from

[Crauwels et al., 1998]. For each job $j \in [1, 100]$, an integer processing time p_j is drawn randomly from the interval $[1, 100]$, and, after all jobs have been assigned a processing time, the due dates for each job j are drawn randomly from the interval

$$I = \left[\sum_{j=1}^{100} p_j \cdot (1 - TF - \frac{RDD}{2}) \ , \ \sum_{j=1}^{100} p_j \cdot (1 - TF + \frac{RDD}{2})\right] \tag{6.20}$$

RDD is the relative $Range$ of Due $Dates$ and determines the size of the interval from which the due dates are drawn. TF is the $Tardiness$ $Factor$ and indicates where the center of the above interval is located. For both instances, $RDD = 0.6$ was used; in instance A, we set $TF = 0.4$ and in instance B, $TF = 0.6$. The construction of these instances leads to instance A being easier to find good solutions for than instance B. The four-criterial instance which is a combination of instance A and B is called instance AB.
For the algorithm, we evaluate a number of parameter configurations. All combinations of population sizes $k \in \{1, 3, 5, 8\}$, $q_0 \in \{0.0, 0.5, 0.9\}$ and $\tau_{max} \in \{1, 5, 25, 125\}$ are tested for each of the two bi-criterial instances using the probability aggregations p_{ij}^{Σ} and p_{ij}^{Π}. The ants use $\alpha = 1, \beta = 5$ for the changeover based probability distributions, which is in accordance with previous tests on the similar TSP, and $\alpha = 1, \beta = 1$ for the tardiness based criteria respectively. The reason for choosing different values of β is the varying range of values and quality of the guidance provided by the heuristic functions for the two criterion-classes. We test all configurations with $m \in \{1, 2, 5\}$ ants per iteration before a new active population is drawn. In the case of more than 1 ant per iteration, all solutions of the iteration are candidates for Q. We let the algorithm construct 50000 solutions on the bi-criterial instances before termination. On the four-criterial instance, we allow for the construction of 100000 solutions to compensate for the increased complexity, predominately using parameter settings that perform well for the two bi-criterial instances which compose the four-criterial one.
In the following section we present and compare the median attainment surfaces of the fronts of non-dominated solutions found for 15 runs of the ant algorithm with different randoms seeds. The median attainment surface is the median line of all the attainment surfaces connecting the fronts of non-dominated solutions in each of the 15 runs. Figure 6.2 gives an example of the median attainment surface of three separate fronts of non-dominated solutions. Intuitively, the median front is the line-segment which is in the center according to the criterion to the axis of which the line is moving orthogonally.

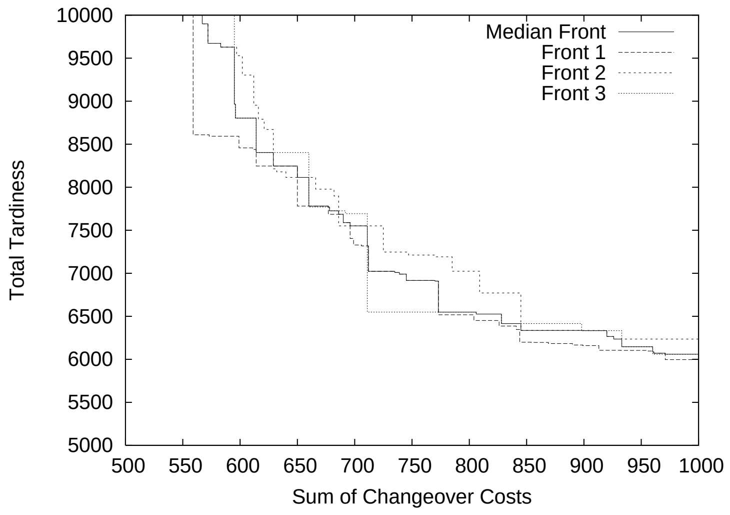
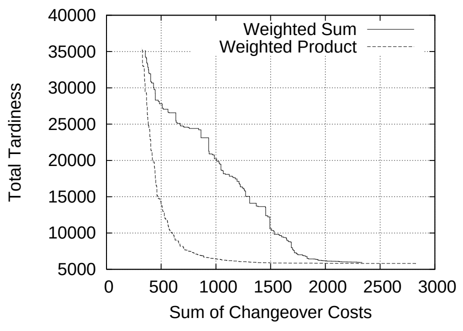


Figure 6.2: Median attainment surface of three separate fronts of non-dominated solutions.

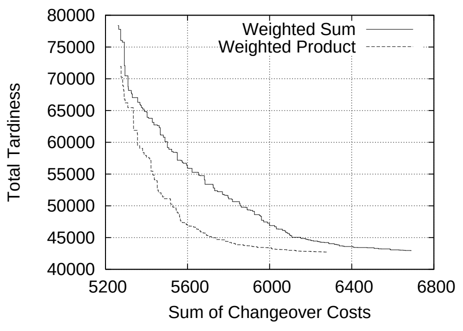
6.4.3 Results

We start the evaluation with two general results. The first is that the probability aggregation by weighted product vastly outperforms the method by weighted sum, particularly in the central parts of the front where solutions should perform well in both criteria, see Figure 6.3. For instances A and B, the respective median front of solutions found using the weighted product aggregation completely dominates the one attained via the weighted sum aggregation. The two methods perform closest at the fringes of the respective fronts, where one probability distribution is used almost exclusively. However, toward the center of the fronts, it is evident that the weighted product aggregation is more successful in creating a probability distribution which enables the algorithm to construct solutions which represent good compromises between the two criteria. For the instance B, the difference in solution quality is not as pronounced as for instance A, which might be an indicator that it is more difficult to find very good solutions for the former instance with the specified number of solution constructions.

The second general result is the expected fact that using $m > 1$ ants per iteration does not offer better results. Instead, in most cases the opposite is the case by a significant margin, as can be seen in Figure 6.4. The shape of the



(a) Instance A



(b) Instance B

Figure 6.3: Best performing combination of parameters for weighted sum and weighted product probability aggregation for instances A and B.

curves for $m > 1$ ants is representative for all configurations and also to some extent for instance A. While the total tardiness criterion is still optimized well, the algorithm is less capable of minimizing the changeover costs with more ants, resulting in fewer and worse solutions being found compared to 1 ant.

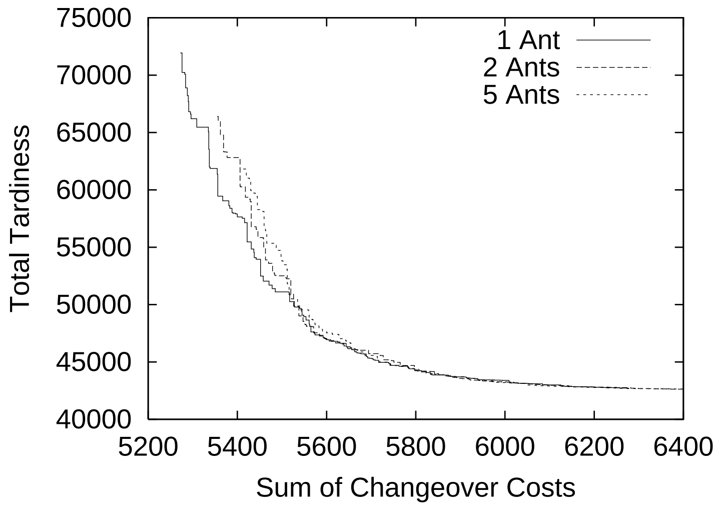
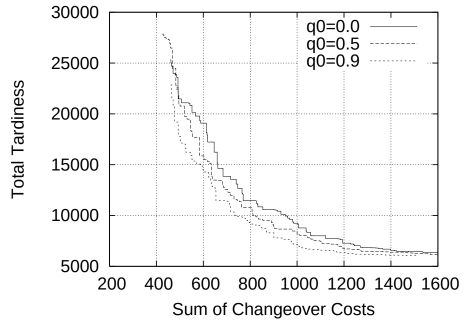
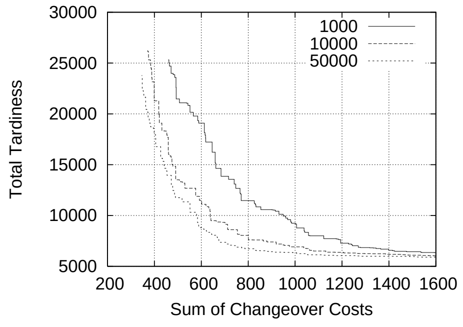


Figure 6.4: Non-dominated Sets for $m \in \{1, 2, 5\}$ on instance B.

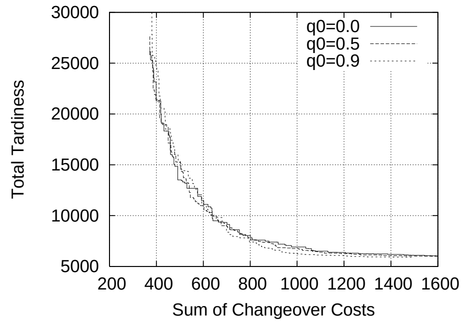
Since the weighted product aggregation outperforms the weighted sum in all cases, and using more than 1 ant gives no benefit, we concentrate on examin-



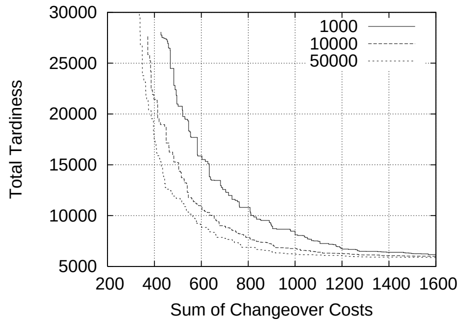
(a) Performance after 1000 Solutions



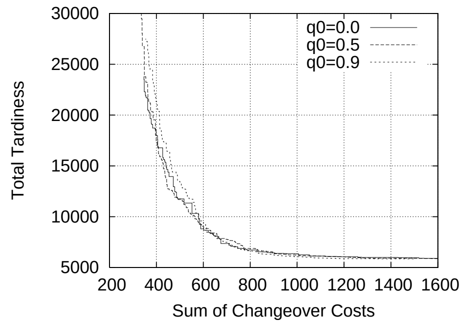
(b) $(k, q_0, \tau_{\max}) = (5, 0.0, 5.0)$



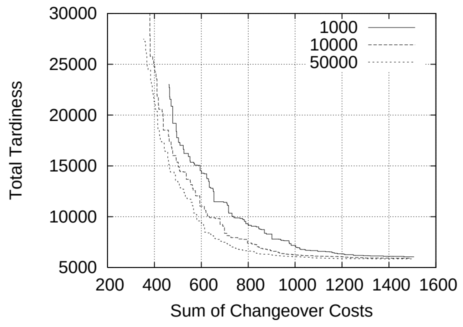
(c) Performance after 10000 Solutions



(d) $(k, q_0, \tau_{\max}) = (3, 0.5, 1.0)$



(e) Performance after 50000 Solutions



(f) $(k, q_0, \tau_{\max}) = (3, 0.9, 1.0)$

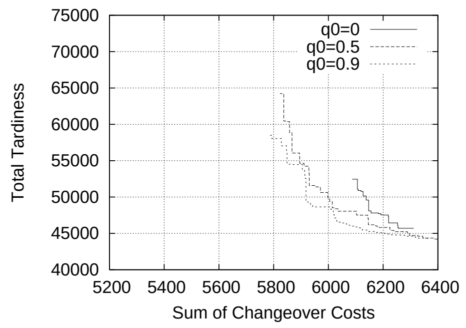
Figure 6.5: Best performing combinations of parameters for weighted product probability aggregation for instance A at different points in time. The left column compares the combinations after 1000, 10000, and 50000 solution constructions, while the right column shows the development of each individual parameter set.

ing the influence of the individual parameters on the PACO algorithm using only the weighted product aggregation method with $m = 1$. We begin with an analysis of the performance on instance A. On this instance, a number of parameter combinations perform well. Three of the best performing parameter configurations $(k, q_0, \tau_{max}) = (5, 0.0, 5.0)$, $(k, q_0, \tau_{max}) = (3, 0.5, 1.0)$, and $(k, q_0, \tau_{max}) = (3, 0.9, 1.0)$ are shown in Figure 6.5 after the construction of 1000 solutions, 10000 solutions, and at the end of the run, after the construction of 50000 solutions. After 1000 solutions, it is still possible to see a clear difference in performance between the above parameter configurations, but after 10000 or 50000 iterations, the performance of these configurations is on par, and a number of configurations with similar parameters exist whose performance is also nearly identical. The reason for this behavior is not convergence, as can be seen in the right column of Figure 6.5, since the front is still being improved. Rather, it seems that the problem is simple enough to allow a broad range of parameters to perform well. Mediocre performance was only observed for the extreme population sizes $k \in \{1, 8\}$, which perform significantly worse than $k \in \{3, 5\}$.

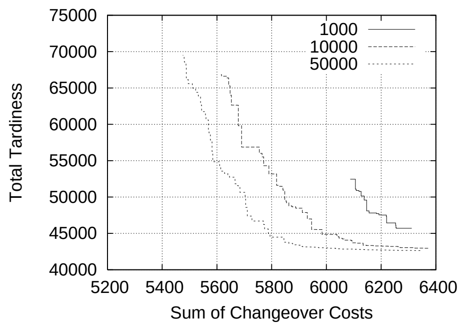
An interesting observation is the correlation of the values q_0 with the maximum pheromone level τ_{max}. For a lower value of q_0, higher values of τ_{max} are preferred by the algorithm, which have a similar effect as a higher q_0, increasing the likelihood of an element which carries a pheromone value $\tau_{ij} > \tau_0$ being chosen. Thus, for this instance, there seems to exist something like an optimal probability distribution according to which solutions should be constructed to ensure both exploration and exploitation.

For instance B, the more difficult instance, Figure 6.3(b) has already hinted at the fact that is seems more difficult to find solutions which offer good quality in both criteria. The parameter configurations we analyze in Figure 6.6 are $(k, q_0, \tau_{max}) = (3, 0.0, 25)$, $(k, q_0, \tau_{max}) = (3, 0.5, 5)$, and $(k, q_0, \tau_{max}) = (3, 0.9, 1)$, which are the best performing combinations for the individual q_0 values.

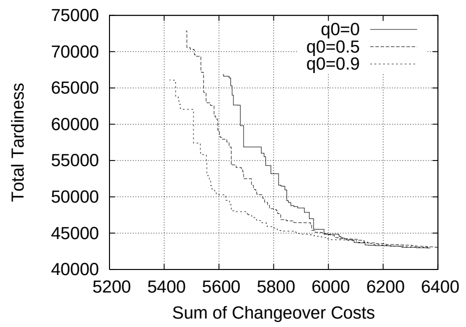
Many characteristics that were observed on instance A are also present here. Again, a population size $k = 3$ performs well, and a higher value of q_0 is accompanied by a lower value for τ_{max}, signifying that the probability distribution should have a distinct form for the algorithm to perform best. The most notable difference in comparison to instance A is that the relative development of the front occurs mostly with respect to the “Sum of Changeover Costs” criterion, while the values for “Total Tardiness” found after 50000 solution constructions are only slightly better than those found



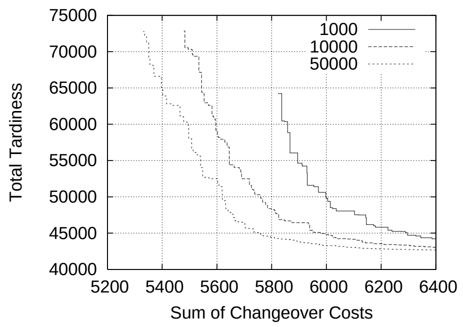
(a) Performance after 1000 Solutions



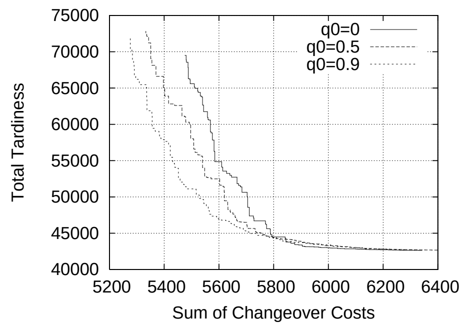
(b) $(k, q_0, \tau_{\max}) = (3, 0.0, 25.0)$



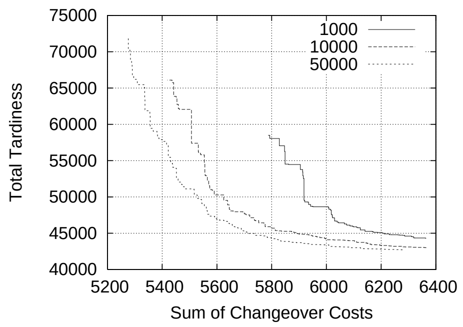
(c) Performance after 10000 Solutions



(d) $(k, q_0, \tau_{\max}) = (3, 0.5, 5.0)$



(e) Performance after 50000 Solutions



(f) $(k, q_0, \tau_{\max}) = (3, 0.9, 1.0)$

Figure 6.6: Best performing combinations of parameters for weighted product probability aggregation for instance B at different points in time. The left column compares the combinations after 1000, 10000, and 50000 solution constructions, while the right column shows the development of each individual parameter set.

after 10000 solutions have been built. Also, we can see that the quality of the set of non-dominated solutions found by the parameter configuration $(k, q_0, \tau_{max}) = (3, 0.9, 1)$ is superior to $(k, q_0, \tau_{max}) = (3, 0.5, 5)$, which in turn is superior to $(k, q_0, \tau_{max}) = (3, 0.0, 25)$.

A comparison in performance between the Multi Colony Ant Algorithm approach for bi-criterial problems proposed in [Iredi et al., 2001], which uses 10 colonies specialized on exploring different, overlapping areas of the Pareto-front, and the PACO algorithm for instance A is shown in Figure 6.7. For the greatest part, the set of solutions found by the PACO algorithm dominates the set found by the 10 colony Ant Algorithm. Only the upper left portion of the curve, which corresponds to solutions where practically only the changeover costs were minimized, is somewhat underdeveloped. Note that the comparison between these two algorithms takes place after the construction of 30000 solutions, which was the runtime for the 10 colony Ant Algorithm.

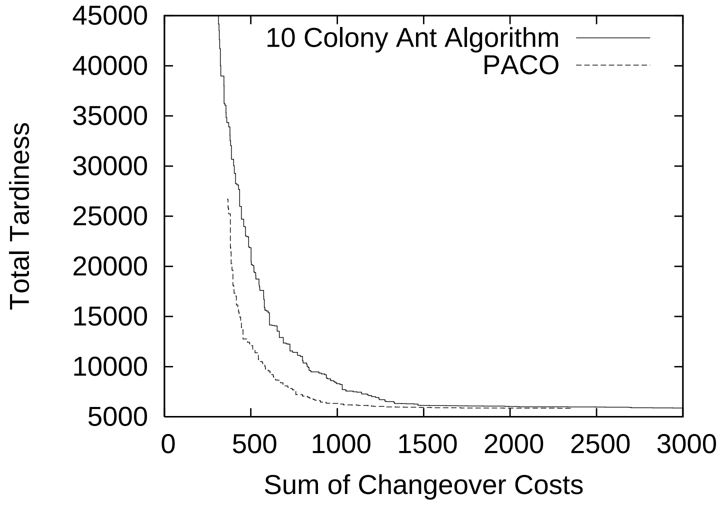
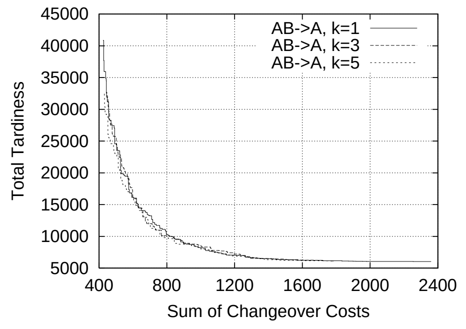
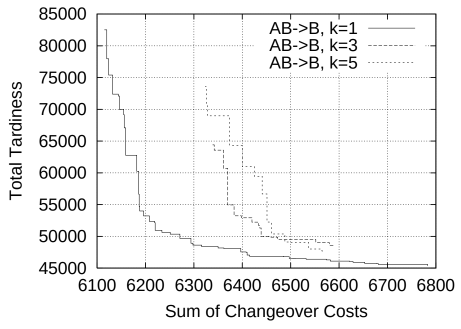


Figure 6.7: Comparison between 10 colony Ant Algorithm from [Iredi et al., 2001] and PACO algorithm with configuration $(k, q_0, \tau_{max}) = (3, 0.5, 1)$ after 30000 iterations.

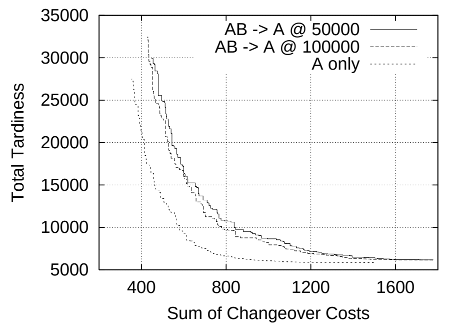
For the 4-dimensional instance AB, we evaluate the performance of the PACO algorithm with the parameter configurations $q_0 = 0.9$, $k \in \{1, 3, 5\}$ and $\tau_{max} \in \{1, 5\}$. These parameters work well for the two 2-dimensional problem instances A and B which compose AB, and should therefore also offer the best performance here. In order to gauge the performance on AB, we project the 4-dimensional non-dominated sets found by the algorithm to the two dimensions corresponding to instance A or B respectively, and compare these projected fronts among each other and to the ones found by the algorithm working on the 2-dimensional instance. The results are depicted in Figure 6.8.



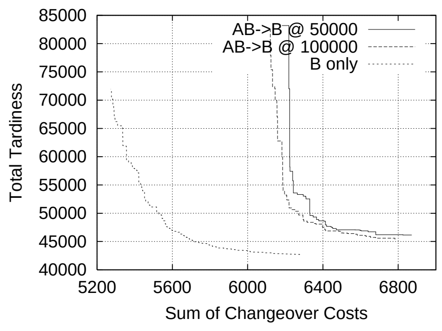
(a) $k \in \{1, 3, 5\}$ on Instance A



(b) $k \in \{1, 3, 5\}$ on Instance A



(c) best $AB \to A$ vs. best A



(d) best $AB \to B$ vs. best B

Figure 6.8: Comparison of 2-dimensional projections of 4-dimensional fronts corresponding to instances A and B in subfigures (a) and (b) respectively for different active population sizes after 100000 iterations. In (c) and (d), the best respective 2-dimensional projection for A and B after 50000 and 100000 iterations is compared with the best performing parameter combination on A and B individually.

Using a value of $\tau_{\max} = 1$ is superior for all population sizes to $\tau_{\max} = 5$, which is why we concentrate on highlighting the differences in performance for the population sizes $k \in \{1, 3, 5\}$. For easier reference, we will refer to the 2-dimensional projections of the 4-dimensional non-dominated sets of solutions as projections, while the curves resulting from a run on a 2-dimensional instance will be called fronts.

While for the projection to instance A in Subfigure 6.8(a) the different active population sizes perform very much alike, with $k = 5$ being marginally best, the situation changes for the more difficult instance B in 6.8(b). Here, $k = 1$ dominates the other values of k completely. Compared to the best performing parameter configuration on the bi-criterial instances, the projections are completely dominated by the fronts in both cases, even when given twice the number of iterations. Again, it seems especially difficult to develop a good front of solutions in the section corresponding to instance B, where the difference between best projection and best front is much larger than for instance A.

The reason for the worse quality of the projections compared to the fronts is the larger set of objectives which must be optimized concurrently and the much larger set Q which must be maintained as a result. For the bi-criterial runs, the average number of solutions in a front after 50000 iterations is about 75.53 , while after 100000 iterations on the best parameter configurations on instance AB, an average of around 9893.33 solutions is located in the front, 131 times more. This greater spread results in the individual solutions being used less often to create a pheromone matrix and hence less development taking place in all directions of the 4-dimensional front, which is the cause for the comparatively bad quality of the projections. Note however that the quality is bad only compared to the algorithm running on a bi-criterial instance; taken for themselves, the projections show that using the PACO algorithm for multi-dimensional problem instances with more than 2 objectives is indeed feasible and results in an acceptable solution quality, given the limited runtime.

6.5 Summary and Future Work

We have presented a number of augmentations to the PACO algorithm which enable it to solve Multi-Criteria Optimization Problems, outperforming a previous Ant Algorithm on bi-criterial instances and scaling to an arbitrary number of objectives. Two methods for aggregating the probability distributions for the individual criteria are compared, with the weighted product method outperforming the weighted sum in all cases. The concept of drawing an active population from a super-population was introduced and shown to be successful by the predominance of smaller populations in the best configurations for the individual test instances.

Compared to the Multi-Colony Ant Algorithm in [Iredi et al., 2001], the PACO algorithm performs better but has an underdeveloped front in the portion where the sum of changeover costs is minimized to the exclusion of total tardiness minimization. The reason for this is that the parameters necessary to achieve a good performance in the tardiness criterion and, most importantly, in the area where the best trade-offs between the two criteria exist are not optimal for minimizing the changeover costs by themselves. A potential solution to this problem is letting the PACO algorithm run for a number of iterations in single-criterion mode for each individual criterion, always choosing the best k solutions as guidance for the criterion that is being optimized, and later switching to the regular operation mode. Another possibility is empirically determining the best parameter configurations for minimizing the ξ individual criteria respectively and interpolating between these parameter vectors depending on the position of the active population $P \subset Q$. Evaluating these ideas is part of planned future work, as well as identifying new test instances or different problems to gauge the performance of the PACO algorithm, e.g. instances with a concave Pareto-optimal Front.

Another major focus of future work with the PACO algorithm on MCO problems is finding ways to limit the size of the super-population Q, keeping only those solutions that promise to guide the ants toward the broadest and best developed set of non-dominated solutions. As we explained above, restricting the size of Q is necessary both for a better development of the front as well as out of CPU-time considerations, since $O(|Q|)$ time is needed per iteration for maintaining the non-dominated set. A number of possibilities for eliminating superfluous solutions have been examined for Evolutionary Algorithms, and we plan to study which of these methods is best suited for a combination with PACO as well as devise new methods if necessary.

Besides the study of inherently multi-criterial problems, another motivation for the use of multi-objective optimization comes from trying to escape local optima in single-objective optimization, see [Knowles et al., 2001, Jensen, 2003]. In this so called multi-objectivization, the objective function f of the single criterion is decomposed into a sum $f = \sum_{i=1}^{\xi} f_i$, and the algorithm tries to optimize two objectives $f_A = \sum_{i \in A} f_i$ and $f_B = \sum_{i \in B} f_i$ with $A, B \subseteq [1, \xi]$ and $A \cup B = [1, \xi]$; the optimal solution to the underlying single-objective problem obviously lies on the Pareto-optimal front of the multi-objectivized version. We feel that this procedure could be especially useful for dynamic problems, where the fitness landscape changes and it is important for the algorithm to maintain a good overall solution quality (first objective) while at the same time trying to integrate the changes as efficiently as possible (second objective). It is conceivable that a dynamic problem is treated as a normal single-objective problem until a change occurs, upon which a second criterion is created which judges how well the change is integrated into the solutions constructed afterwards. After a number of iterations have passed or a given quality is met, the second criterion is dropped and the algorithm continues with the best solutions according to the primary objective. We will evaluate this scenario in future work.

Chapter 7

Conclusion

We have presented new ACO-based approaches for finding solutions to probabilistic and dynamic combinatorial optimization problems and an entirely new Ant Algorithm called Population-based Ant Colony Optimization (PACO), which enables us to efficiently solve a number of dynamic as well as multi-objective problems.

The approximation of the fitness function and the improved heuristic guidance on the PTSP enabled our algorithm to find better solutions than previous versions of ACO in less time. Furthermore, our algorithm outperforms the previously best heuristic for the PTSP, Hilbertsorting with 1-Shift, in nearly all cases, taking only a fraction of the time on the larger instances.

On the DTSP, the pheromone diversification techniques we proposed were successful in balancing exploration with exploitation after an unforeseen change to a problem instance, all partial-reset strategies performing better than restarting the algorithm or doing nothing in response to a change. Furthermore, the best local strategies which take the locality of a change into account outperformed the best global strategy.

Partially inspired by the idea to apply the KeepElitist strategy, which repairs the solution represented by the Elitist Ant in case of a change, to multiple solutions, we have introduced an new Ant Algorithm called Population-based Ant Colony Optimization (PACO). This algorithm no longer maintains an explicit pheromone matrix containing all previous updates, but rather keeps a relatively small population, typically less than 10 individuals, of solutions which implicate pheromone values for the ants to use as guidance. The versatility provided by this population of solutions let the PACO algorithm perform on par with state-of-the-art Ant Algorithm paradigms like the *MAX-MIN* Ant System on static problems, and performs almost as well as the specialized

pheromone reset strategies on dynamic problems. This is a good result considering that PACO is immediately applicable to a number of dynamic problems (DTSP and DQAP were evaluated) with only minimal adjustments.
The flexibility of the PACO approach becomes even more apparent for the case of Multi-Criteria Optimization. Here, the PACO algorithm utilizes the set of non-dominated solutions as a super-population from which it derives the active population to construct the pheromone information and aggregates the individual probability distributions corresponding to the separate criteria during the construction phase. With these augmentations, the PACO algorithm outperforms the Multi-Colony Ant Algorithm on the bi-criterial test-problem. In addition, the PACO algorithm can scale to an arbitrary number of criteria with only two pheromone matrices and is still capable of developing a front of non-dominated solutions, as shown on a four-dimensional test-instance.
Numerous possibilities for future work have been illustrated in the summarizing sections of the individual chapters. We believe the most promising to be the following:

- The pheromone diversification methods we have proposed have so far only been tested on the Dynamic TSP by the author and on the Dynamic Vehicle Routing Problem by third parties. We see two possibilities for future research based on these methods. First, we need to consider how the principle of pheromone diversification could be applied to other problem classes which are not variants of graph-routing, e.g. dynamic scheduling problems. Since the diversification is either uniform or based on heuristic or pheromone information, both of which usually exists when ACO is applied, this should be feasible. Second, another aspect lies in the application of partial pheromone resetting to counteract a stagnation of the search process on static problems, which takes place when the pheromone values on certain decisions have become too large to enable further exploration. Using the entropy of the pheromone rows as a guideline when to apply a partial pheromone reset could prolong the search process indefinitely.

- The greatest potential for future research is given by the framework of the PACO algorithm. The current implementation of the algorithm for mono-criterial, static problems is to start with an empty population and using the entire population as the active population from which the pheromone information is derived. Instead of starting with an empty population, it could make sense to use one or a number of different one-shot heuristics to seed the population with good solutions that guide the ants from the start. In addition, using only the best $k' < k$ solutions currently in the population while maintaining an Age-based update improves results e.g. for the TSP, where preliminary results show that PACO finds the optimal solution to `eil101`, a 101 customer instance from the [TSPLIB, 2003], in almost every run of 50000 iterations. Proceeding in this fashion would also be beneficial for the dynamic problems studied, since the search process while the instance is static is improved, and a larger population is available for repair, increasing the likelihood of good, repaired solutions.

The similarity between PACO and Evolutionary Algorithms also creates the opportunity to hybridize these two meta-heuristics into a common framework, where the most appropriate techniques from the two components could be identified and utilized for best results on a per-problem basis.

- For Multi-Criteria Optimization (MCO), there are also a number of interesting directions in which to proceed. The most important research topics are analyzing further methods for as well as the limits of aggregating the individual probability distributions during the construction phase, e.g. on problem instances with concave Pareto-optimal fronts. Furthermore, it should be explored how good parameter settings for the individual criteria are best combined to promote the exploration of the "center" of the front. Finally, to increase the efficiency of the search process, it is necessary to introduce a mechanism for limiting the size of the super-population Q, which is currently composed of all non-dominated solutions found, to a small yet evenly spread number of solutions which reduces the computational cost of maintaining the super-population and enables a better exploitation of the available solutions, ideally resulting in better performance.

Bibliography

S. V. B. Aiyer, M. Niranjan, and F. Fallside. A theoretical investigation into the performance of the hopfield model. *IEEE Transactions on Neural Networks 1*, pages 204–215, 1990.

R. H. Arnett. *American Insects: a handbook of the insects of America north of Mexico.* Van Nostrand Rehinhold, New York, 1985.

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

S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning,. Technical Report CMU-CS-94-163, Pittsburgh, PA, 1994.

S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, *The Int. Conf. on Machine Learning 1995*, pages 38–46, San Mateo, CA, 1995. Morgan Kaufmann Publishers.

J.J. Bartholdi and L.K. Platzman. An $o(n \log n)$ planar traveling salesman heuristic based on spacefilling curves. *Operations Research Letters*, 1:121–125, 1982.

A. Bauer, B. Bullnheimer, R.F. Hartl, and C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999)*, pages 1445–1450, 1999.

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.

G. Beni. The concept of cellular robotic system. In *Proceedings 1988 IEEE Int. Symp. on Intelligent Control*, IEEE Computer Society Press, pages 57–62, 1988.

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

G. Beni and J. Wang. Swarm intelligence in cellular robotic systems. In *Proceedings of the NATO Advanced Workshop on Robotics and Biological Systems*, 1989b.

D. Bertsimas and L. H. Howell. Further results on the probabilistic traveling salesman problem. *European Journal of Operational Research*, 65:68–95, 1993.

L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman prob lem. In J. J. Merelo Guervos et al., editor, *Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 883–892. Springer, 2002a.

L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO me taheuristic. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms*, volume 2463 of *LNCS*, pages 176–187. Springer, 2002b.

E. Bonabeau, M. Dorigo, and G. Théraulaz. *Swarm Intelligence.* Oxford University Press, 1999.

J. Bracken and J. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21:37–44, 1973.

J. Branke. Evolutionary algorithms for dynamic optimization problems - a survey. Technical Report 387, Insitute AIFB, University of Karlsruhe, February 1999.

J. Branke, C. Barz, and I. Behrens. Ant-based crossover for permutation problems. In E. Cantu-Paz, editor, *Genetic and Evolutionary Computation Conference*, volume 2723 of *LNCS*, pages 754–765. Springer, 2003.

J. Branke and M. Guntsch. New ideas for applying ant colony optimization to the probabilistic TSP. In *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2003*, number 2611 in Lecture Notes in Computer Science, pages 165–175. Springer Verlag, 2003.

J. Branke and H. Schmeck. Designing evolutionary algorithms for dynamic optimization problems. In S. Tsutsui and A. Ghosh, editors, *Theory and Application of Evolutionary Computation: Recent Trend s*, pages 239–262. Springer, 2002.

B. Bullnheimer, R. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. Technical report, POM Working Paper No. 10/97, University of Vienna, 1997.

B. Bullnheimer, R. 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, 1999.

W. Candler and R. Norton. Multilevel programming. Technical Report 20, World Bank Development Research Center, Washington D.C., 1977.

Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Plenum Pub Corp, 2002.

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.

S. A. Cook. The complexity of theorem prooving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, ACM, pages 151–158, 1971.

O. Cordón, I. Fernández de Viana, F. Herrera, and L. Moreno. A new aco model integrating evolutionary computation concepts: The best-worst ant system. In M. Dorigo, M. Middendorf, and T. Stützle, editors, *From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms ANTS'2000*, pages 22–29, Brussels, Belgium, 2000.

D. Corne, M. Dorigo, and F. Glover. *New Ideas in Optimization.* McGraw-Hill, 1999.

H.A.J. Crauwels, C.N. Potts, and L.N. van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10:341–359, 1998.

C. Darwin and A. Wallace. On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection. *Journal of the Proceedings of the Linnean Society, Zoology*, (3):45–62, August 1858. URL http://www.life.umd.edu/emeritus/reveal/pbio/darwin/dw01.html.

I. Das and J. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural Optimization*, 14(1):63–69, 1997.

Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley, 2001.

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.

O. Diessel, H. ElGindy, M. Middendorf, M. Guntsch, B. Scheuermann, H. Schmeck, and K. So. Population based ant colony optimization on FPGA. In *IEEE International Conference on Field-Programmable Technology (FPT)*, pages 125–132, December 2002.

E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, (1):269–271, 1959.

K. Doerner, J.W. Gutjahr, R.F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*. to appear.

K. Doerner, J.W. Gutjahr, R.F. Hartl, C. Strauss, and C. Stummer. Investitionsentscheidungen bei mehrfachen Zielsetzungen und künstliche Ameisen. In Chamoni and et al., editors, *Operations Research Proceedings*, pages 355–362, Berlin, Heidelberg, 2001a. Springer.

K. Doerner, R.F. Hartl, and M. Reimann. Are COMPETants more competent for problem solving? - the case of a multiple objective transporta tion problem. In L. Spector et al., editor, *Proceedings of the GECCO'01*, page 802ff, Berlin, Heidelberg, 2001b. Morgan Kaufmann.

K. Doerner, R.F. Hartl, and M. Reimann. Cooperative ant colonies for optimizing resource allocation in transportation. In W. Boers and et al., editors, *Proceedings of the EvoWorkshops 2001*, pages 70 – 79, Berlin, Heidelberg, 2001c. Springer.

M. Dorigo. *Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale (Optimization, learning and natural algorithms)* (in italienisch). PhD thesis, Dipartimento di Elettronica , Politecnico di Milano, Italien, 1992. 140 Seiten.

M. Dorigo and Gianni Di Caro. Ant colony optimization: A new metaheuristic. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1470–1477, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press. ISBN 0-7803-5537-7 (Microfiche).

M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 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.

F. Y. Edgeworth. *Mathematical Physics*. P. Keagan, London, England, 1881.

Euclid. *Elements*, volume 7, chapter Proposition 2. ca. -350.

C. J. Eyckelhof and M. Snoek. Ant systems for a dynamic tsp: Ants caught in a traffic jam. In M. Dorigo, Gianni Di Caro, and Michael Sampels, editors, *Ant Algorithms: Third International Workshop ANTS 2002*, number 2463 in Lecture Notes in Computer Science, pages 88–99, Brussels, Belgium, 2002. Springer.

S. Fenet and C. Solnon. Searching for maximum cliques with ant colony optimization. In Günther Raidl et al., editor, *Applications of Evolutionary Computing - EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*, number 2611 in LNCS, pages 236–245. Springer, 2003.

E. J. Fittkau and H. Klinge. On biomass and trophic structure of the central amazonian rain forest ecosystem. *Biotropica*, 5(1):2–14, 1973.

L. R. Ford and D. R. Fulkerson. *Flow in Networks*. Princeton University Press, 1962.

C. Gagné, M. Gravel, and W.L. Price. Scheduling a single machine where setup times are sequence dependent using an ant-colony heuristic. In *From*

Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms ANTS'2000, pages 157–160, 2000.

L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric tsps by ant colonies. pages 622–627, 1996.

L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

L. M. Gambardella, E. D. Taillard, and M. Dorigo. Ant colonies for the qap. *Journal of Operations Research Society*, 2(50):167–176, 1999a.

L.M. Gambardella, É. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, London, 1999b.

M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* Freeman, Oxford, UK, 1979.

A. H. Gee. *Problem Solving with Optimization Networks.* PhD thesis, Queen's College, Cambridge University, Cambridge, UK, 1993.

D. E. Goldberg. *Genetic Algorithms.* Addison-Wesley, 1989.

S. Goss, S. Aron, J.-L Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, (76):579–581, 1989.

R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Proc. Lett.*, (1):132–133, 1972.

P.P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

M. Gravel, W.L. Price, and C. Gagné. Scheduling continous casting of aluminium using a multiple objective ant colony optimization heuristic. *European Journal of Operational Research*, 143:218–229, 2002.

M. Guntsch, J. Branke, M. Middendorf, and H. Schmeck. Aco strategies for dynamic tsp. In M. Dorigo, M. Middendorf, and T. Stützle, editors, *From*

Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms ANTS'2000, pages 59–62, Brussels, Belgium, 2000.

M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In E.J.W. Boers et al., editor, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, number 2037 in Lecture Notes in Computer Science, pages 213–222. Springer Verlag, 2000.

M. Guntsch and M. Middendorf. Applying population based aco to dynamic optimization problems. In *Ant Algorithms, Proceedings of Third International Workshop ANTS 2002*, number 2463 in LNCS, pages 111–122. Springer, 2002a.

M. Guntsch and M. Middendorf. A population based approach for ACO. In S. Cagnoni and et al., editors, *Applications of Evolutionary Computing - EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, number 2279 in LNCS, pages 72–81. Springer, 2002b.

M. Guntsch and M. Middendorf. Solving multi-criteria optimization problems with population-based aco. In C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization, Second International Conference (EMO'03)*, number 2632 in LNCS, pages 464–478, Berlin, Heidelberg, 2003. Springer.

M. Guntsch, M. Middendorf, and H. Schmeck. An ant colony optimization approach to dynamic TSP. In L. Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 860–867. Morgan Kaufmann Publishers, 2001.

W. Gutjahr. Aco algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82(3):145–153, 2002.

S. Haykin. *Neural Networks*. Prentice Hall, 2nd edition, 1999.

J. Holland. *Adapdation in Natural and Articial Systems*. MIT Press, Ann Arbor, Michigan, 1975.

J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics 52*, pages 141–152, 1985.

J. P. Ignizio. *Goal Programming and Extensions*. Lexington Books, Lexington, Mass., 1976.

S. Iredi, D. Merkle, and M. Middendorf. Bi-criterion optimization with multi colony ant algorithms. In E. Zitzler and et al., editors, *Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01)*, number 1993 in LNCS, pages 359–372, Berlin, Heidelberg, 2001. Springer.

P. Jaillet. *Probabilistic traveling salesman problems.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.

P. Jaillet. Analysis of probabilistic combinatorial optimization problems in euclidean spaces. *MOR: Mathematics of Operations Research*, 18, 1993.

M. T. Jensen. Guiding single-objective optimization using multi-objective methods. In Günther Raidl et al., editor, *Applications of Evolutionary Computing - EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*, number 2611 in LNCS, pages 268–279. Springer, 2003.

A. Jezequel. Probabilistic vehicle routing problems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.

M. Karpinski. Polynomial time approximation schemes for some dense instances of np-hard problems. *Algorithmica*, (30):386–397, 2001.

J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence.* Morgan Kaufmann, San Francisco, CA, 2001.

J. D. Knowles, R. A. Watson, and D. W. Corne. Reducing local optima in single-objective problems by multi-objectivization. In E. Zitzler and et al., editors, *Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01)*, number 1993 in LNCS, pages 269–283, Berlin, Heidelberg, 2001. Springer.

T. C. Koopmans and M. J. Beckman. Assignmente porblems and the location of economic activities. *Econometrica*, (25):53–76, 1957.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc*, (7):48–50, 1956.

G. Laporte, F. V. Louveaux, and H. Mercure. A priori optimization of the probabilistic traveling salesman problem. *Operations Research*, 42(3):543–549, 1994.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The traveling salesman problem: A guided tour of combinatorial optimization*. Wiley, Chichester, 1985.

G. Leguizamón and Z. Michalewicz. A new version of ant system for subset problems. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 2, pages 1459–1464, Mayflower Hotel, Washington D.C., USA, 1999. IEEE Press.

S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, (21), 1973.

M. Lüscher. Air-conditioned termite nests. *Scientific American*, (205): 138–145, 1961.

C. Malandraki. *Time-dependent vehicle routing problems: Formulations, solution algorithms, and computational experience*. PhD thesis, Department of Civil Engineering, Northwestern University, 1990.

C. Malandraki and M. S. Daskin. Time-dependent vehicle routing problems: Formulations, properties, and heuristic algorithms. *Transportation Science 26*, pages 185–200, 1992.

V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4), 1999.

V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Trans. Knowledge and Data Engineering*, 5(11): 769–778, 1998.

Carlos E. Mariano and Eduardo Morales. MOAQ an ant-Q algorithm for multiple objective optimization problems. In W. Banzhaf and et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, volume 1, pages 894–901. Morgan Kaufmann, 1999.

D. Merkle and M. Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In *Proceeding of the EvoWorkshops 2000*, number 1803 in LNCS, pages 287–296. Springer Verlag, 2000.

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), pages 893–900. Morgan Kaufmann Publishers, 2000a.

D. Merkle, M. Middendorf, and H. Schmeck. Pheromone evaluation in ant colony optimization. In *Proceedings of the Third Asia-Pacific Conference on Simulated Evolution and Learning (SEAL2000)*, pages 2726–2731. IEEE Press, 2000b.

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.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1996.

R. Michel and M. Middendorf. An aco algorithm for the shortest common supersequence problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

D. Mitra, F. Romeo, and A. Sangiovanni-Vincetelli. Convergence and finite-time behavior of simulated annealing. *Adv. Appl. Prob.*, 18:747–771, 1986.

N. Monmarche. On data clustering with artificial ants. In Alex Alves Freitas, editor, *Data Mining with Evolutionary Algorithms: Research Directions*, pages 23–26, Orlando, Florida, 18 1999. AAAI Press. ISBN 1-57735-090-1.

R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. ODYSSEUS 2003 Second International Workshop on Freight Transportation and Logistics, May 2003.

S. Murthy. *Synergy in Cooperating Agents: Designing Manipulators from Task Specifications*. PhD thesis, Carnegie Mellon University, 1992.

G. Nicolis and I. Prigogine. *Self-Organization in Non-Equilibrium Systems*. Wiles & Sons, 1977.

V. Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, Switzerland, 1896.

H. N. Psaraftis. Dynamic vehicle routing problems. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, volume 16 of *Studies in Management Science and Systems*, pages 223–248. North-Holland, 1988.

H. N. Psaraftis. Dynamic vehicle routing: status and prospects. Annals of Operations Research, 61:143–164, 1995.

QAPLIB. http://www.opt.math.tu-graz.ac.at /qaplib/, 2003.

M. Reimann, K. Doerner, and R. Hartl. Analyzing a unified ant system for the vrp and some of its variants. In G. Raidl et al., editor, *Applications of Evolutionary Computing - EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*, number 2611 in LNCS, pages 300–310. Springer, 2003.

F. A. Rossi and I. Gavioli. Aspects of heuristic methods in the probabilistic traveling salesman problem, 1987.

G. Rudolph. *Convergence properties of evolutionary algorithms.* Kovac, Hamburg, 1997.

S. Sahni and T. Gonzales. P-complete approximation problems. *J. ACM*, (23):555–565, 1976.

M. J. Schniederjans. *Goal programming : methodology and applications.* Kluwer Academic Publishers, Boston, 1995.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

Kate Smith, Marimuthu Palaniswami, and Mohan Krishnamoorthy. Neural techniques for combinatorial optimization with applications. *IEEE Transactions on Neural Networks*, 9:1301–1318, 1998.

Kate A. Smith. Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS Journal on Computing*, 11 (1):15–34, 1999.

C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, August 2002.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, (35):254–265, 1987.

P. Souza. *Asynchronous Organizations for Multi-Algorithm Problems.* PhD thesis, Carnegie Mellon University, 1993.

T. Stützle and M. Dorigo. A short convergence proof for a class of aco algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4):358–365, 2002.

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, Aachen, 1998.

T. Stützle and H. Hoos. Max-min ant system and local search for combinatorial optimization problems. In *Proceedings Second International Conference on Metaheuristics, MIC '97*, Kluwer Academic, 1998.

T. Stützle and H. H. Hoos. The max-min ant system and local search for the traveling salesman problem. In *icec1997*, pages 309–314, 1997.

T. Stützle and H.H. Hoos. Max-min ant system. *Future Generation Computer Systems*, 16:889–914, 2000.

TSPLIB. http:// www.iwr.uni-heidelberg.de /groups /comopt /software /TSPLIB95 /index.html, 2003.

A. Vesel and J. Zerovnik. How good can ants color graphs? *Journal of computing and Information Technology - CIT*, 8:131–136, 2000.

K. von Frisch. *The Dance Language and Orientation of Bees.* Harvard University Press, 1967.

K. Weicker. *Evolutionary Algorithms and Dynamic Optimization Problems.* PhD thesis, Institut für Formale Methoden der Informatik der Universität Stuttgart, 2002a.

K. Weicker. Performance measures for dynamic environments. In J.J. Merelo, P. Adamidis, H.-G. Beyer, J.L. Fernández-Villaca nas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 64–73. Springer, 2002b.

Index