

California State University, San Bernardino

CSUSB ScholarWorks

Theses Digitization Project

John M. Pfau Library

2007

Parallelizing a nondeterministic optimization algorithm

Sammy Raymond D'Souza

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Theory and Algorithms Commons](#)

Recommended Citation

D'Souza, Sammy Raymond, "Parallelizing a nondeterministic optimization algorithm" (2007). *Theses Digitization Project*. 3084.

<https://scholarworks.lib.csusb.edu/etd-project/3084>

This Thesis is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

PARALLELIZING A NONDETERMINISTIC OPTIMIZATION ALGORITHM

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Sammy Raymond D'Souza

June 2007

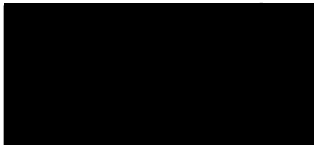
PARALLELIZING A NONDETERMINISTIC OPTIMIZATION ALGORITHM

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino


by
Sammy Raymond D'Souza

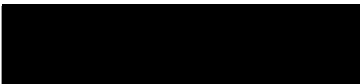
June 2007

Approved by:


Ernesto Gomez, Advisor, Computer Science

5/30/2007
Date


Keith Evan Schubert


Tong Lai Yu

© 2007 Sammy Raymond D'Souza

ABSTRACT

This thesis is an intersection of three fields - viz., distributed computing, combinatorial optimization and evolutionary computation.

The traditional view of algorithmic efficiency is that the total work done in a parallel execution (across all nodes) cannot be less than in its serial counterpart. However, in a parallel run, execution results are available at different times and in a different sequence. It is therefore possible for any one node to shortcut others and thereby achieve higher parallel efficiencies.

This research explores the idea that for certain optimization problems there is a way to parallelize the algorithm such that the parallel efficiency can exceed one hundred percent. Specifically, a parallel compiler, PC, is used to apply shortcutting techniques to a metaheuristic, Ant Colony Optimization (ACO), to solve the well-known Traveling Salesman Problem (TSP) on a cluster running Message Passing Interface (MPI). The results of both serial and parallel execution are compared using test datasets from the TSPLIB. Directions for future work are cited.

ACKNOWLEDGEMENTS

- God - without Him, I am nothing.
- My parents, Jacintha and Raymond D'Souza. Everything I am, everything I have achieved, I owe it to them.
- My wife, my pillar of support, my friend and philosopher, the love of my life – Priya D'Souza.
- My advisor, Dr. Ernesto Gomez, who gave me the freedom to explore, the latitude to go after huge goals, and the wisdom to pick well. I do not think I would have embarked on a thesis if it were not for him.
- My faculty at the California State University, San Bernardino.
- My committee member, Dr. Keith Evan Schubert for his words of wisdom and encouragement, for seeing me through difficult times, and also for undertaking to complete the L^AT_EX template for producing this thesis document. All coming generations of CSUSB Math and Computer Science students will be indebted to you for this.
- My committee member, Dr. Tong Lai Yu, for his patience, his guidance and his insight, and some very thought provoking conversations.
- My manager at E.S.R.I. www.esri.com, Clark Swinehart for always encouraging me to pursue my dreams and providing the trust and freedom to take those risks.

- Geraint Bundy, Ph.D., my friend, my colleague, my philosopher. His perspective and maturity have always kept me in good stead.
- My dear friends Fiona and Peter D'Souza, for everything you've done for us. It is our good fortune to have you as our friends.
- Max Manfrin, whom I have never met, for responding via email to a query sent at 4 a.m. on a sleepless night after the birth of my son, Ryan. I have learnt greatly from his writings.
- Marco Dorigo and Thomas Stutzle (I have met neither), for their work in the field of Ant Colony Optimization, and also for making the source for their ACO book freely available. This gesture alone enabled me to focus on my aspect of the thesis without getting lost in other intricacies and cut down tons of time that I would have needlessly spent.
- The wonderful organization I work for – E.S.R.I. (Environmental Systems Research Institute). Thanks to Jack and Laura Dangermond for nurturing this company, for fostering a culture of intellect, curiosity and caring and for the opportunity to work with some of the best minds in the world.
- And finally, my beloved son, Ryan Sammy D'Souza, who was born while I was doing this thesis. He is a truly a gift from God.

DEDICATION

To my parents – Raymond and Jacintha D'Souza

TABLE OF CONTENTS

<i>Abstract</i>	iii
<i>Acknowledgements</i>	iv
<i>List of Tables</i>	ix
<i>List of Figures</i>	x
1. Introduction	1
1.1 Background	1
1.2 Significance	2
1.3 Goals of the Thesis	3
1.4 Original Contributions	4
1.5 Organization and Structure of the Thesis	5
2. Ant Colony Optimization	6
2.1 Introduction	6
2.2 Stigmergy	7
2.2.1 Pheromone	7
2.2.2 The Double Bridge Experiments	8
2.3 The Ant Colony Optimization Metaphor	8
2.4 Ant Colony Optimization for the Traveling Salesman Problem	10
2.5 Ant Colony Algorithms	11

2.6	Ant Colony Implementations	14
2.7	Closing Notes	14
3.	<i>Distributed Computing - Parallel C and Shortcutting</i>	17
3.1	Introduction	17
3.2	Parallel C	17
3.2.1	Distributed Computing and Determinism	19
3.2.2	Processor Identification	19
3.2.3	Collective Communications	20
3.2.4	Range Operations	21
3.3	The Shortcutting Library	21
3.3.1	Streams	21
3.3.2	Overlapping	23
3.3.3	Shortcutting	25
3.4	Alternatives	26
3.5	Summary	28
4.	<i>Analysis</i>	29
4.1	Introduction	29
4.2	Analysis of Choices	30
4.2.1	Choice of Distributed Platform	30
4.2.2	Choice of Algorithm	31
4.2.3	Choice of Problem	32
4.3	Strategy for Parallelization	32
4.4	Results	35
4.5	Summary	37
5.	<i>Conclusions</i>	49

<i>Appendix A: THE ANT COLONY METAHEURISTIC</i>	51
<i>Appendix B: PORTING FROM C TO PC</i>	57
<i>Appendix C: SOURCE CODE</i>	60
<i>Appendix D: GLOSSARY OF TERMS</i>	197
<i>References</i>	201

LIST OF TABLES

2.1	The Ant Colony Optimization Metaheuristic	10
2.2	Applications of Ant Colony Algorithms	15
3.1	Parallel C Distilled	18
4.1	Instances Used in the Thesis	33
4.2	Program Execution Options	34
4.3	Reference for Reading the Graphs	36

LIST OF FIGURES

2.1	The Double Bridge Experiment	9
3.1	A User Defined Reduction Function	20
3.2	Streams Split At Logic Statements	22
3.3	Streams Merge At Points in the Control Flow Graph Where All and Only Processes From A Previous Split Must Pass	23
3.4	Streams Do Not Merge At Points in the Control Flow Graph Which May Be Bypassed By Processes from a Previous Split	24
3.5	Serial and Parallel Shortcutting	25
4.1	Shortcutting Results for 2 Ants with Nearest Neighbor Size 1	38
4.2	Shortcutting Results for 2 Ants with Nearest Neighbor Size 5	39
4.3	Shortcutting Results for 2 Ants with Nearest Neighbor Size 10	40
4.4	Shortcutting Results for 4 Ants with Nearest Neighbor Size 1	41
4.5	Shortcutting Results for 4 Ants with Nearest Neighbor Size 5	42
4.6	Shortcutting Results for 4 Ants with Nearest Neighbor Size 10	43
4.7	Shortcutting Results for 16 Ants with Nearest Neighbor Size 1	44
4.8	Shortcutting Results for 16 Ants with Nearest Neighbor Size 5	45
4.9	Shortcutting Results for 16 Ants with Nearest Neighbor Size 10	46
4.10	Summary of Shortcutting Results	47

1. INTRODUCTION

1.1 Background

In recent years, several major trends can be seen in the field of computation.

- Computing power has been growing exponentially in the last two decades. This has meant a significant increase in the ability of humans to solve large-scale problems using computers, yet there are ultimate physical limitations in this trend. Hardware designers must increasingly look for alternatives to increase performance other than clock cycle and circuitry design improvements.
- In the thrust of computer science to design, analyze, implement and evaluate algorithms and techniques to solve critical problems, Combinatorial Optimization problems have emerged as a prominent class. They are conceptually easy to model, (sometimes deceptively so), extremely challenging to solve in realistic terms, and yet hugely important in the scientific as well as the industrial world. The number of active researchers in this field is growing day by day.
- Parallel and distributed computing continues to merge into the mainstream, especially with the availability of multi-core machines on one end and distributed computed networks built from commodity units on the other.
- Nature inspired paradigms such as Ant Colony Optimization (ACO) [9] are be-

ing increasingly used to provide efficient solutions to problems in a wide variety of problems, ranging, from economic forecasting, stock market analysis and operations to bioinformatics.

1.2 Significance

Significant efforts have been made to the finding of exact solutions to some combinatorial optimization problems, using techniques such as dynamic programming, cutting planes, and branch and cut methods. Nevertheless, many hard combinatorial problems are yet to be solved exactly or in a reasonable amount of time. This is where good heuristic methods come in.

Heuristic techniques provide produce good-quality solutions quickly without necessarily guaranteeing of their optimality.¹ Metaheuristics, on the other hand, are higher level procedures that coordinate simple heuristics, such as local search, to find better quality solutions in a reasonably short computational time and limited resources.

Parallel (and distributed processing) can be considered as a further attempt to toward faster completion of an application, using a combination of algorithm design, programming language structure, and computer architecture. Any work that extends any of these areas becomes interesting and significant at the same time.

¹ This is acceptable because in practice a model is usually an approximations of reality itself.

1.3 Goals of the Thesis

The traditional view of parallel efficiency and speedup [4, 17, 3, 16] is that the total amount of work done in a parallel execution WILL ALWAYS exceed (or in the best case, equal) that done in the sequential counterpart. However, the results in a parallel execution are available at different times and in different sequences. Hence, in certain *irregular* problems, it is possible for one processor to attain an optimal value faster and thereby interrupt or *shortcut* the execution of the other processors. This thesis aims to validate this hypothesis.

Secondly, modern metaheuristics are probably one of the most promising research topics in optimization for the last two decades. These include simulated annealing, genetic algorithms, tabu search, GRASP (greedy randomized adaptive search procedure), ant colony optimization and their hybrids. This thesis aims to see:-

1. If they can be parallelized,
2. If Shortcutting techniques can be applied successfully to them, and
3. What, if any, patterns emerge from this.

Finally, the PLanguage paradigm of deterministic computing and matched send-receives, has been applied in conjunction with shortcutting techniques to a text-book problem. This thesis aims to see if Parallel C (PC) is ready for mainstream computation, and if regular code used to solve real-world algorithms and heuristics can be used ported to PC.

1.4 Original Contributions

The original contributions from this work are:

1. In previous work ([15]) shortcutting as an algorithmic improvement was applied to a specific problem that, by its nature, intrinsically favored Shortcutting techniques. In this thesis, we show that Shortcutting can not just be applied to another problem, but to an entire class of algorithms.²
2. The thesis continues the tradition of using the Traveling Salesman Problem (TSP)³ to show that PC as an approach is ready for mainstream.
3. The thesis applies shortcutting to an incredibly large variation of the ACO meta-heuristic; this makes it possible to see the influence of shortcutting on the underlying algorithm itself, as well as the parametric behavior of ACO-TSP
4. The thesis provides proof of how PC makes it extremely easy and straightforward to write parallel code. The code used for the experiments are a straight port to *PC* from the *C* code used to write the book [9]. The programmer can focus on the actual semantics of execution without worrying about the details of message passing.
5. Most importantly, the thesis shows comprehensively that using Shortcutting techniques, total work done in a parallel execution can be significantly less than that in the sequential equivalent; especially so when a good-enough solution is required.⁴

² It is proposed to present this work as a Journal submission or a Paper presentation at one or more of the peer reviewed conferences or publications.

³ the *Hello World!* equivalent of combinatorial problems

⁴ Improvements of several orders of magnitude have been observed in the latter case.

1.5 Organization and Structure of the Thesis

This thesis is an intersection of three fields - viz., distributed computing, combinatorial optimization and evolutionary computation. The remainder of this thesis is organized as follows.

Chapter 2 describes the Ant Colony Optimization metaheuristic while Chapter 3 describes the PLanguage-model of computation for distributed and parallel processing. Chapter 3 is divided into two sections; PC is described at length in section 3.2, while Shortcutting is discussed in section 3.3. In both these chapters alternative methodologies and approaches are also briefly explained for the sake of completeness.

Chapter 4 starts with a detailed analysis of the methodology and choices made in the thesis. Techniques for parallelization are described in Section 4.3. Next, some computational experiments and results are presented. The chapter concludes with an interpretation of the results.

Chapter 5 concludes the thesis with a brief summary followed by suggestions for improvement and directions for future research.

A set of useful Appendices is also provided for interested readers. Appendix A has a formal notation of the ACO metaheuristic; section A.2.3 describes the Traveling Salesman Problem interpreted as an ACO instance, while Section A.4 describes a specific variant of the ACO, viz., the MAX-MIN Ant Colony System. Appendix B provides details on porting *C* code to *PC*. Appendix C has the source code for the thesis, made available under the GPL. Finally Appendix D has a glossary of some important terms.

2. ANT COLONY OPTIMIZATION

2.1 Introduction

Ant Colony Optimization (ACO) was introduced in the 1990s as a novel nature-inspired method for the solving hard combinatorial optimization problems.[8] It takes inspiration from the foraging behavior of some ant species. These ants deposit pheromone on the ground in order to mark some favorable path that should be followed by other members of the colony. ACO exploits a similar mechanism for solving optimization problems. It has attracted the attention of an increasing numbers of researchers as well as practitioners and many successful applications are now available[9]. In this thesis, ACO has been used in conjunction with Parallel C and applied to the Traveling Salesman Problem.

The goal of this chapter is to introduce Ant Colony Optimizaiton and to survey its most notable applications. For a thorough treatment of the topic, interested readers are referred to [9], or to [22, 2, 5, 6, 11]. Section 2.2 provides an insight into the process of stigmergy. Section 2.3 discusses how this behavior is applied and mapped to combinatorial problems. The actual Ant Colony metaheuristic is also discussed in this section¹. Section 2.4 applies the ACO metaphor to the Traveling Salesman Problem. Variants of ACO are described in section 2.5, while ACO implementations are outlined

¹ Please also see Appendix A for a formal model of the metaheuristic

in Section 2.6. The chapter concludes with a discussion of other techniques in the field of Evolutionary Computation in section 2.7.

2.2 Stigmergy

The French entomologist Pierre-Paul Grassé first used the term *Stigmergy* to describe a peculiar form of communication in insect colonies [9]. He observed that some species of termites react to what he called ‘significant stimuli’, such as a significant source of food. These reactions in turn act as new stimuli for both the insect that produced them and for the other insects in the colony. i.e., “*workers are stimulated by the performance they have achieved*”. Stigmergy differs from other forms of communication in two significant ways – it is *indirect* and it is *local*. The insects do not ‘talk’ to each other, rather they exchange information by modifying their environment. Also, the information is available only locally. Another insect must ‘visit the region’ in which it was released (or its immediate neighborhood) to access it.

2.2.1 Pheromone

Many ant species use a substance called *pheromone* for Stigmergy. When searching for food, ants initially explore the area surrounding their nest in a random manner. As soon as an ant finds a food source, it evaluates it and carries some food back to the nest. During the return trip, the ant deposits a pheromone trail on the ground. The amount of pheromone deposited depends on the quantity and quality of the food. Other ants perceive the presence of pheromone and tend to follow paths

where pheromone concentration is higher. Also over time, pheromone also evaporates, so sub-optimal paths get ignored.

2.2.2 *The Double Bridge Experiments*

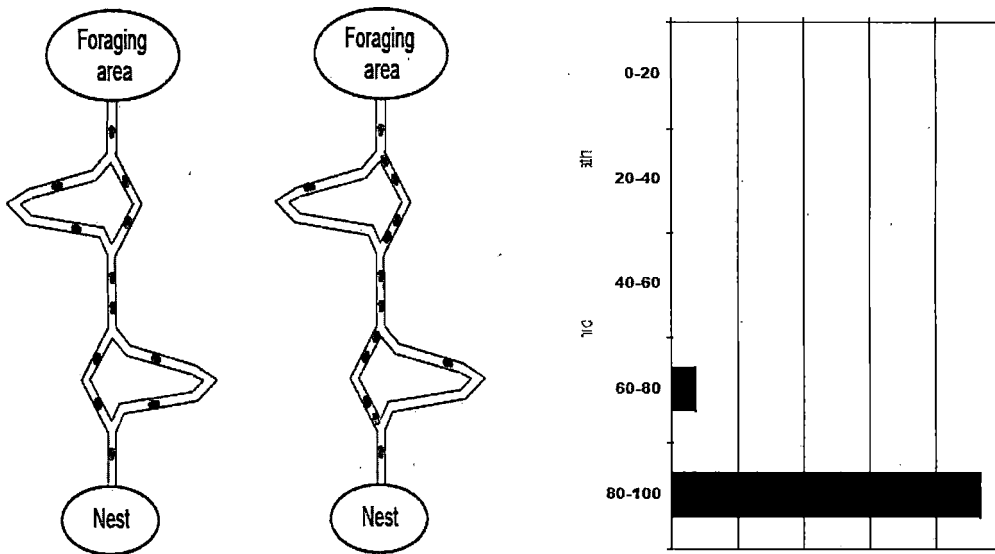
Indirect communication among ants via pheromone trails enables them to find shortest paths between their nest and food sources. This fact was verified in an experiment known as the “double bridge experiment”. A nest of a colony of Argentine ants was connected to a food source first by two bridges of equal lengths, and then by two of unequal lengths. In the first setting, a roughly equal number of ants ended up following both the paths. However, in the second case, the ants that chose the shorter bridge would reach the nest first. The short bridge would, therefore, receive pheromone earlier than the long one. This would increase the probability that further ants would select the shorter route over the longer one. Eventually the whole colony converges toward the use of the shorter bridge. See Figure 2.1 ²

2.3 *The Ant Colony Optimization Metaphor*

The capability of real ant colonies has inspired the definition of artificial ant colonies that can find approximate solutions to hard combinatorial optimization problems. This section outlines a framework for doing so. The key idea is that ACO can be applied to any algorithm that iteratively generates and evaluates paths on a weighted graph.³ The combinatorial problem is encoded on the graph such that each solution of the combinatorial problem corresponds to at least one path on the graph. In the

² Adapted from [7]

³ In this sense ACO really is a meta-algorithm



(a) Ants start exploring. (b) Eventually most ants choose the shortest path
(c) Distribution of percentage of ants.

Fig. 2.1: The Double Bridge Experiment

encoding, weights are associated with the edges of the graph. These weights are assigned are such that the cost of a path, i.e. the sum of the weights of its composing edges, equals the cost function of the combinatorial problem for the associated solution. The goal of ACO then becomes finding a path of minimum cost. To find such a path, a number of paths are generated in a Monte Carlo fashion. Stigmergy then plays in – the cost of such paths is used to bias the generation of further paths. This process is iterated and more and more information is gathered on the graph to eventually produce a path of minimum cost.

The basic ACO metaheuristic is shown in Table 2.1. It consists of four algorithmic components in a ‘ScheduleActivities’ construct. The construct does not specify how these activities are scheduled and synchronized; this decision is left to the algorithm designer. The interested reader is referred to [2] or to Appendix A for

```

while Termination Conditions not met do
  ScheduleActivities
    ConstructAntBasedSolutions
    ApplyLocalSearch optional
    PheromoneUpdate
    DaemonActions optional
  end ScheduleActivities
end-while

```

Tab. 2.1: The Ant Colony Optimization Metaheuristic

a formal framework and a rigorous, theoretical foundation. Metaphorically, the ACO metaheuristic can be visualized as follows: The generation of a path corresponds to the walk of an ant. At each node, the ant stochastically selects the following one based on the problem specific constraint and also on the basis of local information called pheromone trail. In turn, the pheromone trail is modified by the ants themselves in order to bias the generation of future paths toward better solutions. The central component of the ACO algorithms is the pheromone model.

2.4 Ant Colony Optimization for the Traveling Salesman Problem

This describes in simple terms how the generic ACO framework is applied to the well known traveling salesman problem (TSP). In the TSP, a set of cities is given along with distances between each of them. The goal is to find the shortest tour that allows each city to be visited exactly once. Formally, the objective is to find a Hamiltonian tour of minimal length on a fully connected graph.

In ACO, a number of artificial ants simulate their natural counterparts by moving on a graph. The graph is constructed to encode the problem itself: each vertex represents a city and each edge represents a connection between two cities. A ‘pheromone’ variable is associated with each edge and can be read and modified by ants. The algorithm proceeds iteratively. Several artificial ants independently build solutions by walking from vertex to vertex. If a terminating condition is not reached, the iteration repeats. The TSP problem constraint is that any vertex already visited must not be visited again; the ant achieves this by keeping a memory of vertices already visited in the walk.⁴

Now, at each step of the solution construction, an ant selects the following vertex to be visited according to a nondeterministic mechanism that is biased by the pheromone. If an ant is at vertex i and vertex j has not been previously visited, then the ant will select j as the next vertex with a probability that is proportional to the pheromone associated with edge (i, j) . At the end of the iteration, the pheromone values are modified on the basis of the quality of the solutions constructed by the ants. These values are used to bias ants in future iterations to construct solutions similar to the best ones previously constructed.

2.5 Ant Colony Algorithms

This section describes the various variants of ACO algorithm that have evolved over the years. A chronological history of ACO Algorithms is maintained at the website at <http://iridia.ulb.ac.be/~mdorigo/ACO/index.html>.

⁴ In this sense it differs from a real ant.

Ant System(AS) AS was the first ACO algorithm. It was applied to TSP and found encouraging results, but was not as good as other state-of-the-art algorithms for TSP. Ant System inspired and spawned several algorithms and extensions; almost all successors of ACO are direct extensions of AS.

Elitist Ant System(EAS) M. Dorigo's Ph.D. thesis; the EAS added a *daemon action* to the metaheuristic. Arcs belonging to the best tour since the start of the algorithm, T^{bs} , were provided additional reinforcement of pheromone deposit.

Ant-Q This was an ant algorithm intended to create a link between reinforcement learning and Ant Colony Optimization. Computational experiments showed that some aspects of Ant-Q, in particular the pheromone update rule, could be strongly simplified without affecting performance. It is for this reason that Ant-Q was abandoned in favor of its successor the simpler and equally good ACS.

Ant Colony System (ACS) ACS differs from AS in three areas.

- The accumulated search experience of the ants is exploited more strongly via a more aggressive action choice rule.
- Pheromone deposit and evaporation is restricted to arc of the best-so-far tour.
- When an ant moves from city i to j it 'removes' some pheromone from the arc to increase exploration of alternative paths.

Max-Min Ant System (MMAS) One of the best performing variants, MMAS is also used in this Thesis. It differs from AS in four respects.

- Only the iteration-best-ant or the best-so-far ant is allowed to deposit pheromone. See 5 for the update rule, thus strongly exploiting best tours found.
- Pheromone trail values are bounded to an interval $[\tau_{min}, \tau_{max}]$ to avoid stagnation.
- Pheromone trails are initialized to τ_{max} at the start and a small evaporation is applied, increasing the exploration of tours at the start of the search.
- Pheromone trails are reinitialized each time the system approaches stagnation or when no improved tour has been generated for a certain number of iterations.

Rank-based Ant System (AS_{rank}) AS_{rank} is another improvement over AS. Each ant deposits an amount of pheromone that decreases with its rank. Additionally, as in EAS, the best-so-far ant always deposits the largest amount of pheromone in each iteration.

Approximate Nondeterministic Tree Search (ANTS) ANTS is an ACO variant that exploits ideas from mathematical programming.⁵ ANTS computes lower bounds on the completion of a partial solution as the heuristic information used by each ant during solution construction. It is also an exact algorithm that can be extended from ACO to branch & bound.

Hyper-Cube Framework for ACO This variant also has roots in mathematical programming. Just as solutions of combinatorial optimization problems are represented as binary vectors, the hyper-cube ACO automatically rescales pheromone

⁵ Incidentally, it was serendipity that led to the choice of Ant Colony Optimization for this thesis as the metaheuristic to be parallelized. A keyword search on *Nondeterministic* led to ANTS which led to ACO.

values to always lie in the interval $[0, 1]$. A solution to a problem then corresponds to one corner of an n -dimensional hyper-cube where n is the number of decision variables.

2.6 *Ant Colony Implementations*

Many implementations of the ACO metaheuristic are available and have been applied to many different types of combinatorial optimization problems. Table 2.2, adapted from [7] is a listing of the problems that the ACO metaheuristic has been applied.

2.7 *Closing Notes*

For the sake of completeness, it must be noted that many other metaheuristics besides ACO are also available. Some of the more successful ones in the literature are Tabu Search(TS), Guided Local Search(GLS), Iterated Local Search(ILS), Greedy Randomized Adaptive Search Procedures(GRASP), Evolutionary Computation(EC) and Scatter Search. The interested reader is referred to Chapter 2 in [9].

On the other hand, for being a young metaheuristic, ACO has raised a lot of interest in the scientific community. There are now available numerous successful implementations of the ACO metaheuristic applied to a wide range of different combinatorial optimization problems. There is also a successful biannual workshop (ANTS From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms; <http://iridia.ulb.ac.be>) where researchers meet to discuss the properties of ACO and other ant algorithms both theoretically and experimentally. From the

Problem Type	Problem Name
Routing	Traveling salesman Vehicle Routing Sequential ordering
Assignment	Quadratic assignment Graph coloring Generalized assignment Frequency assignment University course timetabling
Scheduling	Job shop Open shop Flow shop Total tardiness Total weighted tardiness Project scheduling Group shop
Subset	Multiple knapsack Max independent set Redundancy allocation Set covering Weight constrained graph tree partition Arc-weighted l -cardinality tree Maximum clique
Other	Shortest common subsequence Constraint satisfaction 2D-HP protein folding Bin packing
Machine learning	Classification rules Bayesian networks Fuzzy systems
Network routing	Connection-oriented network routing Connectionless network routing Optical network routing

Tab. 2.2: Applications of Ant Colony Algorithms

theory side, researchers are trying either to extend the scope of existing theoretical results, or to find principled ways to set parameters values. From the experimental side, most of the current research is in the direction of increasing the number of problems that are successfully solved by ACO algorithms, including real-word, industrial

applications. These applications comprise two main fields - *NP*-Hard problems, and dynamic graph problems. Finally, this thesis attempts to parallelize ACO-MMAS using ParallelC and shortcutting techniques.

3. DISTRIBUTED COMPUTING - PARALLEL C AND SHORTCUTTING

3.1 Introduction

Much of parallel computing in scientific computation has been limited to static parallelism [20], the most common form often being the Single Program Multiple Data (SPMD) model. The PLanguages, however, are an explicitly parallel notation that allow for a very elegant expression of parallel algorithms. This chapter presents a quick overview of the PLanguage model.

3.2 Parallel C

The PLanguages which consist of PFortran and PC extend their regular declarative counterparts, Fortran and C, respectively. Thus all knowledge and experience with Fortran or C just carries over to the corresponding PLanguage. Table 3.1 is a distillation of the essential differences between PC and C; each row points to a major concept or to a noteworthy idea.

PC extends C with a small set of operations. The duality of the send and receive operations in the message passing paradigm is encapsulated in an infix operator and in reducing functions. The PC compiler generates calls to a system-dependent library from user-supplied expressions containing communication operators, particularly those listed in Table 3.1

A key addition to the PC language is the @ operator. @ indicates a variable ‘at’ a processor. For instance, $x = y@0$ assigns the value of variable y at processor 0, to x at every processor, effectively a *broadcast*. Mixed expressions such as $x = y@p + z$; are also legal. Multiple uses of @ are also permissible, however only in assignment statements, or reduction operations. @ also has the highest precedence among the binary operators. So, $x = y@b + 1$ is not the same as $x = y@(b + 1)$. Also, $x@a + 1 = y@b$ is meaningless, but $x@(a + 1) = y@b$ is legal. @ is also non-associative; $x = a@b@c$ is illegal, but $x = a@(b@c)$ and $x = (a@b)@c$ are not.

One of the snippets above introduced the concept of a fusion. In the serial world, the statement $x = y$ is a memory move of the value at y to that at x . Extending this idea, in PC, the fusion statement, $x@p = y@q$ is combination send-receive statement that assigns the value of y at node q to the variable x at node p . For more

Snippet	Feature / Comment
$x = y@q;$	@ Operator
$x@p = y@q;$	Fusion statement
$myProc$ and $nProc$	Reserved Variables for Processor Identification
$sum = +\{y\};$	Reduction Operation - summation
$x = max\{y\};$	Reduction Operation - user defined given a function such as <pre>int max(int x, int y) { return (x ≥ y ? x : y); }</pre>
$x@(a : b) = y@c;$	Range Operations
$A(0 : 4)@a = B(5 : 9)@b;$	

Tab. 3.1: Parallel C Distilled

details, refer to [20], or to the respective manuals [12, 18]¹. The fusion statements hide the complexity of the send and receive calls, making it easier, elegant and more intuitive to write parallel code. More importantly, the fusion assures the semantics of implicit, albeit one-sided, synchronization; p must wait for q but not vice-versa. This concept is explored in more detail in section 3.3.

3.2.1 Distributed Computing and Determinism

The two classic models of distributed computing, viz., shared memory and message passing, each have their strengths and merits. Yet, they are also non-deterministic by design. In contrast, the PLanguage model, is intrinsically deterministic. This determinism is largely achieved by matching sends and receives. PC, in its current implementation, is based on MPI. Hence, if necessary, it is still possible to reach beneath the hood and make raw calls to MPI - thereby offering the best of both worlds.

3.2.2 Processor Identification

The PLanguages use reserved variables $myProc$ and $nProc$ to allow for processor specific numbering and identification. The variable $nProc$ denotes the total number of nodes. Nodes are numbered from 0 to $nProc - 1$. An individual node identifies itself using the variable $myProc$. Thus any fusion statement or any code expression that includes the $myProc$ variable will cause different processes to be in different states. A common usage is to create a *master-slave* configuration based on the value of $myProc$; 0 denotes the master, all other values represent slave processors.

¹ [12, 18] describe the inner workings and also have numerous examples

3.2.3 Collective Communications

In the Message-Passing paradigm, collective operations are those that work with groups of data distributed over all nodes [16]. They could be a *broadcast* from a single node to all other nodes, or a combination (also sometimes called a *reduction*) of certain data from all nodes. Frequently, the results of combines are also broadcast.

While it is straightforward to write collective operations using sends and receives, the system frequently provides constructs for the same. These constructs can be optimized by the system or by individual implementations. For instance, MPI provides functions such as MPI.Scatter, MPI.Gather, MPI.Bcast and MPI.Reduce. In particular, MPI.Reduce can accept from a list of pre-defined operations such as MPI.MAX, MPI.MIN, MPI.PROD and so on.

The PLanguages take this a step further by providing a scheme for intrinsic as well as *user defined* reductions operations. For instance, $sum = +\{y\}$; is a succinct notation of a summation. Similarly, $u = max\{v\}$; would assign the maximum value of v to the variable u in every processor, provided the user defined function max were defined as in Figure 3.1

```
int max(int x, int y)
{
    return ( x > y ? x : y );
}
```

Fig. 3.1: A User Defined Reduction Function

A key thing to note is the subtle change in the memory model between MPI and PC. In MPI, the name of the variable bears no intrinsic meaning in a commu-

nication context - the messages sent and received are just data. However, in the PLanguage model, the notation $a@b$ really *IS* the variable a guarded by processor b . In fact, PC assumes that every processor knows the names of the variables in all other processors [12]. This means the same code must execute on all processors; ergo., an SPMD execution.

3.2.4 Range Operations

The PLanguages borrow the “.” notation from Fortran to denote Ranges of processors as well as sections of arrays. For instance $x@(a : b) = y@c$ would assign the value of y at processor c to that of x in processors a through b . The RHS of the expression could also have ranges, in which case there would be a 1:1 mapping. These operations hold on arrays too. Thus it is legal to write $A(0 : 4)@a = B(5 : 9)@b$. For more detailed examples, see [20].

3.3 The Shortcutting Library

SOS refers to *ipStreams*, *Overlapping* and *Shortcutting*.² It is a function library, accessible from C or Fortran, that supports “process groups in parallel computation, an overlap communications optimization protocol, irregular problems, a form of nondeterministic computation” [13]

3.3.1 Streams

In an SPMD execution, a program can be considered to be a set of processes P that start together, executing the same code. See Figure 3.2.

² This section on SOS is heavily abstracted from [15] and [14].

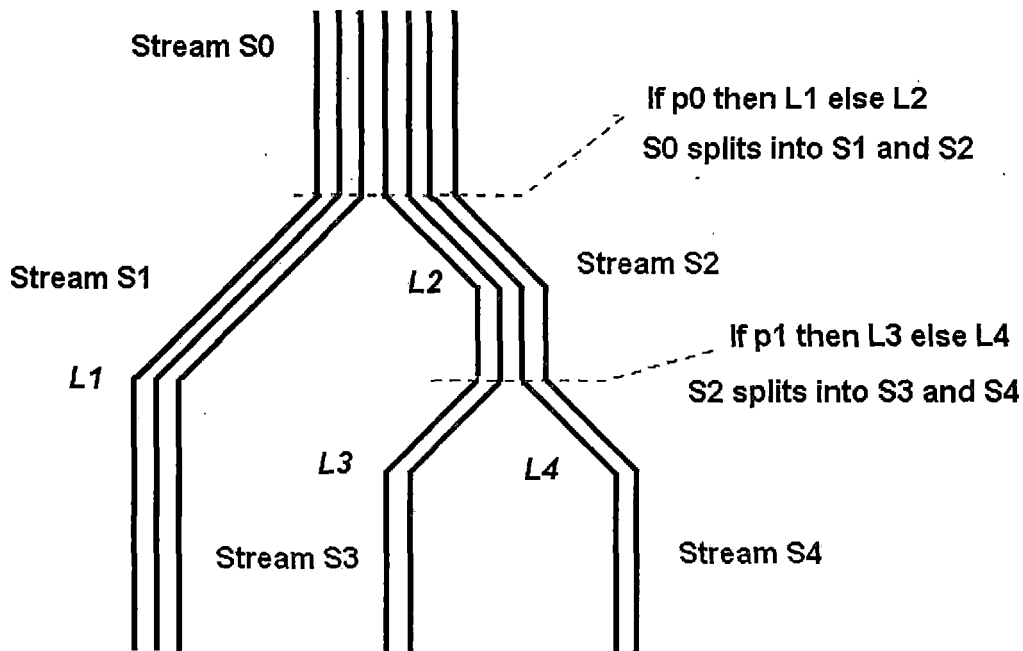


Fig. 3.2: Streams Split At Logic Statements

Now, if the code has some logic that will not execute identically on branches, (say, because of an if statement dependent on the value of *myProc*) then control flow for some subset P_1 of P will follow one branch of code, whereas still other subsets $P_2 \cdots P_N$ of processes may follow one or more different branches. This is shown in Figures 3.3 and 3.4. Note that at the end of the program all subsets merge back to P .

These subsets are called ipStreams (Implicit Process Streams) or Streams for short³. The salient features of streams are:

- All processes within a stream execute the same code.
- At some points in the control flow graph, streams may merge.

³ Figures 3.2 through 3.4 have been adapted from [13]

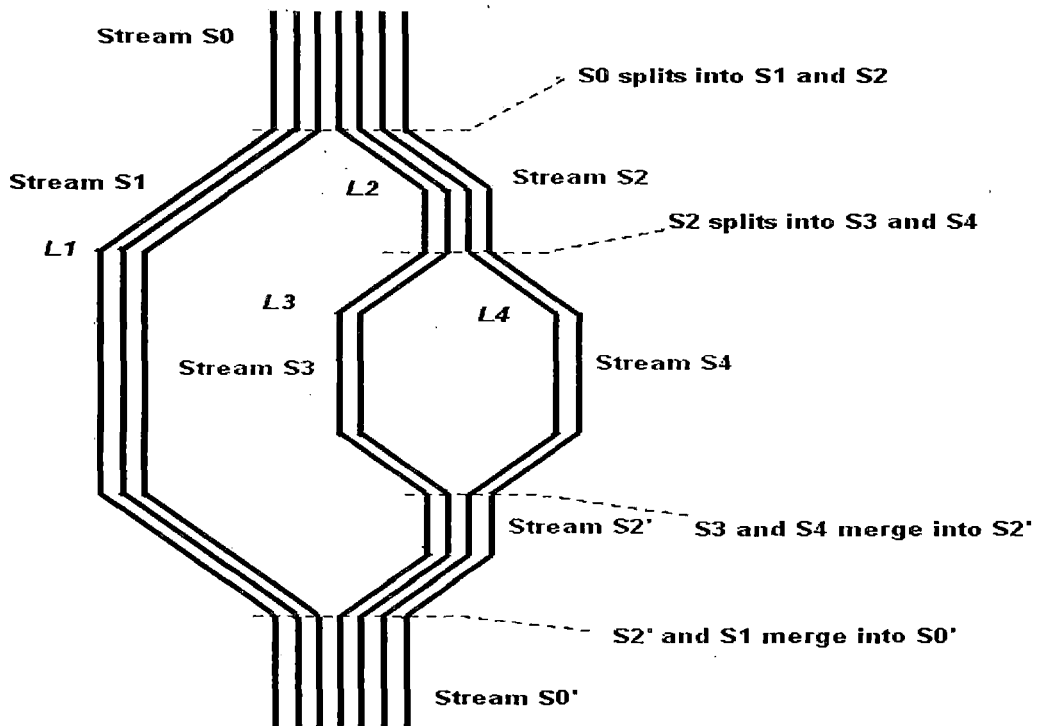


Fig. 3.3: Streams Merge At Points in the Control Flow Graph Where All and Only Processes From A Previous Split Must Pass

- If inter-process communication is limited to processes in the same stream, then analysis to ensure determinism and freedom from deadlock is relatively straightforward.

3.3.2 Overlapping

Overlapping uses two constructs: a protocol and a runtime system, to schedule data transfers between processes in the overlap between definition / redefinition at a producer process and use / reuse at a consumer process. This process is described in brief here, for a detailed description the reader is referred to [14, 15]. Any execution model with different control paths executing simultaneously will have large synchrono-

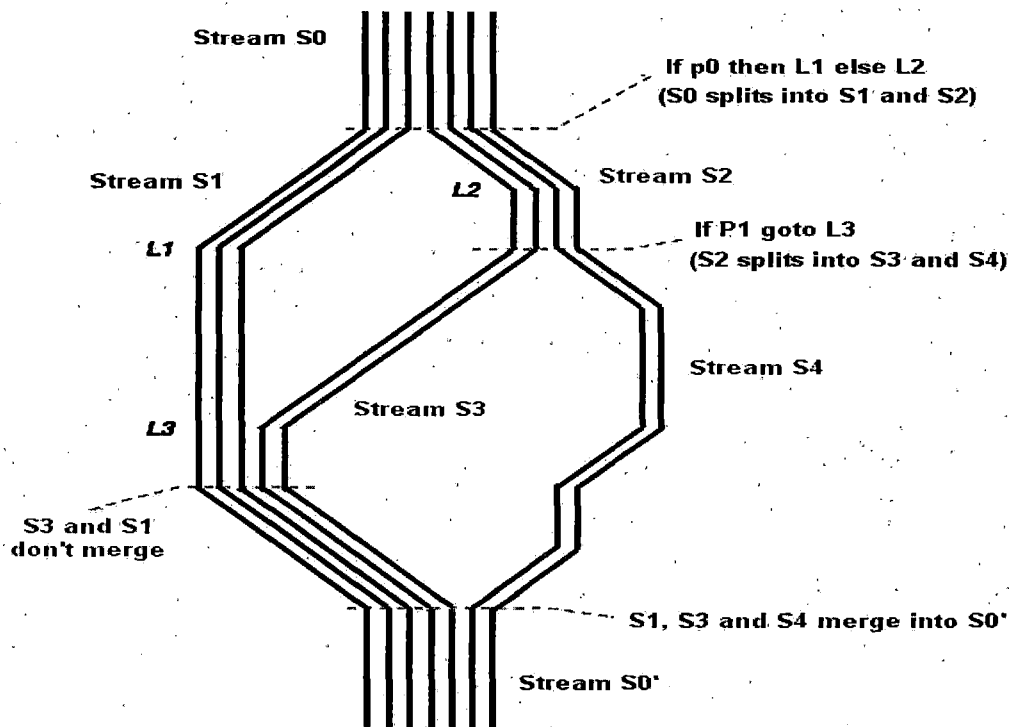


Fig. 3.4: Streams Do Not Merge At Points in the Control Flow Graph Which May Be Bypassed By Processes from a Previous Split

nization lags. This is particularly true where streams that from different execution paths merge. Moreover, data distribution or algorithmic requirements may also force processes to wait for others at synchronization points.

With standard program analysis methods, it is possible within a stream, to determine where data is generated and where it is used. SOS uses this information in the overlapping protocol. Communication initiation is done as close as possible to data definition and communication termination (and blocking if needed) as close as possible to data usage. Overlapping additionally tries to schedule communications dynamically minimizing the need for synchronizing waits.

3.3.3 Shortcutting

Shortcutting allows parallel algorithms to be inherently better than their serial counterparts in that they do less work to reach the same solution. It also dynamically

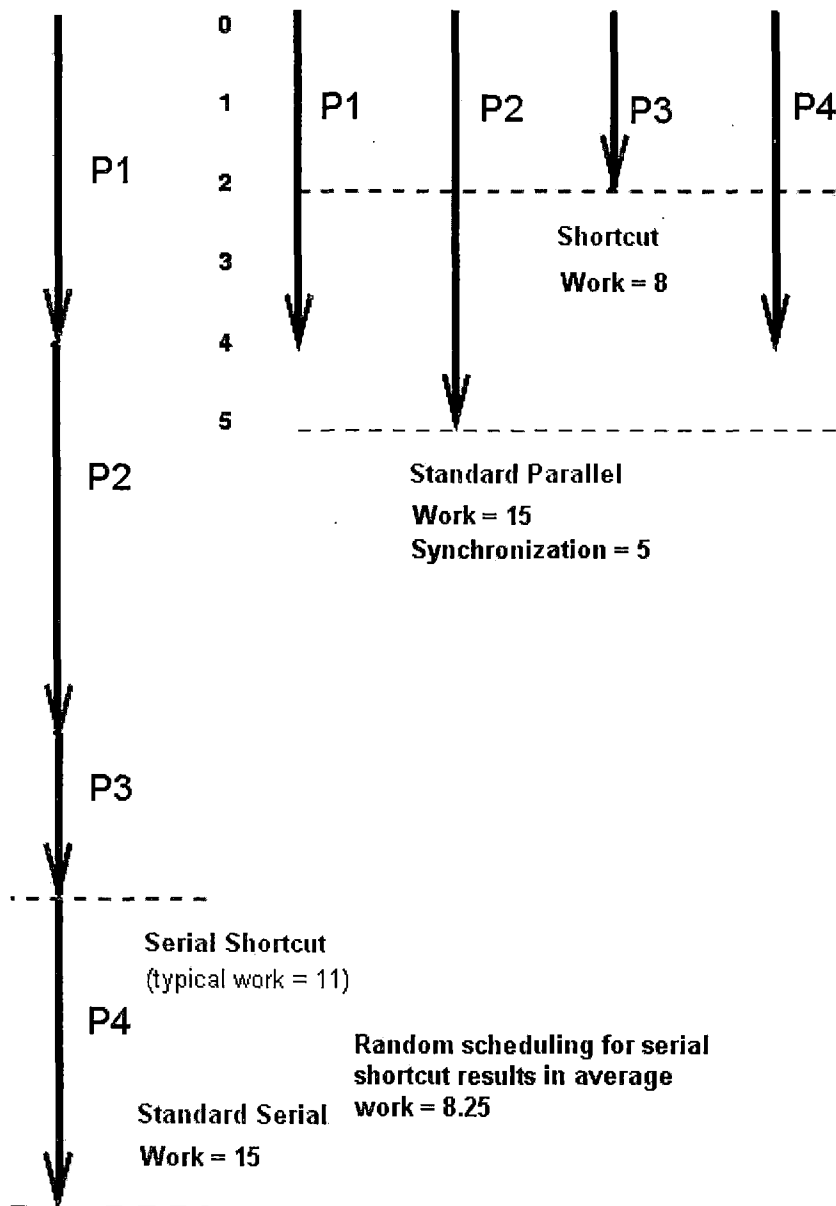


Fig. 3.5: Serial and Parallel Shortcutting

load balances parallel execution, since all processes will take the same clock time when shortcut. [15], from which Figure 3.5 was adapted, conjectured that *in some 'irregular' problems, a parallel execution may give dramatic improvement over its serial counterpart.* That statement is the basis of this thesis.

For instance, suppose the computation involves testing various cases and terminates as soon as any one case passes a test (say, by finding a good enough answer). In a parallel execution, as soon as one process finds a “passing” case, it can “short-cut” the other processes by telling them to quit. The parallel version of the algorithm may do less work if it finds the answer on one node before finishing the computation on other nodes. In [15], this technique was applied to a minimization problem, that seemed catered towards shortcutting.

3.4 Alternatives

The use of PC as a parallel compiler and Shortcutting as a speedup technique and was a natural choice.⁴ This section lists some other Parallel Compilers and some Shortcutting-like approaches.

1. Parafrase: From the UIUC (<http://www.csr.d.uiuc.edu/parafrase2>), Parafrase-2 is a “vectorizing/parallelizing compiler implemented as a source to source code restructurer. It consists of Fortran/C front-ends and passes for analysis, transformation and code generation.”
2. Bulldog compiler: The Bulldog compiler is “sophisticated compiler for a very specific problem: exploiting the parallelism in a VLIW machine for scientific

⁴There is a PC Research Group at California State University, San Bernardino. See <http://csci.csusb.edu/egomez/PC/PC.html>

programs in Fortran”. It is based on the premise that a VLIW machine can be generated along with the compiler, which itself is based on three techniques: trace scheduling, memory-reference disambiguation, and memory bank disambiguation. See [10] for more details.

3. Cilk2c compiler: Cilk2C is a source-to-source compiler (including a type-checking preprocessor) that translates Cilk code into C code. Cilk itself, developed at the MIT, is a multithreaded parallel programming language based on ANSI C. It extends C using three basic keywords, `cilk`, `spawn` and `sync`. The latest version of Cilk is available at <http://supertech.csail.mit.edu/cilk/index.html>
4. F90 / High Performance Fortran (HPF): High performance Fortran extends F90 to support data parallel programming. Compiler directives allow programmer specification of data distribution and alignment. New compiler constructs and intrinsics allow the programmer to do computations and manipulations on data with different distributions.
5. Nagging: [21] introduced a parallel search-pruning technique called *nagging* as a means of coordinating the activity of a number of different search procedures. In Nagging, search-based problem solvers compete in parallel to solve parts of a particular problem instance; each contributing to advancing the search wherever it is the most effective. A prototype implementation was developed for first-order theorem proving.

3.5 Summary

This chapter touches briefly on some of the features of PC. PC makes two very important contributions to the world of parallel computation. It makes it extremely easy and straightforward to write parallel code. The programmer can focus on the actual semantics of execution without worrying about the details of message passing. At the same time, it ensures that data movement is done in a correct and deterministic manner. One of the contributions of this thesis is to show how easy it is to take a non-trivial chunk of sequential code and port it for parallel execution.

Overlapping and shortcutting are two techniques that, separately and in conjunction, can be used to speedup parallel computation in irregular (nondeterministic) programs. Overlapping takes advantage of the regions of code between data definition and data use in different processes and "lightens" the synchronization load by spreading the work of synchronization over a period of time. Where applicable, the shortcutting method allows the implementation of parallel algorithms that are inherently better than their serial counterparts in that they do less work to reach the same solution.

4. ANALYSIS

4.1 *Introduction*

This thesis has been an interesting study, as it meandered through widely disparate fields. The preceding chapters laid the foundation and background for Ant Colony Optimization, Parallel C and Shortcutting. This chapter brings it all together. Through the course of this thesis, several decisions points were reached, and accordingly paths chosen. Section 4.2 provides a detailed look at these decisions and the rationales behind them. The consequences of these choices are described. Section 4.3 looks at strategies for parallelization of ACO. Section 4.4 describes the experimental results of the thesis. First the methodology used for testing is explained, next the input TSP instances and their optimal values, next: the various possible inputs to the program itself. This is followed up by a host of figures and graphs, analyzing each one of the possible inputs at every stage. The chapter ends with a look at the future, and directions for such endeavors.

4.2 *Analysis of Choices*

4.2.1 *Choice of Distributed Platform*

PC was a natural choice for basing all implementations on. This was largely due to the fact that significant work on PC was being done here [14]. At the same time, PC presented significant advantages in terms of its elegance and simplicity. Being an extension of the *C* language meant that effort could be focused on distributed computing, without worrying about the intricacies of message passing or shared memory or other constraints normally associated with distributed computing. Chapter 3.2.1 also describes the determinism and synchronization afforded by PC.

Results

The prime benefit from this choice was that it allowed the freedom for a much broader problem to be tackled. In hindsight, this thesis was able to study the behavior of a class of algorithms, ACO (see Chapter 2) rather than just one or two more algorithms and yet remain within reasonable scope solely because of this choice. Along the way, some issues were found with the PC compiler itself. Some of these were resolved, and some had to be worked around. As a side-effect, the documentation accompanying the PC distribution was enhanced and will keep future development in good stead.

4.2.2 *Choice of Algorithm*

PC in general, and shortcutting in particular, had been applied to a specific problem [14], viz., downhill simplex minimization. The challenge was to find another algorithm or two that would show the applicability of the approach described in [14]. However, an analysis of the mechanics of shortcutting revealed that it could also be applied to problems where a good enough solution was also acceptable. A literature survey with an enhanced scope revealed the rapidly growing field of Ant Colony Optimization. More details about ACO are in Chapter 2. Practitioners were also implementing ACO algorithms [9] to solve combinatorial optimization problems. The application of PC to a such a field became more of a challenge.

Results

As with any work of research, this choice made a huge impact in terms of scope. The effort in learning and understanding ACO was significant; this was, however, offset to a certain extent by the elegance of PC, and the fact that the ACO authors had basic C implementations of certain algorithms under the GPL license. Another outcome of this choice is that there are now interesting directions for future research. Indeed, it is possible that better efficiencies may be obtained by applying shortcutting to algorithms in ACO other than the TSP, for instance AntNet - an ACO algorithm designed for the network routing problem.

4.2.3 Choice of Problem

The Traveling Salesman Problem (TSP) has been extensively studied in the literature. Moreover, it is an important NP-hard optimization problem; it can be described easily, yet it has several diverse applications.³ While any optimization problem would have been sufficient, a successful implementation of shortcutting with TSP, i.e., one with good performance, would serve as proof of its usefulness. It also was the first problem addressed in almost all of the ACO variants and continues to be a topic of research⁴ - making it a natural candidate for this work.

Results

A huge benefit of this choice was the availability of TSPLIB. TSPLIB is a benchmark library for the TSP and related problems and is accessible via and maintained at <http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95>. The best part about the choice of TSP was that the basic algorithm is straightforward and not “obscured by too many technicalities”[9], thus allowing the thesis to focus on the shortcutting aspect.

4.3 Strategy for Parallelization

Ants in a colony are independent and can operate asynchronously, thus making ACO particularly well suited for parallel implementations. Indeed, there have been many attempts to propose parallel ACO algorithms in the literature[24]. They are usually classified by their parallel grain⁵ as either coarse-grained or fine-grained

³ For instance, drilling holes on printed circuit boards, or positioning of X-ray devices.

⁴ See [1] for a application of ACO to PTSP (Probabilistic TSP) and Uncertainty

⁵ The relationship between computation and communication

Instance Size	Instance Name	Optimal Tour Length
22	ulysses22.tsp	7013
48	att48.tsp	10628
76	eil76.tsp	538
100	kroB100.tsp	22141
124	pr124.tsp	59030
229	gr229.tsp	134602
264	pr264.tsp	49135
280	a280.tsp	2579
318	lin318.tsp	42029
400	rd400.tsp	15281
431	gr431.tsp	171414
442	pcb442.tsp	50778
493	d493.tsp	35002
532	att532.tsp	27686

Tab. 4.1: Instances Used in the Thesis

models. While the former are characterized by many ants using the same CPU and rare communication between the CPUs, in the latter only few ants use each CPU and there is a lot of communication going on. A review of the trends and strategies in designing parallel algorithms may be found in [10]. In this thesis, the strategy of *Parallel Independent Runs* was followed for program execution itself, but a fine-grained approach for the shortcutting part. Thus each execution on a node would proceed independently, however, the decision to compute the next iteration was made only after ‘checking’ with the system for any shortcutting. An advantage of this approach was that the ‘shortcuted’ code was relatively clean and easier to follow. Also, all com-

munication between nodes was restricted to either the ‘raising of’ or ‘responding to’ a shortcut. What this meant was that any communication delays or synchronization lags due to the shortcutting subsystem itself would show up rather directly.

Option	Long Form	Explanation
-a	-alpha	alpha (influence of pheromone trails)
-b	-beta	beta (influence of heuristic information)
-c	-elitistants	number of elitist ants
-d	-dlb	1: use don't look bits in local search
-e	-rho	rho: pheromone trail evaporation
-f	-rasranks	number of ranks in rank-based Ant System
-g	-nnants	nearest neighbours in tour construction
-h	-help	display this help text and exit
-i	-tsplibfile	input file (TSPLIB format necessary)
-k	-nnls	number of nearest neighbors for local search
-l	-localsearch	0: no local search 1: 2-opt 2: 2.5-opt 3: 3-opt
-m	-ants	number of ants
-o	-optimum	stop if tour better or equal optimum is found
-q	-q0	q_0 : prob. of best choice in tour construction
-r	-tries	number of independent trials
-s	-tours	number of steps in each trial
-t	-time	maximum time for each trial
-u	-as	apply basic Ant System
-v	-eas	apply elitist Ant System
-w	-ras	apply rank-based version of Ant System
-x	-mmas	apply MAX-MIN ant system
-y	-bwas	apply best-worst ant system
-z	-acs	apply ant colony system

Tab. 4.2: Program Execution Options

Finally, in terms of a biological equivalent, ACO with Shortcutting techniques would be analogous to ants (or really, colonies of ants) communicating via Stigmergy as they go about foraging for food as usual. However, when any ant finds a *really good* source of food, it shouts out its location to all other ants, which then abandon their current search and rapidly converge to this ‘good’ source.

4.4 Results

We now focus on the experimental results of the thesis, starting with the Methodology. Table 4.1 describes the TSPLIB instances that were used in the thesis. Table 4.2 shows the options available for executing the program. All runs were on the Raven cluster at Cal State San Bernardino. Raven is a cluster of 13 Compaq Proliant DL-360 G2 machines. (raven0 to raven12). Each node has two Pentium III⁶ processors running at 1.4 GHz, and 256MB of SDRAM. The basic program was ran as follows.

1. Only those TSP instances in TSPLIB that had known values of optimal tour lengths were considered.
2. 15 TSPLIB instances were selected in all; these ranged in problem size from 22 (ulysses22.tsp) to 532 (att532.tsp).
3. The size of Nearest Neighbor (input k) was varied from 1 to 5 to 10.
4. Shortcutting comparisons were made by running on 1 node, on 4 nodes and on 8 nodes.

⁶ Although the cluster is 5 years old and does not represent the latest technology, Raven is capable [13] of sustained performance of more than 6GFlops, or about 60% of the performance new Cray XD-1.

5. The number of ants (input m to the algorithm) was varied from 2 ants, 4 ants and 16 ants.
6. Work done in an execution was counted as the total number of tours constructed in that execution.
7. From the ACO variants possible in Section 2.5, MMAS (input x) was picked.
8. The other options were left at their default values.
9. For each instance, the ACO algorithm was executed three times, once with the known optimal value and then once each with a relaxation of 5% and 10%. (i.e., with 1.05 times and 1.10 times the optimal value, for input o).
10. Each such combination was repeated for 10 iterations.

No.	Ants	NN Size	Reference
1	2	1	Figure 4.1
2	2	5	Figure 4.2
3	2	10	Figure 4.3
4	4	1	Figure 4.4
5	4	5	Figure 4.5
6	4	10	Figure 4.6
7	16	1	Figure 4.7
8	16	5	Figure 4.8
9	16	10	Figure 4.9

Tab. 4.3: Reference for Reading the Graphs

We now go into a detailed look at the results. As always, pictures speak better than words, so we do our analysis on a series of graphs, rather than using the actual

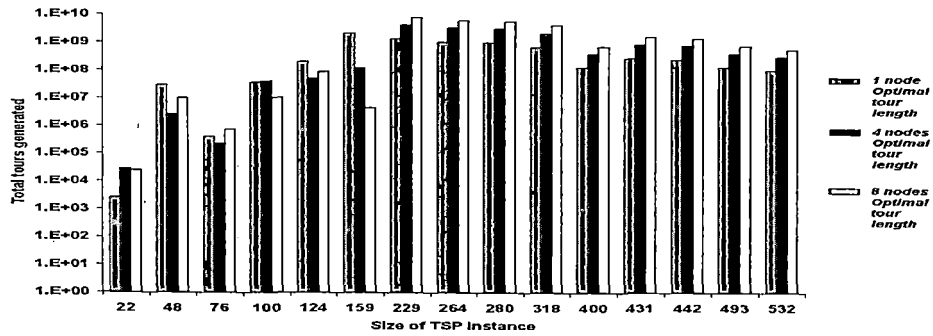
numbers themselves. The large number of input option variations resulted in several graphs. Table 4.3 presents a useful reference for reading the graphs.

Some very interesting observations can be ascertained from these figures. Focussing on Figure 4.1 we note that the in Figure 4.1(c) all instances show an improvement over the 1 node case, while in Figure 4.1(b), all except 493 do. Note that the scale for the tours generated is logarithmic, so the improvement in most cases, for instance, at size 159, are quite spectacular. Also observe that the 4 node case usually performs less work than the corresponding 8 node case, all other factors being equal. Similar observations can be made from Figures 4.2 through 4.9.

4.5 Summary

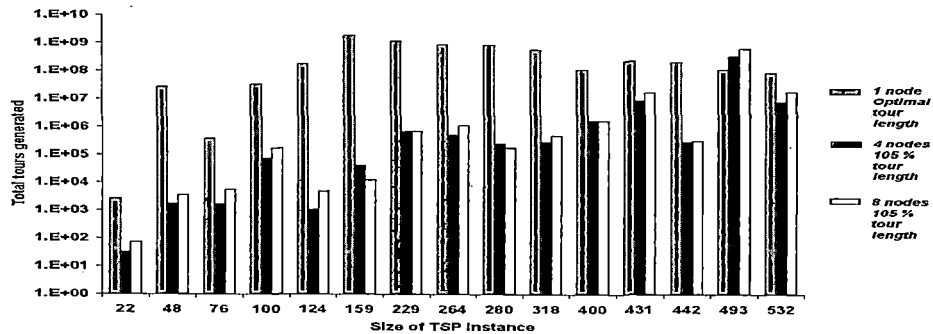
Figure 4.10 describes a summary of the results of the thesis. Figure 4.10(a), represents a direct comparison between single node and multiple node executions. The tick mark represents those iterations where an improvement in total work done was noted, in either the 4-node or the 8-node case. As can be seen, there is no direct pattern by which we can deduce that shortcutting impacts the total work done. As a matter of fact, in some runs the work done in parallel seemed to be significantly larger. Since we are comparing the total work done (as an integer count), and not the time required to finish an operation, this seems to show the influence of communication lag.

However, Figures 4.10(b) and 4.10(c) seem to show a different picture altogether. The grid is now almost completely filled with tick marks (and in almost all cases, these tick marks referred to BOTH the 4 node and the 8 node variant).

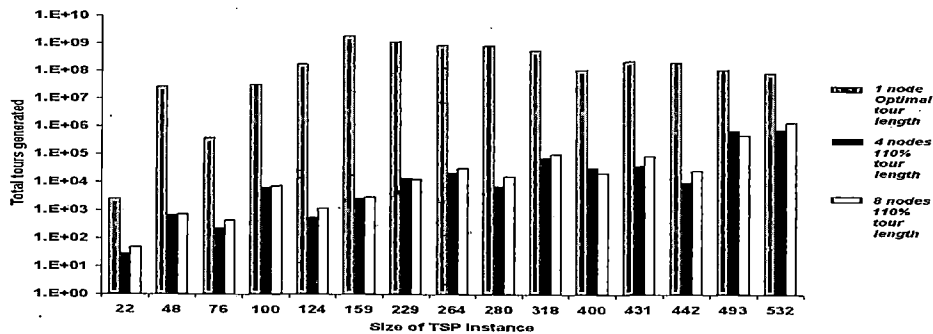


(a) Targeting optimal value

Instance sizes 48, 124 and 229 show improvement; 76 and 100 show mixed results, while all others show none.



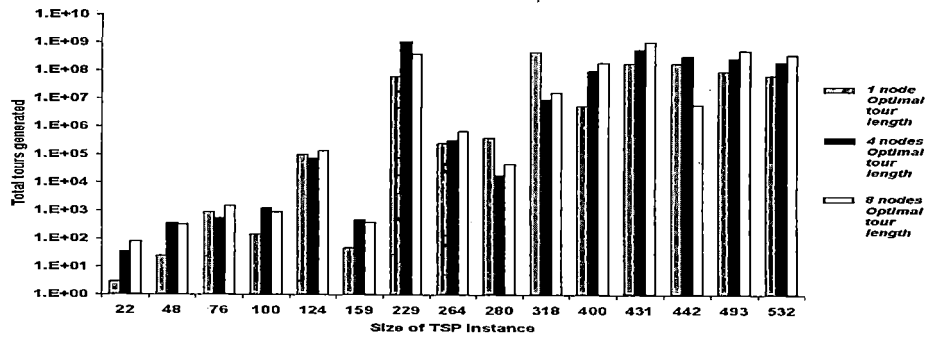
(b) Target 5% relaxed



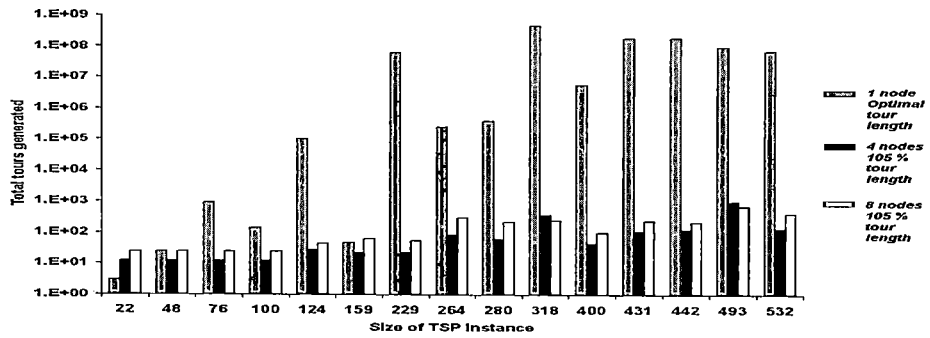
(c) Target 10% relaxed

In 4.1(c) all instances show an improvement over the 1 node case, while in 4.1(b), all except 493 do. Note that the scale for the tours generated is logarithmic, so the improvement in most cases are quite spectacular (for instance - at size 159). Also observe that the 4 node case usually performs less work than the corresponding 8 node case, all other factors being equal.

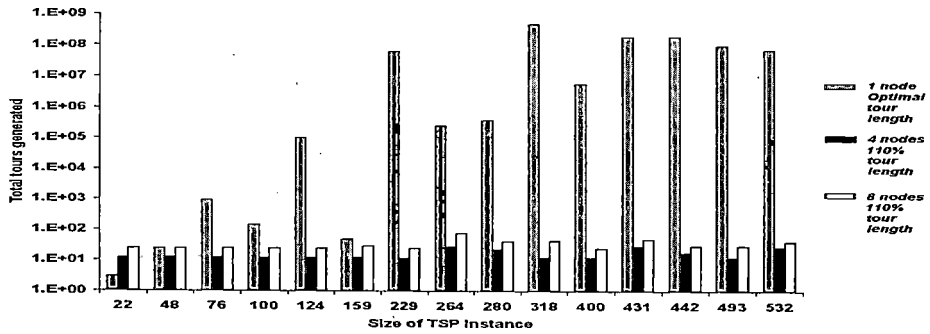
Fig. 4.1: Shortcutting Results for 2 Ants with Nearest Neighbor Size 1



(a) Targeting optimal value

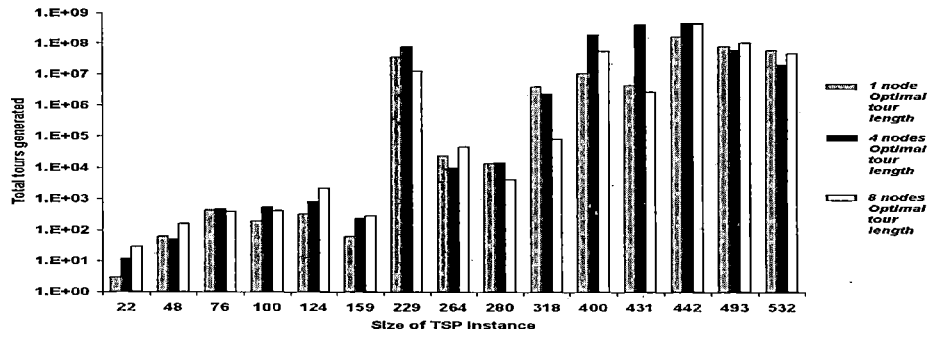


(b) Target 5% relaxed

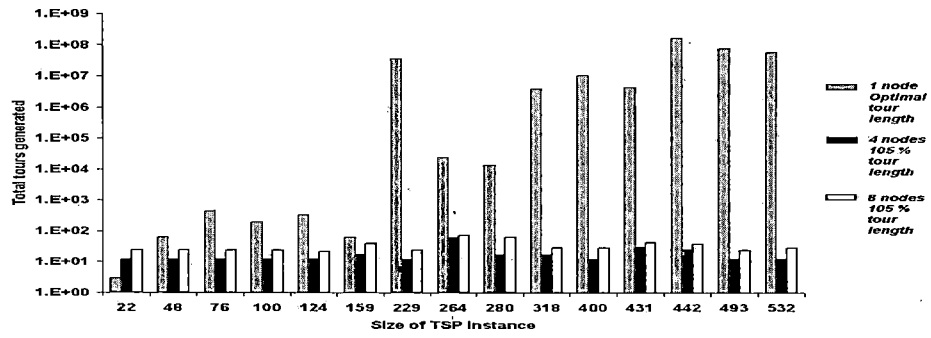


(c) Target 10% relaxed

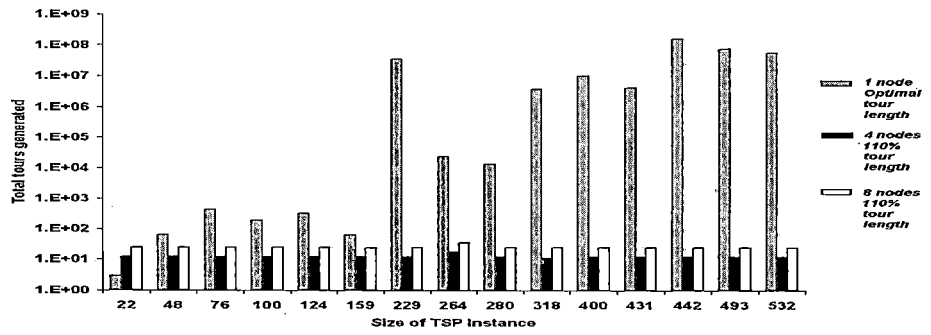
Fig. 4.2: Shortcutting Results for 2 Ants with Nearest Neighbor Size 5



(a) Targeting optimal value

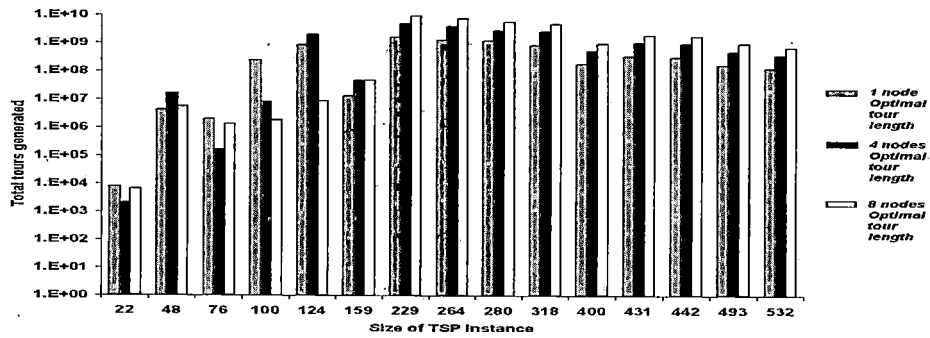


(b) Target 5% relaxed

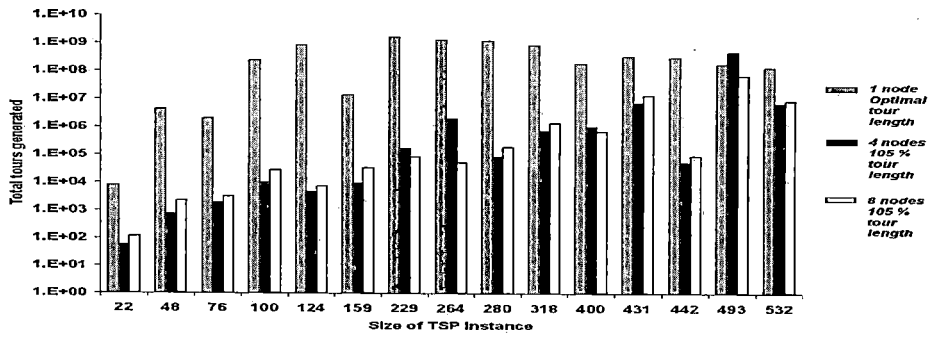


(c) Target 10% relaxed

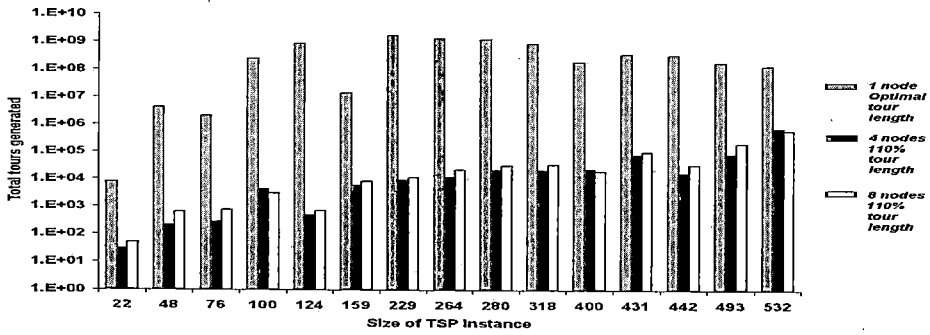
Fig. 4.3: Shortcutting Results for 2 Ants with Nearest Neighbor Size 10



(a) Targeting optimal value

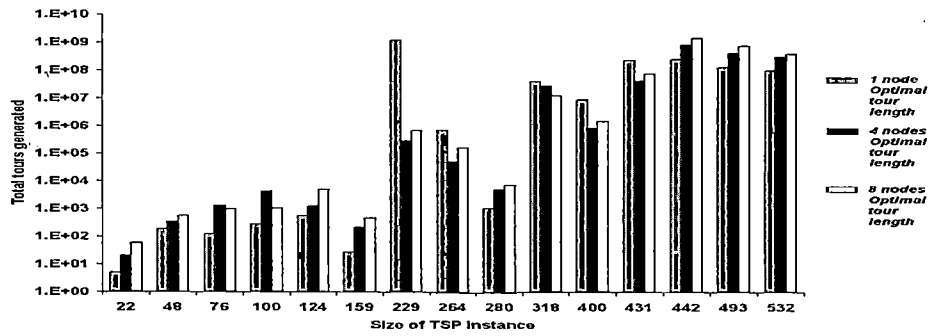


(b) Target 5% relaxed

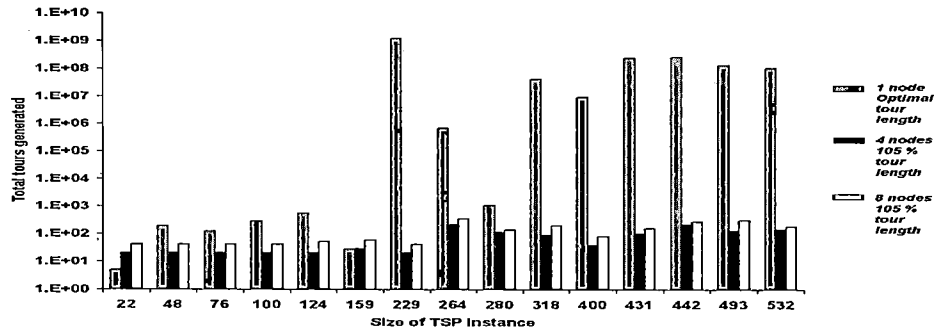


(c) Target 10% relaxed

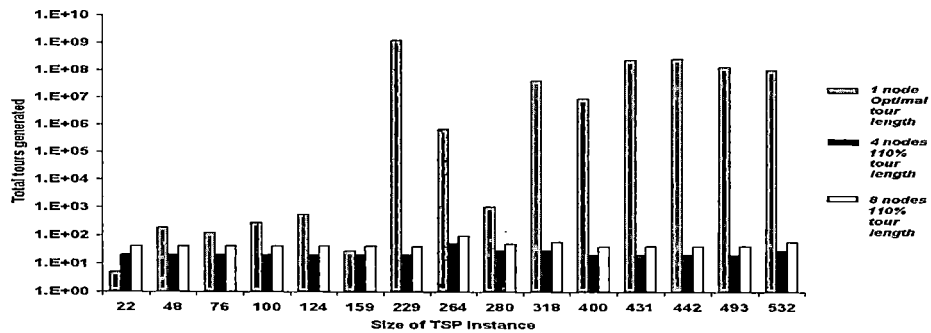
Fig. 4.4: Shortcutting Results for 4 Ants with Nearest Neighbor Size 1



(a) Targeting optimal value

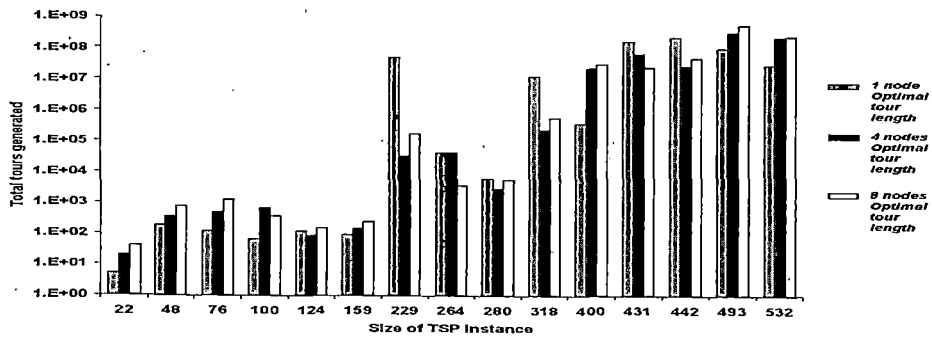


(b) Target 5% relaxed

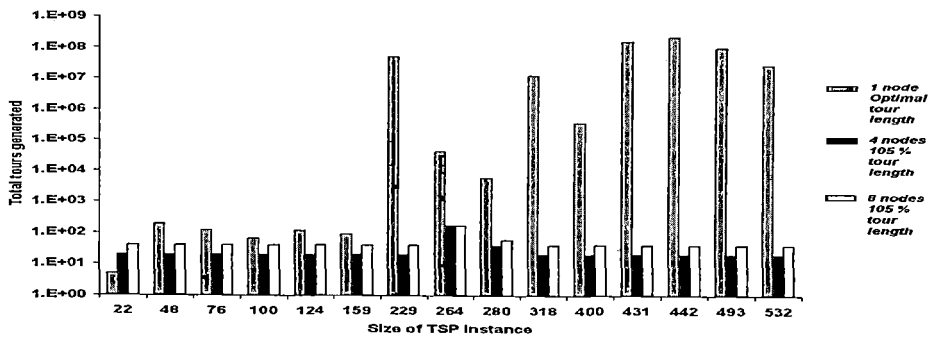


(c) Target 10% relaxed

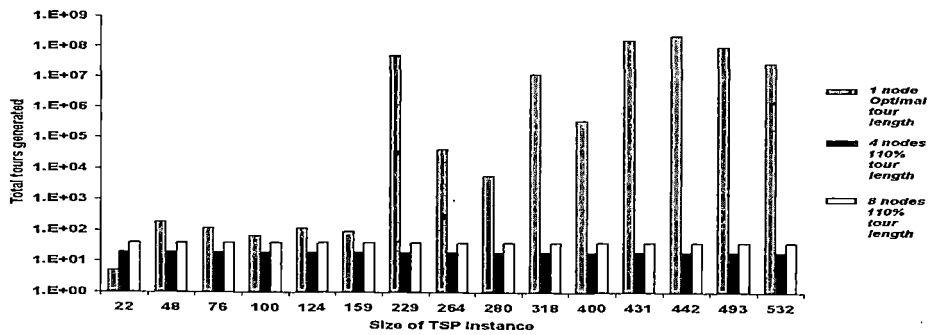
Fig. 4.5: Shortcutting Results for 4 Ants with Nearest Neighbor Size 5



(a) Targeting optimal value

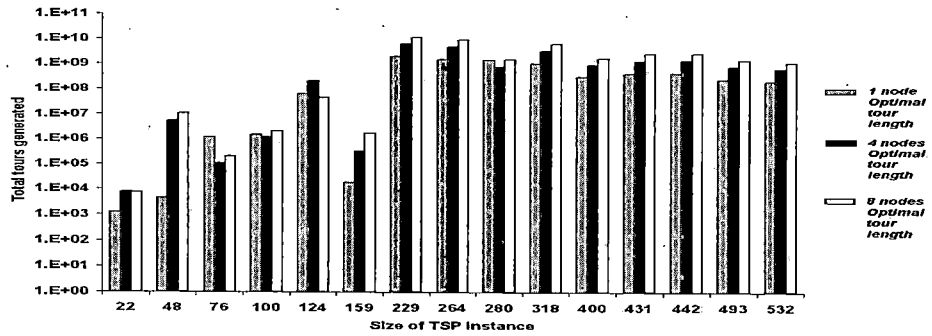


(b) Target 5% relaxed

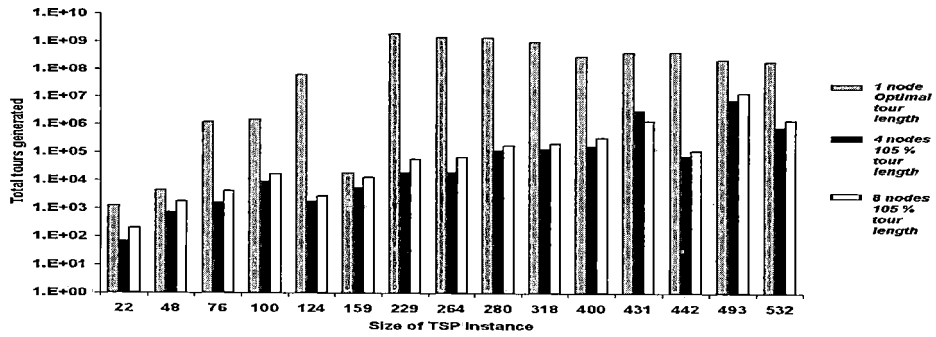


(c) Target 10% relaxed

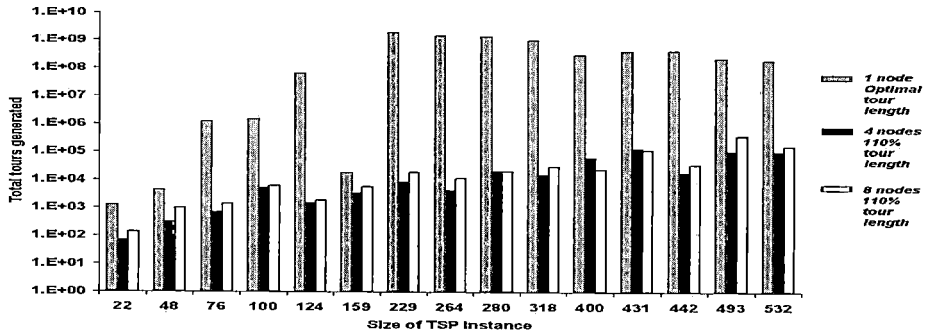
Fig. 4.6: Shortcutting Results for 4 Ants with Nearest Neighbor Size 10



(a) Targeting optimal value

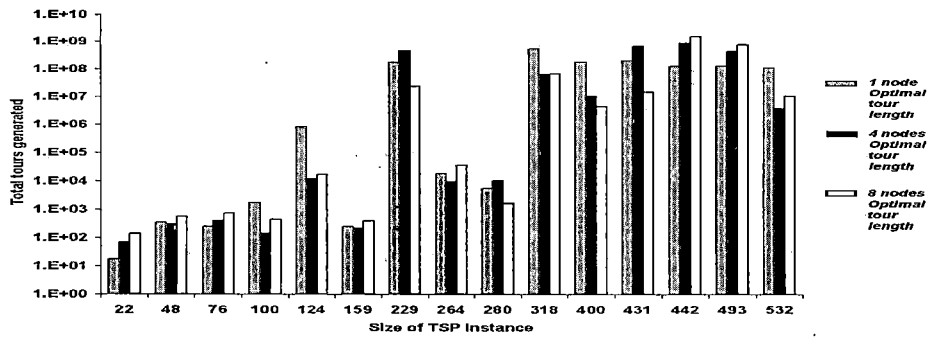


(b) Target 5% relaxed

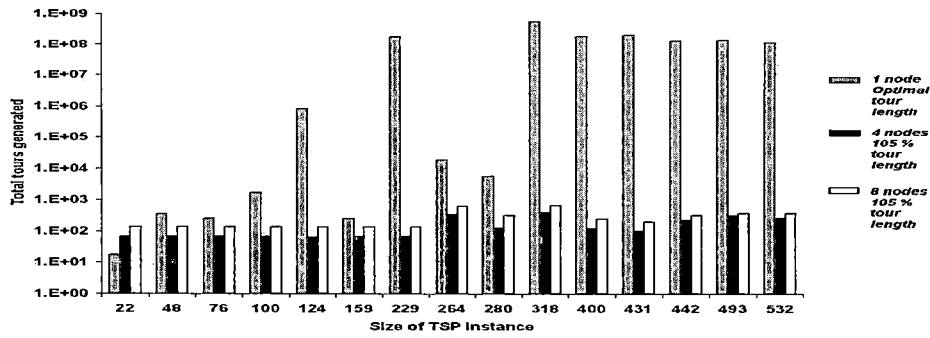


(c) Target 10% relaxed

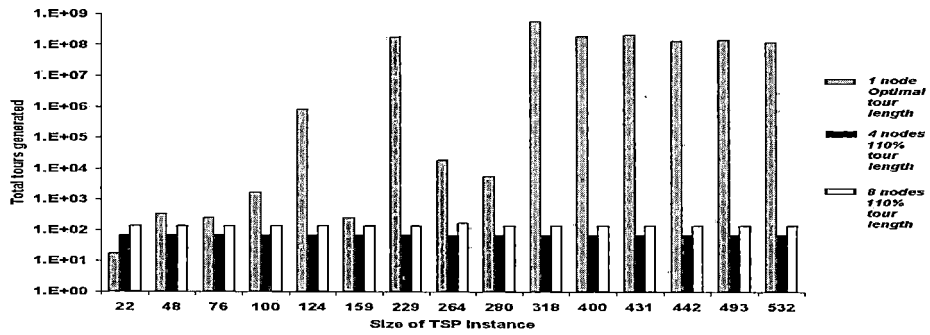
Fig. 4.7: Shortcutting Results for 16 Ants with Nearest Neighbor Size 1



(a) Targeting optimal value

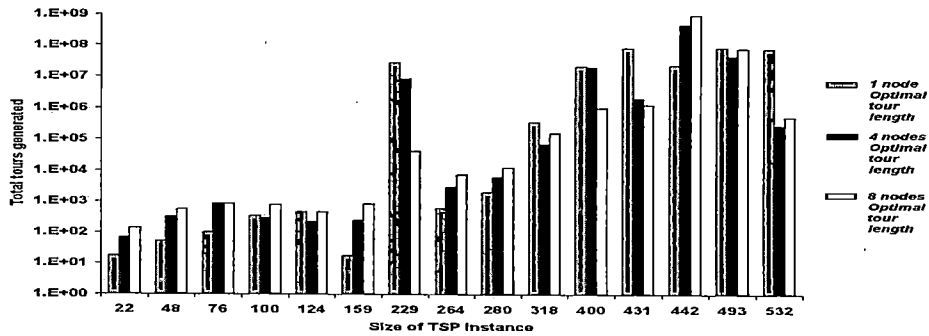


(b) Target 5% relaxed

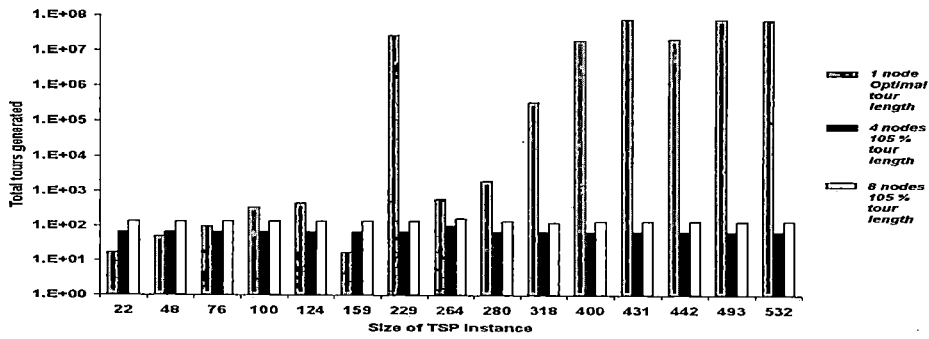


(c) Target 10% relaxed

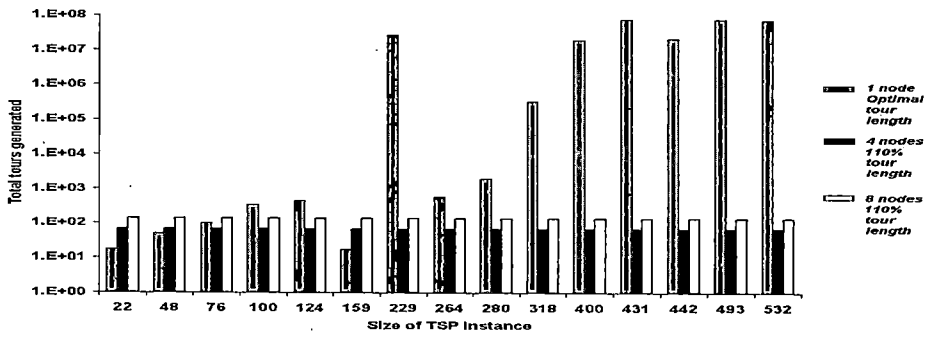
Fig. 4.8: Shortcutting Results for 16 Ants with Nearest Neighbor Size 5



(a) Targeting optimal value



(b) Target 5% relaxed



(c) Target 10% relaxed

Fig. 4.9: Shortcutting Results for 16 Ants with Nearest Neighbor Size 10

Ants	2			4			16			Total
NN Size	1	5	10	1	5	10	1	5	10	
ulysses22				✓						1
att48	✓		✓					✓		3
eil76	✓	✓	✓	✓				✓		5
kroB100	✓			✓			✓	✓	✓	5
pr124	✓	✓		✓		✓	✓	✓	✓	7
u159	✓							✓		2
gr229			✓		✓	✓		✓	✓	5
pr264			✓		✓	✓		✓		4
a280		✓	✓		✓	✓	✓	✓		5
lin318		✓	✓		✓	✓		✓	✓	6
rd400					✓			✓	✓	3
gr431			✓		✓	✓		✓	✓	5
pcb442		✓				✓				2
d493			✓						✓	2
att532			✓					✓	✓	3
Total	5	5	9	4	5	7	4	11	8	58

(a) Targeting optimal value

Ants	2			4			16			Total
NN Size	1	5	10	1	5	10	1	5	10	
ulysses22	✓			✓			✓			3
att48	✓	✓	✓	✓	✓	✓	✓	✓	✓	8
eil76	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
kroB100	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
pr124	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
u159	✓	✓	✓	✓	✓	✓	✓	✓	✓	8
gr229	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
pr264	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
a280	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
lin318	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
rd400	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
gr431	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
pcb442	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
d493	✓	✓	✓	✓	✓	✓	✓	✓	✓	8
att532	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
Total	14	14	14	15	14	14	15	14	12	127

(b) Target 5% relaxed

Ants	2			4			16			Total
NN Size	1	5	10	1	5	10	1	5	10	
ulysses22	✓			✓			✓			3
att48	✓	✓	✓	✓	✓	✓	✓	✓	✓	8
eil76	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
kroB100	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
pr124	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
u159	✓	✓	✓	✓	✓	✓	✓	✓	✓	8
gr229	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
pr264	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
a280	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
lin318	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
rd400	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
gr431	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
pcb442	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
d493	✓	✓	✓	✓	✓	✓	✓	✓	✓	8
att532	✓	✓	✓	✓	✓	✓	✓	✓	✓	9
Total	15	14	14	15	14	14	15	14	12	127

(c) Target 10% relaxed

A tick mark represents those cases where an improvement in total work done, due to shortcutting, was noted, in either the 4-node or the 8-node case. Observe that:

1. No pattern seems discernible in 4.10(a). pr124 seems to have the best improvement across all combinations (row total = 7), while the 16 ant, nn = 5 (column total = 11) case seems to have consistent improvement across all instance sizes. This indicates the true nondeterministic nature of the problem.
2. Once the target is 'relaxed', 4.10(b) and 4.10(c), there are significant improvements, for almost all instances and combinations. This indicates shortcutting may be of immense value for problems when good-enough results are needed.
3. There was no consistent improvement for ulysses22, even with the target tour length being relaxed by 5 or 10%. This indicates that synchronization delays may exceed the time taken to complete computation.

Fig. 4.10: Summary of Shortcutting Results

Again, the improvement in many cases was several orders of magnitudes. This now seems to suggest that for a *good-enough* solution, say, one that relaxes the tour-length constraint by 5 or 10 percent, ACO with shortcutting might be more than a viable alternative; it may actually be an excellent one.

5. CONCLUSIONS

This thesis demonstrates shortcutting techniques in a metaheuristic, ACO, using a parallel compiler, PC. By virtue of the fact that results are available at different times in parallel executions, it is possible to obtain parallel speedups greater than 100%.

The implementation was based on an existing and well-proven sequential code base. The fact that one could easily parallelize the code with just a few lines of PC calls goes to show the elegance behind the design of PC and also demonstrates its ease of use. Shortcutting was also obtained with the judicious placement of a few lines of SOS code.

PC has been used earlier in problems that were pretty much geared towards it. By applying PC to not just another algorithm, but a metaheuristic, this work has paved the way for the application of PC to industrial strength problems. These could very well be other algorithms, or even other NP - hard applications of ACO or other swarm algorithms.

Another direction for future work would be to upgrade PC to the latest versions of the C compiler. It is also expected that SOS functionality will be embedded into the PC language. It would be desirable to port PLanguages to C++. Finally, given that implementations of MPI are available on Windows, PC may be brought

to mainstream by making it available on the Windows platform. On the hardware end of the spectrum, an interesting research would be to see the behavior of PC on multi-core CPU machines.

APPENDIX A
THE ANT COLONY METAHEURISTIC

A.1 Introduction

The chapter is divided into three parts. Section A.2 describes a formal model of the ACO, starting with a notation of a combinatorial problem in A.2.1, and its application to the ACO and the TSP in A.2.2 and A.2.3 respectively. Next, Section A.3 expands on the Algorithm presented in Table 2.1. Finally, Section A.4 describes the pheromone update rule for the MMAS (MAX-MIN Ant System) variant of the ACO used in this thesis.

A.2 Modeling the Metaheuristic

A.2.1 Combinatorial Optimization

We start with a model of a **combinatorial optimization problem**. A *model* $P = (S, O, f)$ of a combinatorial optimization problem consists of:

- a search space S defined over a finite set of discrete decision variables $X_i, i = 1, \dots, n$
- a set O of constraints among the variables. An empty set \emptyset denotes an unconstrained problem; and
- an objective function $f : S \mapsto \mathbb{R}_0^+$ to be minimized.¹

The generic variable X_i takes values in $D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$. A feasible solution $s \in S$ is one complete assignment of values to variables that satisfies all constraints in O . A solution $s^{opt} \in S$ is called a global optimum if and only if $f(s^{opt}) \leq f(s) \forall s \in S$. The

¹ or equivalently maximized

set of all globally optimal solutions is denoted by $S^{opt*} \in S$. Solving a COP requires finding at least one $s^{opt} \in S^{opt*}$

This model is used to define the pheromone model of ACO. A pheromone value is associated with each possible *solution component*; that is, with each possible assignment of a value to a variable. Formally, the pheromone value τ_{ij} is associated with the solution component c_{ij} , which consists of the assignment $X_i = v_i^j$. The set of all possible solution components is denoted by C .

A.2.2 The Ant Colony Optimization Graph

In ACO, an artificial ant builds a solution by traversing the fully connected *construction graph* $G_C(V, E)$, of V vertices and E edges. This graph can be obtained from C by representing solution components as either vertices or edges. Artificial ants move from vertex to vertex along the edges, incrementally building a *partial solution*, and additionally deposit a certain amount of pheromone on the components they traverse. The amount δt of pheromone deposited may depend on the quality of the solution found and is used by subsequent ants as a guide toward promising regions of the search space.

A.2.3 The Traveling Salesman Problem Graph

In an n -city TSP, a solution can be represented through a set of n variables, each associated with a city. The variable X_i indicates the city to be visited after city i . A solution component is a pair of cities to be visited in order one after the other, i.e., the solution component $c_{ij} = (i, j)$ indicates that for the given solution, city j

should be visited immediately after city i . The construction graph is now a graph in which the vertices are the cities of the original TSP, and the edges are the solution components. Ants deposit pheromone on the edges of the graph.

A.3 Phases of the Metaheuristic

This section is a more detailed description of the three phases in ACO metaheuristic presented in Table 2.1. After initialization, the metaheuristic iterates over three phases: at each iteration, a number of solutions are constructed by the ants; these solutions are then improved through a local search (this step is optional), and finally the pheromone is updated.

A.3.1 Construct Solutions

A set of m artificial ants constructs solutions from elements of a finite set of available solution components $C = \{c_{ij}\}, i = 1, \dots, n, j = 1, \dots, |D_i|$. A solution construction starts from an empty partial solution $s^p = \emptyset$. At each construction step, s^p is extended by adding a feasible solution component from the set $N(s^p) \subseteq C$, which is defined as the set of components that can be added to the current partial solution s^p without violating any of the constraints in O . The process of constructing solutions can be regarded as a walk on the construction graph $G_C = (V, E)$. The rule for the choice of solution components from $N(s^p)$ vary across different ACO algorithms but are all inspired by the model of the behavior of real ants.

A.3.2 Apply Local Search

Although an optional step, it is common to improve the solutions obtained by the ants through a local search once the solutions have been constructed, and before updating the pheromone. This phase, which is highly problem-specific, and is usually included in state-of-the-art ACO algorithms. The 3-opt local search was used in this thesis. For more details refer to [9].

A.3.3 Update Pheromones

This phase increases the pheromone values associated with good or promising solutions, and decreases those that are associated with bad ones. Usually, this is achieved (i) by decreasing all the pheromone values through pheromone evaporation, and (ii) by increasing the pheromone levels associated with a chosen set of good solutions.

A.3.4 Daemon Actions

This is an optional phase. Daemon actions can be used to implement centralized actions which cannot be performed by single ants. An examples would be collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective.

A.4 Max-Min Ant System Pheromone Update Rule

The MMAS algorithm [23] is an improvement over the original Ant System (AS). In AS at each iteration, the pheromone values are updated by all the m ants

that have built a solution in the iteration itself. However, in *MMAS* only the best ant updates the pheromone trails. Also, the value of the pheromone is bound. The pheromone update is implemented as follows:

$$\tau_{ij} \leftarrow \left[(1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij}^{best} \right]_{\tau_{min}}^{\tau_{max}}$$

where the operator $[x]_b^a$ is defined as follows.

$$[x]_b^a = \begin{cases} a & \text{if } x > a \\ b & \text{if } x < b \\ x & \text{otherwise;} \end{cases}$$

and $\Delta\tau_{ij}^{best}$ is:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & \text{if } (i, j) \text{ belongs to the tour} \\ 0 & \text{otherwise;} \end{cases}$$

where L_{best} is the length of the tour of the best ant. This may be either the best tour found in the current iteration, or found since the start of the algorithm, or a combination of both. [23] provides guidelines for defining the lower and upper bounds on the pheromone values, τ_{min} and τ_{max} . These can also be obtained empirically and tuned to the specific problem considered.

APPENDIX B
PORTING FROM C TO PC

The elegance of the design behind PC is reflected in the minimal changes required to port a non-trivial chunk of C code to PC. This chapter lists those that were made to the ACO software available at the Ant Colony Website.² Most of the changes listed below were very minor in scope and could easily be addressed in subsequent versions of PC

TRACE macros PC seemed to have trouble expanding the TRACE macro. A

simple workaround was to comment it, and use *printf* statements to debug wherever necessary. For instance,

```
TRACE ( printf("apply local search\n"); );
```

becomes

```
/* TRACE ( printf("apply local search\n"); ); */
```

or

```
printf("apply local search\n");
```

Single PC module The ACOTSP code was spread across several *c* files.

However, PC could have only *pc* file, in which the variables *myProc*, *nProc*, etc., were defined. If any section of code not in the single *pc* file needed access to those variables, they had to be passed in explicitly. For instance,

```
void init_program(int argc, char *argv[] )
```

was changed to

```
void init_program(int myProc, int argc, char *argv[])
```

Function signatures inside Comments The PC compiler had trouble when chunks of commented code had C-style function signatures in them. This could also be

² <http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html>

because of PC being built against an older version of gcc. The workaround was to remove the signature in question.

New source file Once a parallelization strategy was selected, it was necessary to convert that specific c source file to PC. This was done by creating a new copy of the file and renaming it as ACOTSP.PC

Makefile The ACO software came with its own makefile. This had to be modified to also invoke the pcc compiler. This was a trivial task using the sample makefile provided with the PC distribution.

APPENDIX C
SOURCE CODE

ants.h

/*

```
AAAA   CCCC   OOOO   TTTTT   SSSSS   PPPPP
AA  AA  CC    OO  OO   TT    SS    PP  PP
AAAAAA  CC    OO  OO   TT    SSSS   PPPPP
AA  AA  CC    OO  OO   TT    SS    PP
AA  AA  CCCC   OOOO   TT    SSSSS   PP
```


ACO algorithms for the TSP #####
#####

Version: 2.0
File: ants.h
Author: Sammy D'Souza
Purpose: modifications for PC
Check: README and gpl.txt
Copyright (C) 2007 Sammy D'Souza

Version: 1.0
File: ants.h
Author: Thomas Stuetzle
Purpose: implementation of procedures for ants' behaviour
Check: README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle

*/

/******

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik
Hochschulstr. 10

D-64283 Darmstadt
Germany

```
*****/

#define HEURISTIC(m,n)      (1.0 / ((double) instance.distance[m][n] + 0.1))
/* add a small constant to avoid division by zero if a distance is
zero */

#define EPSILON              0.00000000000000000000000000000001

#define MAX_ANTS             1024      /* max no. of ants */
#define MAX_NEIGHBOURS      512      /* max. no. of nearest neighbours in candidate
set */

/* Note that *tour needs to be allocated for length n+1 since the first city of
a tour (at position 0) is repeated at position n. This is done to make the
computation of the tour length easier
*/
typedef struct {
    long int *tour;
    char *visited;
    long int tour_length;
} ant_struct;

extern ant_struct *ant;          /* this (array of) struct will hold the colony */
extern ant_struct *best_so_far_ant; /* struct that contains the best-so-far
ant */
extern ant_struct *restart_best_ant; /* struct that contains the restart-best
ant */

extern double **pheromone; /* pheromone matrix, one entry for each arc */
extern double **total; /* combination of pheromone times heuristic
information */

extern double *prob_of_selection;

extern long int n_ants; /* number of ants */
extern long int nn_ants; /* length of nearest neighbor lists for the ants'
solution construction */

extern double rho; /* parameter for evaporation */
extern double alpha; /* importance of trail */
extern double beta; /* importance of heuristic evaluate */
extern double q_0; /* probability of best choice in tour construction
*/

extern long int as_flag; /* = 1, run ant system */
extern long int eas_flag; /* = 1, run elitist ant system */
extern long int ras_flag; /* = 1, run rank-based version of ant system */
extern long int mmas_flag; /* = 1, run MAX-MIN ant system */
extern long int bwas_flag; /* = 1, run best-worst ant system */
extern long int acs_flag; /* = 1, run ant colony system */

extern long int elitist_ants; /* additional parameter for elitist ant system,
```

```

        it defines the number of elitist ants */

extern long int ras_ranks;      /* additional parameter for rank-based version
of ant
        system */

extern double trail_max;      /* maximum pheromone trail in MMAS */
extern double trail_min;      /* minimum pheromone trail in MMAS */
extern long int u_gb;          /* every u_gb iterations update with
best-so-far ant;
        parameter used by MMAS for scheduling best-so-far update
        */

extern double trail_0;        /* initial pheromone trail level in ACS and
BWAS */

/* Pheromone manipulation etc. */

void init_pheromone_trails ( double initial_trail );

void evaporation ( void );

void evaporation_nn_list ( void );

void global_update_pheromone ( ant_struct *a );

void global_update_pheromone_weighted ( ant_struct *a, long int weight );

void compute_total_information( void );

void compute_nn_list_total_information( void );

/* Ants' solution construction */

void ant_empty_memory( ant_struct *a );

void place_ant( ant_struct *a , long int phase );

void choose_best_next( ant_struct *a, long int phase );

void neighbour_choose_best_next( ant_struct *a, long int phase );

void choose_closest_next( ant_struct *a, long int phase );

void neighbour_choose_and_move_to_next( ant_struct *a , long int phase );

/* Auxiliary procedures related to ants */

long int find_best ( void );

long int find_worst( void );

void copy_from_to(ant_struct *a1, ant_struct *a2);

void allocate_ants ( void );

```

```
long int nn_tour( void );

long int distance_between_ants( ant_struct *a1, ant_struct *a2);

/* Procedures specific to MAX-MIN Ant System */

void mmas_evaporation_nn_list( void );

void check_nn_list_pheromone_trail_limits( void );

void check_pheromone_trail_limits( void );

/* Procedures specific to Ant Colony System */

void global_acs_pheromone_update( ant_struct *a );

void local_acs_pheromone_update( ant_struct *a, long int phase );

/* Procedures specific to Best Worst Ant System */

void bwass_worst_ant_update( ant_struct *a1, ant_struct *a2);

void bwass_pheromone_mutation( void );
```

ants.h

/*

```

AAAA   CCCC   OOOO   TTTTT   SSSSS   PPPPP
AA  AA  CC     OO  OO   TT     SS     PP  PP
AAAAAA  CC     OO  OO   TT     SSSS   PPPPP
AA  AA  CC     OO  OO   TT     SS     PP
AA  AA  CCCC   OOOO   TT     SSSSS   PP

```

```

#####
#####      ACO algorithms for the TSP      #####
#####

```

```

Version: 2.0
File:    ants.h
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza

```

```

Version: 1.0
File:    ants.h
Author:  Thomas Stuetzle
Purpose: implementation of procedures for ants' behaviour
Check:   README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle

```

*/

/******

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt
Germany

```
*****/

#define HEURISTIC(m,n)      (1.0 / ((double) instance.distance[m][n] + 0.1))
/* add a small constant to avoid division by zero if a distance is
zero */

#define EPSILON              0.00000000000000000000000000000001

#define MAX_ANTS             1024      /* max no. of ants */
#define MAX_NEIGHBOURS 512      /* max. no. of nearest neighbours in candidate
set */

/* Note that *tour needs to be allocated for length n+1 since the first city of
a tour (at position 0) is repeated at position n. This is done to make the
computation of the tour length easier
*/
typedef struct {
    long int  *tour;
    char      *visited;
    long int  tour_length;
} ant_struct;

extern ant_struct *ant;      /* this (array of) struct will hold the colony */
extern ant_struct *best_so_far_ant; /* struct that contains the best-so-far
ant */
extern ant_struct *restart_best_ant; /* struct that contains the restart-best
ant */

extern double  **pheromone; /* pheromone matrix, one entry for each arc */
extern double  **total;    /* combination of pheromone times heuristic
information */

extern double  *probab_of_selection;

extern long int n_ants;     /* number of ants */
extern long int nn_ants;   /* length of nearest neighbor lists for the ants'
solution construction */

extern double  rho;        /* parameter for evaporation */
extern double  alpha;     /* importance of trail */
extern double  beta;      /* importance of heuristic evaluate */
extern double  q_0;       /* probability of best choice in tour construction
*/

extern long int as_flag;   /* = 1, run ant system */
extern long int eas_flag; /* = 1, run elitist ant system */
extern long int ras_flag; /* = 1, run rank-based version of ant system */
extern long int mmas_flag; /* = 1, run MAX-MIN ant system */
extern long int bwas_flag; /* = 1, run best-worst ant system */
extern long int acs_flag; /* = 1, run ant colony system */
```

```

extern long int elitist_ants;    /* additional parameter for elitist ant system,
                                it defines the number of elitist ants */

extern long int ras_ranks;      /* additional parameter for rank-based version
of ant
                                system */

extern double trail_max;        /* maximum pheromone trail in MMAS */
extern double trail_min;        /* minimum pheromone trail in MMAS */
extern long int u_gb;           /* every u_gb iterations update with
best-so-far ant;
                                parameter used by MMAS for scheduling best-so-far update
                                */

extern double trail_0;          /* initial pheromone trail level in ACS and
BWAS */

/* Pheromone manipulation etc. */

void init_pheromone_trails ( double initial_trail );

void evaporation ( void );

void evaporation_nn_list ( void );

void global_update_pheromone ( ant_struct *a );

void global_update_pheromone_weighted ( ant_struct *a, long int weight );

void compute_total_information( void );

void compute_nn_list_total_information( void );

/* Ants' solution construction */

void ant_empty_memory( ant_struct *a );

void place_ant( ant_struct *a , long int phase );

void choose_best_next( ant_struct *a, long int phase );

void neighbour_choose_best_next( ant_struct *a, long int phase );

void choose_closest_next( ant_struct *a, long int phase );

void neighbour_choose_and_move_to_next( ant_struct *a , long int phase );

/* Auxiliary procedures related to ants */

long int find_best ( void );

long int find_worst( void );

void copy_from_to(ant_struct *a1, ant_struct *a2);

```



```
void allocate_ants ( void );

long int nn_tour( void );

long int distance_between_ants( ant_struct *a1, ant_struct *a2);

/* Procedures specific to MAX-MIN Ant System */

void mmas_evaporation_nn_list( void );

void check_nn_list_pheromone_trail_limits( void );

void check_pheromone_trail_limits( void );

/* Procedures specific to Ant Colony System */

void global_acs_pheromone_update( ant_struct *a );

void local_acs_pheromone_update( ant_struct *a, long int phase );

/* Procedures specific to Best Worst Ant System */

void bwast_worst_ant_update( ant_struct *a1, ant_struct *a2);

void bwast_pheromone_mutation( void );
```

acotsp.pc

/*

```
AAAA   CCCC   OOOO   TTTTT   SSSSS   PPPPP
AA AA  CC    OO  OO   TT    SS    PP  PP
AAAAAA  CC    OO  OO   TT    SSSS   PPPPP
AA AA  CC    OO  OO   TT    SS    PP
AA AA  CCCC   OOOO   TT    SSSSS   PP
```

```
#####
#####      ACO algorithms for the TSP      #####
#####
```

```
Version: 2.0
File:    actosp.pc
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0
File:    main.c
Author:  Thomas Stuetzle
Purpose: main routines and control for the ACO algorithms
Check:   README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle
```

*/

/******

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt
Germany

```
*****/  
  
#include <stdio.h>  
#include <math.h>  
#include <limits.h>  
#include <assert.h>  
#include <string.h>  
#include <stdlib.h>  
#include <time.h>  
  
/*#define SIMPLE_SPRNG */  
  
#include "ants.h"  
#include "utilities.h"  
#include "InOut.h"  
#include "TSP.h"  
#include "timer.h"  
#include "ls.h"  
  
long int termination_condition( void )  
/*  
    FUNCTION:      checks whether termination condition is met  
    INPUT:         none  
    OUTPUT:        0 if condition is not met, number neg 0 otherwise  
    SIDE_EFFECTS:  none  
*/  
{  
    return ( ((n_tours >= max_tours) && (elapsed_time( VIRTUAL ) >= max_time)) ||  
            ((*best_so_far_ant).tour_length <= optimal));  
}  
  
void construct_solutions( void )  
/*  
    FUNCTION:      manage the solution construction phase  
    INPUT:         none  
    OUTPUT:        none  
    SIDE_EFFECTS:  when finished, all ants of the colony have constructed a  
solution  
*/  
{  
    long int k;      /* counter variable */  
    long int step;   /* counter of the number of construction steps */  
  
    /* TRACE ( printf("construct solutions for all ants\n"); ); */  
  
    /* Mark all cities as unvisited */  
    for ( k = 0 ; k < n_ants ; k++) {  
ant_empty_memory( &ant[k] );  
    }  
  
    step = 0;
```

```

    /* Place the ants on same initial city */
    for ( k = 0 ; k < n_ants ; k++ )
place_ant( &ant[k], step);

    while ( step < n-1 ) {
step++;
for ( k = 0 ; k < n_ants ; k++ ) {
    neighbour_choose_and_move_to_next( &ant[k], step);
    if ( acs_flag )
local_acs_pheromone_update( &ant[k], step );
}
}

    step = n;
    for ( k = 0 ; k < n_ants ; k++ ) {
ant[k].tour[n] = ant[k].tour[0];
ant[k].tour_length = compute_tour_length( ant[k].tour );
if ( acs_flag )
    local_acs_pheromone_update( &ant[k], step );
}
    n_tours += n_ants;
}

void init_try( long int ntry )
/*
    FUNCTION: initialize variables appropriately when starting a trial
    INPUT:    trial number
    OUTPUT:   none
    COMMENTS: none
*/
{

    /* TRACE ( printf("INITIALIZE TRIAL\n"); ); */

    start_timers();
    time_used = elapsed_time( VIRTUAL );
    time_passed = time_used;

    fprintf(comp_report, "seed %ld\n", seed);
    fflush(comp_report);
    /* Initialize variables concerning statistics etc. */

    n_tours      = 0;
    iteration    = 0;
    restart_iteration = 1;
    lambda       = 0.05;
    (*best_so_far_ant).tour_length = INFTY;
    found_best   = 0;

    /* Initialize the Pheromone trails, only if ACS is used, pheromones
       have to be initialized differently */
    if ( !(acs_flag || mmas_flag || bwas_flag) ) {
trail_0 = 1. / ( (rho) * nn_tour() );
/* in the original papers on Ant System, Elitist Ant System, and
   Rank-based Ant System it is not exactly defined what the

```

```

        initial value of the pheromones is. Here we set it to some
        small constant, analogously as done in MAX-MIN Ant System.
*/
init_pheromone_trails( trail_0 );
}
if ( bwas_flag ) {
trail_0 = 1. / ( (double) n * (double) nn_tour() );
init_pheromone_trails( trail_0 );
}
if ( mmas_flag ) {
trail_max = 1. / ( (rho) * nn_tour() );
trail_min = trail_max / ( 2. * n );
init_pheromone_trails( trail_max );
}
if ( acs_flag ) {
trail_0 = 1. / ( (double) n * (double) nn_tour( ) );
init_pheromone_trails( trail_0 );
}

/* Calculate combined information pheromone times heuristic information */
compute_total_information();

fprintf(comp_report,"begin try %li \n",ntry);
fprintf(stat_report,"begin try %li \n",ntry);
}

void local_search( void )
/*
    FUNCTION:      manage the local search phase; apply local search to ALL
ants; in
                    dependence of ls_flag one of 2-opt, 2.5-opt, and 3-opt
local search
                    is chosen.
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  all ants of the colony have locally optimal tours
    COMMENTS:      typically, best performance is obtained by applying local
search
                    to all ants. It is known that some improvements (e.g.
convergence
                    speed towards high quality solutions) may be obtained for some
ACO algorithms by applying local search to only some of the ants.
Overall best performance is typically obtained by using 3-opt.
*/
{
    long int k;

    /* TRACE ( printf("apply local search to all ants\n"); ); */

    for ( k = 0 ; k < n_ants ; k++ ) {
if ( ls_flag == 1 )
    two_opt_first( ant[k].tour ); /* 2-opt local search */
else if ( ls_flag == 2 )
    two_h_opt_first( ant[k].tour ); /* 2.5-opt local search */
else if ( ls_flag == 3 )

```

```

    three_opt_first( ant[k].tour ); /* 3-opt local search */
  else {
    fprintf(stderr,"type of local search procedure not correctly
specified\n");
    exit(1);
  }
  ant[k].tour_length = compute_tour_length( ant[k].tour );
}

void update_statistics( void )
/*
    FUNCTION:      manage some statistical information about the trial,
especially
                  if a new best solution (best-so-far or restart-best) is
found and
                  adjust some parameters if a new best solution is found
INPUT:           none
OUTPUT:         none
SIDE_EFFECTS:   restart-best and best-so-far ant may be updated; trail_min
and trail_max used by MMAS may be updated
*/
{

  long int iteration_best_ant;
  double p_x; /* only used by MMAS */

  iteration_best_ant = find_best(); /* iteration_best_ant is a global variable
*/

  if ( ant[iteration_best_ant].tour_length < (*best_so_far_ant).tour_length )
  {

    time_used = elapsed_time( VIRTUAL ); /* best sol found after time_used */
    copy_from_to( &ant[iteration_best_ant], best_so_far_ant );
    copy_from_to( &ant[iteration_best_ant], restart_best_ant );

    found_best = iteration;
    restart_found_best = iteration;
    found_branching = node_branching(lambda);
    branching_factor = found_branching;
    if ( mmas_flag ) {
      if ( !ls_flag ) {
        p_x = exp(log(0.05)/n);
        trail_min = 1. * (1. - p_x) / (p_x * (double)((nn_ants + 1) / 2));
        trail_max = 1. / ( (rho) * (*best_so_far_ant).tour_length );
        trail_0 = trail_max;
        trail_min = trail_max * trail_min;
      } else {
        trail_max = 1. / ( (rho) * (*best_so_far_ant).tour_length );
        trail_min = trail_max / ( 2. * n );
        trail_0 = trail_max;
      }
    }
  }
}

```

```

write_report();
    if ( ant[iteration_best_ant].tour_length < (*restart_best_ant).tour_length )
{
    copy_from_to( &ant[iteration_best_ant], restart_best_ant );
    restart_found_best = iteration;
    printf("restart best: %ld, restart_found_best %ld, time
%.2f\n", (*restart_best_ant).tour_length, restart_found_best, elapsed_time (
VIRTUAL ));
}
}

void search_control_and_statistics( void )
/*
    FUNCTION:      occasionally compute some statistics and check whether
algorithm
                  is converged
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  restart-best and best-so-far ant may be updated; trail_min
                  and trail_max used by MMAS may be updated
*/
{
    /* TRACE ( printf("SEARCH CONTROL AND STATISTICS\n"); ); */

    if (!(iteration % 100)) {
        population_statistics();
        branching_factor = node_branching(lambda);
        /* printf("\nbest so far %ld, iteration: %ld, time %.2f, b_fac
%.5f\n", (*best_so_far_ant).tour_length, iteration, elapsed_time(
VIRTUAL), branching_factor); */

        if ( mmas_flag && (branching_factor < branch_fac) && (iteration -
restart_found_best > 250) ) {
            /* MAX-MIN Ant System was the first ACO algorithm to use
pheromone trail re-initialisation as implemented
here. Other ACO algorithms may also profit from this mechanism.
*/
            /* printf("INIT TRAILS!!!\n"); */
            (*restart_best_ant).tour_length = INFTY;
            init_pheromone_trails( trail_max );
            compute_total_information();
            restart_iteration = iteration;
            restart_time = elapsed_time( VIRTUAL );
        }
        /* printf("try %li, iteration %li, b-fac %f \n\n",
n_try, iteration, branching_factor); */
    }
}

void as_update( void )
/*
    FUNCTION:      manage global pheromone deposit for Ant System
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  all ants deposit pheromones on matrix "pheromone"
*/

```

```

*/
{
    long int    k;

    /* TRACE ( printf("Ant System pheromone deposit\n"); ); */

    for ( k = 0 ; k < n_ants ; k++ )
        global_update_pheromone( &ant[k] );
}

void eas_update( void )
/*
    FUNCTION:      manage global pheromone deposit for Elitist Ant System
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  all ants plus elitist ant deposit pheromones on matrix
"pheromone"
*/
{
    long int    k;

    /* TRACE ( printf("Elitist Ant System pheromone deposit\n"); ); */

    for ( k = 0 ; k < n_ants ; k++ )
        global_update_pheromone( &ant[k] );
        global_update_pheromone_weighted( best_so_far_ant, elitist_ants );
}

void ras_update( void )
/*
    FUNCTION:      manage global pheromone deposit for Rank-based Ant System
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  the ras_ranks-1 best ants plus the best-so-far ant deposit
pheromone
                    on matrix "pheromone"
    COMMENTS:      this procedure could be implemented slightly faster, but
it is
                    anyway not critical w.r.t. CPU time given that ras_ranks
is
                    typically very small.
*/
{
    long int i, k, b, target;
    long int *help_b;

    /* TRACE ( printf("Rank-based Ant System pheromone deposit\n"); ); */

    help_b = malloc( n_ants * sizeof(long int) );
    for ( k = 0 ; k < n_ants ; k++ )
        help_b[k] = ant[k].tour_length;

    for ( i = 0 ; i < ras_ranks-1 ; i++ ) {
        b = help_b[0]; target = 0;
        for ( k = 0 ; k < n_ants ; k++ ) {

```



```

        if ( help_b[k] < b ). {
        b = help_b[k]; target = k;
        }
    }
    help_b[target] = LONG_MAX;
    global_update_pheromone_weighted( &ant[target], ras_ranks-i-1 );
    }
    global_update_pheromone_weighted( best_so_far_ant, ras_ranks );
    free ( help_b );
}

void mmas_update( void )
/*
    FUNCTION:      manage global pheromone deposit for MAX-MIN Ant System
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS: either the iteration-best or the best-so-far ant deposit
pheromone
                    on matrix "pheromone"
*/
{
    /* we use default upper pheromone trail limit for MMAS and hence we
        do not have to worry regarding keeping the upper limit */

    long int iteration_best_ant;

    /* TRACE ( printf("MAX-MIN Ant System pheromone deposit\n"); ); */

    if ( iteration % u_gb ) {
        iteration_best_ant = find_best();
        global_update_pheromone( &ant[iteration_best_ant] );
    }
    else {
        if ( u_gb == 1 && (restart_found_best - iteration > 50))
            global_update_pheromone( best_so_far_ant );
        else
            global_update_pheromone( restart_best_ant );
    }

    if ( ls_flag ) {
        /* implement the schedule for u_gb as defined in the
            Future Generation Computer Systems article or in Stuetzle's PhD thesis.
            This schedule is only applied if local search is used.
        */
        if ( ( iteration - restart_iteration ) < 25 )
            u_gb = 25;
        else if ( (iteration - restart_iteration) < 75 )
            u_gb = 5;
        else if ( (iteration - restart_iteration) < 125 )
            u_gb = 3;
        else if ( (iteration - restart_iteration) < 250 )
            u_gb = 2;
        else
            u_gb = 1;
    } else
}

```

```

u_gb = 25;
}

void bwass_update( void )
/*
    FUNCTION:      manage global pheromone deposit for Best-Worst Ant System
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  either the iteration-best or the best-so-far ant deposit
pheromone
                    on matrix "pheromone"
*/
{
    long int    iteration_worst_ant, distance_best_worst;

    /* TRACE ( printf("Best-worst Ant System pheromone deposit\n"); ); */

    global_update_pheromone( best_so_far_ant );
    iteration_worst_ant = find_worst();
    bwass_worst_ant_update( best_so_far_ant, &ant[iteration_worst_ant] );
    distance_best_worst = distance_between_ants( best_so_far_ant,
&ant[iteration_worst_ant] );
    /* printf("distance_best_worst %ld, tour length worst
%ld\n", distance_best_worst, ant[iteration_worst_ant].tour_length); */
    if ( distance_best_worst < (long int) (0.05 * (double) n) ) {
        (*restart_best_ant).tour_length = INFITY;
        init_pheromone_trails( trail_0 );
        restart_iteration = iteration;
        restart_time = elapsed_time( VIRTUAL );
        printf("init pheromone trails with %.15f, iteration %ld\n", trail_0, iteration);
    }
    else
        bwass_pheromone_mutation();
}

void acs_global_update( void )
/*
    FUNCTION:      manage global pheromone deposit for Ant Colony System
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  the best-so-far ant deposits pheromone on matrix
"pheromone"
    COMMENTS:      global pheromone deposit in ACS is done per default using
                    the best-so-far ant; Gambardella & Dorigo examined also
iteration-best
                    update (see their IEEE Trans. on Evolutionary Computation article),
                    but did not use it for the published computational results.
*/
{
    /* TRACE ( printf("Ant colony System global pheromone deposit\n"); ); */

    global_acs_pheromone_update( best_so_far_ant );
}

```

```

void pheromone_trail_update( void )
/*
    FUNCTION:      manage global pheromone trail update for the ACO
algorithms
    INPUT:        none
    OUTPUT:       none
    SIDE_EFFECTS: pheromone trails are evaporated and pheromones are
deposited
                  according to the rules defined by the various ACO
algorithms.
*/
{
    /* Simulate the pheromone evaporation of all pheromones; this is not
necessary
    for ACS (see also ACO Book) */
    if ( as_flag || eas_flag || ras_flag || bwas_flag || mmas_flag ) {
    if ( ls_flag ) {
        if ( mmas_flag )
            mmas_evaporation_nn_list();
        else
            evaporation_nn_list();
        /* evaporate only pheromones on arcs of candidate list to make the
pheromone evaporation faster for being able to tackle large TSP
instances. For MMAS additionally check lower pheromone trail limits.
        */
    } else {
        /* if no local search is used, evaporate all pheromone trails */
        evaporation();
    }
    }

    /* Next, apply the pheromone deposit for the various ACO algorithms */
    if ( as_flag )
        as_update();
    else if ( eas_flag )
        eas_update();
    else if ( ras_flag )
        ras_update();
    else if ( mmas_flag )
        mmas_update();
    else if ( bwas_flag )
        bwas_update();
    else if ( acs_flag )
        acs_global_update();

    /* check pheromone trail limits for MMAS; not necessary if local
search is used, because in the local search case lower pheromone trail
limits are checked in procedure mmas_evaporation_nn_list */
    if ( mmas_flag && !ls_flag )
        check_pheromone_trail_limits();

    /* Compute combined information pheromone times heuristic info after
the pheromone update for all ACO algorithms except ACS; in the ACS case
this is already done in the pheromone update procedures of ACS */
    if ( as_flag || eas_flag || ras_flag || mmas_flag || bwas_flag ) {

```

```

    if ( ls_flag ) {
        compute_nn_list_total_information();
    } else {
        compute_total_information();
    }
}

/* --- main program ----- */

int main(int argc, char *argv[]) {
/*
    FUNCTION:      main control for running the ACO algorithms, under PC with
shortcutting
    INPUT:         none
    OUTPUT:        none
    SIDE_EFFECTS:  none
    COMMENTS:      this function controls the run of "max_tries" independent
trials
*/

    long int i;
    int PC_ACO_var, SOS_PC_ACO_var; /* the shortcutting variable */
    int myStream;                  /* the SOS stream id */
    int shortcutted, bMustKeepGoing; /* for termination checking */
    int PC_total_tours, PC_dummyvar;
    int SOS_PC_total_tours, SOS_PC_dummyvar; /* compute total tours across
trials */
    int nRunningCount;

    start_timers();

    nRunningCount = 0;

    /* PC specific initialization */
    PC_init_program(myProc, argc, argv);
    sosgetstream_(&myStream);
    bMustKeepGoing = 1;
    PC_dummyvar = 0;
    PC_total_tours = 0;

    seed = (long int) time(NULL);
    seed = (long int) (seed ^ (myProc + 1));
    seed = rand()^(myProc + rand());

    /*
if(0 == myProc ) seed = 4314263534L;
if(1 == myProc ) seed = 85692002123;
if(2 == myProc ) seed = 3659745454L;
if(3 == myProc ) seed = 98799804L;
*/

    instance.nn_list = compute_nn_lists();
    pheromone = generate_double_matrix( n, n );
    total = generate_double_matrix( n, n );

```

```

/* SOS specific initialization */
shortcutted = 0;
SOS_size = sizeof(PC_ACO_var);
sosaddvar_((int*)&PC_ACO_var, &SOS_size, &SOS_type, &SOS_PC_ACO_var, &ZER);
/* sosaddvar_((int*)&PC_total_tours, &SOS_size, &SOS_type,
&SOS_PC_total_tours, &ZER);
sosaddvar_((int*)&PC_dummyvar, &SOS_size, &SOS_type, &SOS_PC_dummyvar, &ZER);
*/

time_used = elapsed_time( VIRTUAL );
/* printf("Initialization took %.10f seconds\n",time_used); */

for ( n_try = 1 ; n_try <= max_tries ; n_try++ ) {

init_try(n_try);

/* while ( !termination_condition() ) { */
while (bMustKeepGoing) {
/* the PC based SOS shortcut termination condition */
/* Note the use of the implicit variable myProc */
if((iteration > 1) /* && (iteration % 10 == 0) */){
if(sosshortcuttest_() > 0) {
shortcutted = 1;
bMustKeepGoing = 0;
printf("QQ** Node : %d shortcutted at iteration %d with %d tours \n",
myProc, iteration, nRunningCount);
break;
}
}
if(termination_condition() != 0) {
sosshortcut_(&SOS_PC_ACO_var, &myStream);
printf("QQ** Node : %d shortcutting at iteration %d with %d tours \n",
myProc, iteration, nRunningCount);
break;
}
}

construct_solutions();

if ( ls_flag > 0 )
local_search();

update_statistics();

pheromone_trail_update();

search_control_and_statistics();

iteration++;

PC_total_tours = PC_total_tours + n_tours;
nRunningCount += n_tours;
}

```

```

exit_try(n_try);
}

/* printf("QQ* Node:%d:runningcount:%d\n", myProc, nRunningCount); */

MPI_Barrier(MPI_COMM_WORLD);

PC_dummyvar = +{PC_total_tours};
PC_total_tours = PC_dummyvar;

MPI_Barrier(MPI_COMM_WORLD);
/* if(myProc == 0) */
printf("QQ** Sammy Total_tours:%ld\n", PC_total_tours);

exit_program();

free( instance.distance );
free( instance.nn_list );
free( pheromone );
free( total );
free( best_in_try );
free( best_found_at );
free( time_best_found );
free( time_total_run );
for ( i = 0 ; i < n_ants ; i++ ) {
free( ant[i].tour );
free( ant[i].visited );
}
free( ant );
free( (*best_so_far_ant).tour );
free( (*best_so_far_ant).visited );
free( prob_of_selection );

/* PC specific cleanup */
/* MPI_Barrier(MPI_COMM_WORLD); */
PCC_stop();
return (0);
}

```

InOut.h

/*

```
AAAA   CCCC   OOOO   TTTTTT   SSSSS   PPPPP  
AA AA  CC    OO OO   TT    SS     PP  PP  
AAAAAA CC    OO OO   TT     SSSS   PPPPP  
AA AA  CC    OO OO   TT     SS     PP  
AA AA  CCCC   OOOO   TT     SSSSS   PP
```

```
#####  
#####  ACO algorithms for the TSP  #####  
#####
```

```
Version: 2.0  
File:    InOut.h  
Author:  Sammy D'Souza  
Purpose: modifications for PC  
Check:   README and gpl.txt  
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0  
File:    InOut.h  
Author:  Thomas Stuetzle  
Purpose: mainly input / output / statistic routines  
Check:   README and gpl.txt  
Copyright (C) 2002 Thomas Stuetzle
```

*/

/*
/*****

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

*****/

```
#define PROG_ID_STR      "\nACO algorithms for the TSP, v2.0 (uses PC) \n"
#define CALL_SYNTAX_STR "call syntax: acotsp <param-list>\n"

#define LINE_BUF_LEN    100

struct point * read_etsp(const char *tsp_file_name);

struct point * read_etsp(const char *tsp_file_name);

extern long int *best_in_try;
extern long int *best_found_at;
extern double *time_best_found;
extern double *time_total_run;

extern long int n_try; /* number of try */
extern long int n_tours; /* number of constructed tours */
extern long int iteration; /* iteration counter */
extern long int restart_iteration; /* iteration counter */
extern double restart_time; /* remember when restart was done if any */

extern long int max_tries; /* maximum number of independent tries */
extern long int max_tours; /* maximum number of tour constructions in one try */
/*

extern double lambda; /* Parameter to determine branching factor */
extern double branch_fac; /* If branching factor < branch_fac => update
trails */

extern double max_time; /* maximal allowed run time of a try */
extern double time_used; /* time used until some given event */
extern double time_passed; /* time passed until some moment*/
extern long int optimal; /* optimal solution value or bound to find */

extern double mean_ants; /* average tour length */
extern double stddev_ants; /* stddev of tour lengths */
extern double branching_factor; /* average node branching factor when searching
*/
extern double found_branching; /* branching factor when best solution is found
*/

extern long int found_best; /* iteration in which best solution is found
*/
extern long int restart_found_best; /* iteration in which restart-best solution
is found */

extern FILE *report, *comp_report, *stat_report;

extern char name_buf[LINE_BUF_LEN];
extern int opt;
```



```
/* Sammy - Modifications for PC */
extern int nProc;
extern int myProc;

void write_report( void );

void print_default_parameters();

void set_default_parameters();

void init_try( long int ntry );

void output_solution( void );

void exit_try( long int ntry );

void exit_program( void );

void PC_init_program( int myProc, long int argc, char *argv[] );

void printDist(void);

void printHeur(void);

void printTrail(void);

void printTotal(void);

void printProbabilities(void);

void printTour( long int *t );

void printTourFile( long int *t );

void checkTour( long int *t );

void population_statistics ( void );

double node_branching(double l);

void write_params ( void );
```

InOut.c

/*

```
AAAA   CCCC   OOOO   TTTTTT   SSSSS   PPPPP  
AA AA  CC    OO OO   TT    SS     PP  PP  
AAAAAA  CC    OO OO   TT    SSSS   PPPPP  
AA AA  CC    OO OO   TT     SS   PP  
AA AA   CCCC   OOOO   TT    SSSSS   PP
```

```
#####  
#####   ACO algorithms for the TSP   #####  
#####
```

Version: 2.0
File: InOut.c
Author: Sammy D'Souza
Purpose: modifications for PC
Check: README and gpl.txt
Copyright (C) 2007 Sammy D'Souza

Version: 1.0
File: InOut.c
Author: Thomas Stuetzle
Purpose: mainly input / output / statistic routines
Check: README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle

*/

/******

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

*****/

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <limits.h>
#include <time.h>

#include "InOut.h"
#include "TSP.h"
#include "timer.h"
#include "utilities.h"
#include "ants.h"
#include "ls.h"
#include "parse.h"

int PC_myProc;

long int *best_in_try;
long int *best_found_at;
double *time_best_found;
double *time_total_run;

long int n_try; /* try counter */
long int n_tours; /* counter of number constructed tours */
long int iteration; /* iteration counter */
long int restart_iteration; /* remember iteration when restart was done if any */
double restart_time; /* remember time when restart was done if any */
long int max_tries; /* maximum number of independent tries */
long int max_tours; /* maximum number of tour constructions in one try */

double lambda; /* Parameter to determine branching factor */
double branch_fac; /* If branching factor < branch_fac => update trails */

double max_time; /* maximal allowed run time of a try */
double time_used; /* time used until some given event */
double time_passed; /* time passed until some moment */
long int optimal; /* optimal solution or bound to find */

double mean_ants; /* average tour length */
double stddev_ants; /* stddev of tour lengths */
double branching_factor; /* average node branching factor when searching */
double found_branching; /* branching factor when best solution is found */

long int found_best; /* iteration in which best solution is found */
long int restart_found_best; /* iteration in which restart-best solution is found
```

```

*/
/* ----- */
FILE *report, *comp_report, *stat_report;

char name_buf[LINE_BUF_LEN];
int opt;

struct point * read_etstp(const char *tsp_file_name)
/*
    FUNCTION: parse and read instance file
    INPUT:    instance name
    OUTPUT:   list of coordinates for all nodes
    COMMENTS: Instance files have to be in TSPLIB format, otherwise procedure
fails
*/
{
    FILE      *tsp_file;
    char      buf[LINE_BUF_LEN];
    long int   i, j;
    struct point *nodeptr;

    tsp_file = fopen(tsp_file_name, "r");
    if (tsp_file == NULL) {
        fprintf(stderr, "No instance file specified, abort\n");
        exit(1);
    }
    assert(tsp_file != NULL);
    /* if(0 == PC_myProc) printf("\nreading tsp-file %s ... \n", tsp_file_name);
*/

    fscanf(tsp_file, "%s", buf);
    while ( strcmp("NODE_COORD_SECTION", buf) != 0 ) {
        if ( strcmp("NAME", buf) == 0 ) {
            fscanf(tsp_file, "%s", buf);
            /* TRACE ( printf("%s ", buf); ) */
            fscanf(tsp_file, "%s", buf);
            strcpy(instance.name, buf);
            /* TRACE ( printf("%s \n", instance.name); ) */
            buf[0]=0;
        }
        else if ( strcmp("NAME:", buf) == 0 ) {
            fscanf(tsp_file, "%s", buf);
            strcpy(instance.name, buf);
            /* TRACE ( printf("%s \n", instance.name); ) */
            buf[0]=0;
        }
        else if ( strcmp("COMMENT", buf) == 0 ){
            fgets(buf, LINE_BUF_LEN, tsp_file);
            /* TRACE ( printf("%s", buf); ) */
            buf[0]=0;
        }
        else if ( strcmp("COMMENT:", buf) == 0 ){
            fgets(buf, LINE_BUF_LEN, tsp_file);

```

```

/*      TRACE ( printf("%s", buf); ) */
    buf[0]=0;
}
    else if ( strcmp("TYPE", buf) == 0 ) {
        fscanf(tsp_file, "%s", buf);
/*      TRACE ( printf("%s ", buf); ) */
        fscanf(tsp_file, "%s", buf);
/*      TRACE ( printf("%s\n", buf); ) */
        if( strcmp("TSP", buf) != 0 ) {
            fprintf(stderr, "\n Not a TSP instance in TSPLIB format !!\n");
            exit(1);
        }
        buf[0]=0;
    }
    else if ( strcmp("TYPE:", buf) == 0 ) {
        fscanf(tsp_file, "%s", buf);
/*      TRACE ( printf("%s\n", buf); ) */
        if( strcmp("TSP", buf) != 0 ) {
            fprintf(stderr, "\n Not a TSP instance in TSPLIB format !!\n");
            exit(1);
        }
        buf[0]=0;
    }
    else if( strcmp("DIMENSION", buf) == 0 ){
        fscanf(tsp_file, "%s", buf);
/*      TRACE ( printf("%s ", buf); ); */
        fscanf(tsp_file, "%ld", &n);
        instance.n = n;
/*      TRACE ( printf("%ld\n", n); ); */
        assert ( n > 2 && n < 6000);
        buf[0]=0;
    }
    else if ( strcmp("DIMENSION:", buf) == 0 ) {
        fscanf(tsp_file, "%ld", &n);
        instance.n = n;
/*      TRACE ( printf("%ld\n", n); ); */
        assert ( n > 2 && n < 6000);
        buf[0]=0;
    }
    else if( strcmp("DISPLAY_DATA_TYPE", buf) == 0 ){
        fgets(buf, LINE_BUF_LEN, tsp_file);
/*      TRACE ( printf("%s", buf); ); */
        buf[0]=0;
    }
    else if ( strcmp("DISPLAY_DATA_TYPE:", buf) == 0 ) {
        fgets(buf, LINE_BUF_LEN, tsp_file);
/*      TRACE ( printf("%s", buf); ); */
        buf[0]=0;
    }
    else if( strcmp("EDGE_WEIGHT_TYPE", buf) == 0 ){
        buf[0]=0;
        fscanf(tsp_file, "%s", buf);
/*      TRACE ( printf("%s ", buf); ); */
        buf[0]=0;
        fscanf(tsp_file, "%s", buf);

```

```

/*      TRACE ( printf("%s\n", buf); ); */
    if ( strcmp("EUC_2D", buf) == 0 ) {
distance = round_distance;
    }
    else if ( strcmp("CEIL_2D", buf) == 0 ) {
distance = ceil_distance;
    }
    else if ( strcmp("GEO", buf) == 0 ) {
distance = geo_distance;
    }
    else if ( strcmp("ATT", buf) == 0 ) {
distance = att_distance;
    }
    else
fprintf(stderr, "EDGE_WEIGHT_TYPE %s not implemented\n", buf);
    strcpy(instance.edge_weight_type, buf);
    buf[0]=0;
}
else if( strcmp("EDGE_WEIGHT_TYPE:", buf) == 0 ){
/* set pointer to appropriate distance function; has to be one of
EUC_2D, CEIL_2D, GEO, or ATT. Everything else fails */
buf[0]=0;
fscanf(tsp_file, "%s", buf);
/*      TRACE ( printf("%s\n", buf); ) */
    if ( strcmp("EUC_2D", buf) == 0 ) {
distance = round_distance;
    }
    else if ( strcmp("CEIL_2D", buf) == 0 ) {
distance = ceil_distance;
    }
    else if ( strcmp("GEO", buf) == 0 ) {
distance = geo_distance;
    }
    else if ( strcmp("ATT", buf) == 0 ) {
distance = att_distance;
    }
    else {
fprintf(stderr, "EDGE_WEIGHT_TYPE %s not implemented\n", buf);
exit(1);
    }
    strcpy(instance.edge_weight_type, buf);
    buf[0]=0;
}
buf[0]=0;
fscanf(tsp_file, "%s", buf);
}

    if( strcmp("NODE_COORD_SECTION", buf) == 0 ){
/*      TRACE ( printf("found section containing the node coordinates\n"); ) */
    }
    else{
fprintf(stderr, "\n\nSome error ocurred finding start of coordinates from tsp
file !!\n");
exit(1);
    }
}

```

```

    if( (nodeptr = malloc(sizeof(struct point) * n)) == NULL )
exit(EXIT_FAILURE);
    else {
    for ( i = 0 ; i < n ; i++ ) {
        fscanf(tsp_file,"%ld %lf %lf", &j, &nodeptr[i].x, &nodeptr[i].y );
    }
}
/* TRACE ( printf("number of cities is %ld\n",n); ) */
/* TRACE ( printf("\n... done\n"); ) */
return (nodeptr);
}

void write_report( void )
/*
    FUNCTION: output some info about trial (best-so-far solution quality,
time)
    INPUT:     none
    OUTPUT:    none
    COMMENTS:  none
*/
{
    /* printf("Node:%d:best:%ld:iteration:%ld:tours:%ld:time:%.2f\n",PC_myProc,
(*best_so_far_ant).tour_length,iteration,n_tours,elapsed_time( VIRTUAL)); */
    fprintf(comp_report,"best %ld\t iteration %ld\t tours %ld\t time
%.3f\n", (*best_so_far_ant).tour_length,iteration,n_tours,time_used);
}

void print_default_parameters()
/*
    FUNCTION: output default parameter settings
    INPUT:     none
    OUTPUT:    none
    COMMENTS:  none
*/
{
    fprintf(stderr,"\nDefault parameter settings are:\n\n");
    fprintf(stderr,"max_tries\t\t %ld\n", max_tries);
    fprintf(stderr,"max_tours\t\t %ld\n", max_tours);
    fprintf(stderr,"max_time\t\t %.2f\n", max_time);
    fprintf(stderr,"optimum\t\t\t %ld\n", optimal);
    fprintf(stderr,"n_ants\t\t\t %ld\n", n_ants);
    fprintf(stderr,"nn_ants\t\t\t %ld\n", nn_ants);
    fprintf(stderr,"alpha\t\t\t %.2f\n", alpha);
    fprintf(stderr,"beta\t\t\t %.2f\n", beta);
    fprintf(stderr,"rho\t\t\t %.2f\n", rho);
    fprintf(stderr,"q_0\t\t\t %.2f\n", q_0);
    fprintf(stderr,"elitist_ants\t\t 0\n");
    fprintf(stderr,"ras_ranks\t\t 6\n");
    fprintf(stderr,"ls_flag\t\t\t %ld\n", ls_flag);
    fprintf(stderr,"nn_ls\t\t\t %ld\n", nn_ls);
    fprintf(stderr,"dlb_flag\t\t %ld\n", dlb_flag);
    fprintf(stderr,"as_flag\t\t\t %ld\n", as_flag);
    fprintf(stderr,"eas_flag\t\t %ld\n", eas_flag);
    fprintf(stderr,"ras_flag\t\t %ld\n", ras_flag);
}

```

```

    fprintf(stderr,"mmas_flag\t\t %ld\n", mmas_flag);
    fprintf(stderr,"bwas_flag\t\t %ld\n", bwas_flag);
    fprintf(stderr,"acs_flag\t\t %ld\n", acs_flag);
}

void set_default_parameters()
/*
    FUNCTION: set default parameter settings
    INPUT: none
    OUTPUT: none
    COMMENTS: none
*/
{
    ls_flag      = 3;      /* per default run 3-opt*/
    dlb_flag     = TRUE;   /* apply don't look bits in local search */
    nn_ls        = 20;     /* use fixed radius search in the 20 nearest
neighbours */
    n_ants       = 25;     /* number of ants */
    nn_ants      = 20;     /* number of nearest neighbours in tour construction
*/
    alpha        = 1.0;
    beta         = 2.0;
    rho          = 0.5;
    q_0          = 0.0;
    max_tries    = 10;
    max_tours    = 100;
    max_time     = 10.0;
    optimal      = 1;
    branch_fac   = 1.00001;
    as_flag      = FALSE;
    eas_flag     = FALSE;
    ras_flag     = FALSE;
    mmas_flag    = TRUE;
    u_gb         = INFTY;
    bwas_flag    = FALSE;
    acs_flag     = FALSE;
    ras_ranks    = 6;
    elitist_ants = 100;
}

void population_statistics (void )
/*
    FUNCTION: compute some population statistics like average tour
length,
                standard deviations, average distance, branching-factor
and
                output to a file gathering statistics
    INPUT: none
    OUTPUT: none
    (SIDE)EFFECTS: none
*/
{
    long int j, k;
    long int *l;
    double pop_mean, pop_stddev, avg_distance = 0.0;

```



```

    l = calloc(n_ants, sizeof(long int));
    for( k = 0 ; k < n_ants ; k++ ) {
l[k] = ant[k].tour_length;
    }

    pop_mean = mean( l, n_ants);
    pop_stddev = std_deviation( l, n_ants, pop_mean );
    branching_factor = node_branching(lambda);

    for ( k = 0 ; k < n_ants-1 ; k++ )
for ( j = k+1 ; j < n_ants ; j++ ) {
    avg_distance += (double)distance_between_ants( &ant[k], &ant[j]);
}
    avg_distance /= ((double)n_ants * (double)(n_ants-1) / 2.);

fprintf(stat_report,"%ld\t%.1f\t%.5f\t%.7f\t%.5f\t%.1f\t%.1f\t%.5f\n",iteration,
pop_mean, pop_stddev, pop_stddev / pop_mean,
branching_factor, (branching_factor - 1.) * (double)n, avg_distance,
avg_distance / (double)n);
}

double node_branching(double l)
/*
    FUNCTION:      compute the average node lambda-branching factor
    INPUT:         lambda value
    OUTPUT:        average node branching factor
    (SIDE)EFFECTS: none
    COMMENTS:      see the ACO book for a definition of the average node
                  lambda-branching factor
*/
{
    long int  i, m;
    double    min, max, cutoff;
    double    avg;
    double    *num_branches;

    num_branches = calloc(n, sizeof(double));

    for ( m = 0 ; m < n ; m++ ) {
        /* determine max, min to calculate the cutoff value */
        min = pheromone[m][instance.nn_list[m][1]];
        max = pheromone[m][instance.nn_list[m][1]];
        for ( i = 1 ; i < nn_ants ; i++ ) {
            if ( pheromone[m][instance.nn_list[m][i]] > max )
max = pheromone[m][instance.nn_list[m][i]];
            if ( pheromone[m][instance.nn_list[m][i]] < min )
min = pheromone[m][instance.nn_list[m][i]];
        }
        cutoff = min + l * (max - min);

        for ( i = 0 ; i < nn_ants ; i++ ) {
            if ( pheromone[m][instance.nn_list[m][i]] > cutoff )

```

```

num_branches[m] += 1.;
}
}
avg = 0.;
for ( m = 0 ; m < n ; m++ ) {
    avg += num_branches[m];
}
free ( num_branches );
/* Norm branching factor to minimal value 1 */
return ( avg / (double) (n * 2) );
}

void output_solution( void )
/*
    FUNCTION:      output a solution together with node coordinates
    INPUT:         none
    OUTPUT:        none
    COMMENTS:      not used in the default implementation but may be useful
anyway
*/
{
    long int i;

    for ( i = 0 ; i < n ; i++ ) {
        fprintf(stat_report," %ld %f
%f\n", (*best_so_far_ant).tour[i], instance.nodeptr[(*best_so_far_ant).tour[i]].x,
instance.nodeptr[(*best_so_far_ant).tour[i]].y);
    }
    printf("\n");
}

void exit_try( long int ntry )
/*
    FUNCTION:      save some statistical information on a trial once it
finishes
    INPUT:         trial number
    OUTPUT:        none
    COMMENTS:
*/
{
    checkTour( (*best_so_far_ant).tour );
/*    printTourFile( (*best_so_far_ant).tour ); */

    /* printf("\n Best Solution in try %ld is %ld\n",ntry,
(*best_so_far_ant).tour_length); */
    fprintf(report,"Best: %ld\t Iterations: %6ld\t B-Fac %.5f\t Time %.2f\t
Tot.time %.2f\n", (*best_so_far_ant).tour_length, found_best,
found_branching, time_used, elapsed_time( VIRTUAL ));
    /* printf(" Best Solution was found after %ld iterations\n", found_best); */

    best_in_try[ntry] = (*best_so_far_ant).tour_length;
    best_found_at[ntry] = found_best;
    time_best_found[ntry] = time_used;
    time_total_run[ntry] = elapsed_time( VIRTUAL );
}

```

```

    /* printf("\ntry %ld, Best %ld, found at iteration %ld, found at time
%f\n",ntry, best_in_try[ntry], best_found_at[ntry],
time_best_found[ntry]); */

    fprintf(comp_report,"end try %ld\n\n",ntry);
    fprintf(stat_report,"end try %ld\n\n",ntry);
    /* TRACE (output_solution();) */
    fflush(report);
    fflush(comp_report);
    fflush(stat_report);
}

void exit_program( void )
/*
    FUNCTION:      save some final statistical information on a trial once it
finishes
    INPUT:         none
    OUTPUT:        none
    COMMENTS:
*/
{
    long int best_tour_length, worst_tour_length;
    double   t_avgbest, t_stdbest, t_avgtotal, t_stdtotal;
    double   avg_sol_quality = 0., avg_cyc_to_bst = 0., stddev_best,
stddev_iterations;

    best_tour_length = best_of_vector( best_in_try ,max_tries );
    worst_tour_length = worst_of_vector( best_in_try , max_tries );

    avg_cyc_to_bst = mean( best_found_at , max_tries );
    stddev_iterations = std_deviation( best_found_at, max_tries, avg_cyc_to_bst );

    avg_sol_quality = mean( best_in_try , max_tries );
    stddev_best = std_deviation( best_in_try, max_tries, avg_sol_quality);

    t_avgbest = meanr( time_best_found, max_tries );
    printf(" t_avgbest = %f\n", t_avgbest );
    t_stdbest = std_deviationr( time_best_found, max_tries, t_avgbest);

    t_avgtotal = meanr( time_total_run, max_tries );
    printf(" t_avgtotal = %f\n", t_avgtotal );
    t_stdtotal = std_deviationr( time_total_run, max_tries, t_avgtotal);

    fprintf(report,"\nAverage-Best: %.2f\t Average-Iterations: %.2f",
avg_sol_quality, avg_cyc_to_bst);
    fprintf(report,"\nStddev-Best: %.2f \t Stddev Iterations: %.2f", stddev_best,
stddev_iterations);
    fprintf(report,"\nBest try: %ld\t\t Worst try: %ld\n", best_tour_length,
worst_tour_length);
    fprintf(report,"\nAvg.time-best: %.2f stddev.time-best: %.2f\n", t_avgbest,
t_stdbest);
    fprintf(report,"\nAvg.time-total: %.2f stddev.time-total: %.2f\n", t_avgtotal,
t_stdtotal);

    if ( optimal > 0 ) {

```

```

    fprintf(report, " excess best = %f, excess average = %f, excess worst =
%f\n", (double) (best_tour_length - optimal) /
(double) optimal, (double) (avg_sol_quality - optimal) /
(double) optimal, (double) (worst_tour_length - optimal) / (double) optimal);
}

```

```

    fprintf(comp_report, "end problem %s\n", instance.name);
}

```

```

void PC_init_program( int myProc, long int argc, char *argv[] )

```

```

/*

```

```

    FUNCTION:      initialize the program,
    INPUT:         program arguments, needed for parsing commandline
    OUTPUT:        none
    COMMENTS:

```

```

*/

```

```

{

```

```

    PC_myProc = myProc;

```

```

    char temp_buffer[LINE_BUF_LEN];

```

```

    if(0 == PC_myProc) printf(PROG_ID_STR);
    set_default_parameters();
    setbuf(stdout, NULL);
    PC_parse_commandline(myProc, argc, argv);

```

```

    assert (n_ants < MAX_ANTS-1);
    assert (nn_ants < MAX_NEIGHBOURS);
    assert (nn_ants > 0);
    assert (nn_ls > 0);
    assert (max_tries <= MAXIMUM_NO_TRIES);

```

```

    best_in_try = calloc(max_tries, sizeof(long int));
    best_found_at = calloc(max_tries, sizeof(long int));
    time_best_found = calloc(max_tries, sizeof(double));
    time_total_run = calloc(max_tries, sizeof(double));

```

```

    seed = (long int) time( NULL );

```

```

/* TRACE ( printf("read problem data ..\n\n"); ) */

```

```

    instance.nodeptr = read_etsp(name_buf);

```

```

/* TRACE ( printf("\n .. done\n\n"); ) */

```

```

    nn_ls = MIN(n-1, nn_ls);

```

```

    sprintf(temp_buffer, "best.%s", instance.name);

```

```

/* TRACE ( printf("%s\n", temp_buffer); ) */

```

```

    report = fopen(temp_buffer, "w");

```

```

    sprintf(temp_buffer, "cmp.%s", instance.name);

```

```

/* TRACE ( printf("%s\n", temp_buffer); ) */

```

```

    comp_report = fopen(temp_buffer, "w");

```

```

    sprintf(temp_buffer, "stat.%s", instance.name);

```

```

/* TRACE ( printf("%s\n", temp_buffer); ) */

```

```

    stat_report = fopen(temp_buffer, "w");

```

```

/* if(0 == PC_myProc) printf("calculating distance matrix .."); */
instance.distance = compute_distances();
/* if(0 == PC_myProc) printf(" .. done\n"); */
if(0 == PC_myProc) write_params();
fprintf(comp_report, "begin problem %s\n", name_buf);

/* if(0 == PC_myProc) printf("allocate ants' memory .."); */
allocate_ants();
/* PC_allocate_ants(); */
/* if(0 == PC_myProc) printf(" .. done\n"); */
DEBUG ( assert ( nn_ls < n && 0 < nn_ls ); )

/* if(0 == PC_myProc) printf("\nFinally set ACO algorithm specific parameters,
typically done as proposed in literature\n"); */
/* default setting for elitist_ants is 0; if EAS is applied and
option elitist_ants is not used, we set the default to
elitist_ants = n */
if ( eas_flag && elitist_ants == 0 )
    elitist_ants = n;
}

void printDist(void)
/*
    FUNCTION:      print distance matrix
    INPUT:         none
    OUTPUT:        none
*/
{
    long int i, j;

    printf("Distance Matrix:\n");
    for ( i = 0 ; i < n ; i++ ) {
        printf("From %ld: ", i);
        for ( j = 0 ; j < n - 1 ; j++ ) {
            printf(" %ld", instance.distance[i][j]);
        }
        printf(" %ld\n", instance.distance[i][n-1]);
        printf("\n");
    }
    printf("\n");
}

void printHeur(void)
/*
    FUNCTION:      print heuristic information
    INPUT:         none
    OUTPUT:        none
*/
{
    long int i, j;

    printf("Heuristic information:\n");
    for ( i = 0 ; i < n ; i++ ) {
        printf("From %ld: ", i);

```

```

    for ( j = 0 ; j < n - 1 ; j++ ) {
        printf(" %.3f ", HEURISTIC(i,j));
    }
    printf(" %.3f\n", HEURISTIC(i,j));
    printf("\n");
}
printf("\n");
}

void printTrail(void)
/*
    FUNCTION:      print pheromone trail values
    INPUT:         none
    OUTPUT:        none
*/
{
    long int i,j;

    printf("pheromone Trail matrix, iteration: %ld\n\n",iteration);
    for ( i = 0 ; i < n ; i++) {
        printf("From %ld: ",i);
        for ( j = 0 ; j < n ; j++ ) {
            printf(" %.10f ", pheromone[i][j]);
            if ( pheromone[i][j] > 1.0 )
                printf("XXXXX\n");
        }
        printf("\n");
    }
    printf("\n");
}

void printTotal(void)
/*
    FUNCTION:      print values of pheromone times heuristic information
    INPUT:         none
    OUTPUT:        none
*/
{
    long int i, j;

    printf("combined pheromone and heuristic info\n\n");
    for (i=0; i < n; i++) {
        for (j = 0; j < n - 1 ; j++) {
            printf(" %.15f &", total[i][j]);
            if ( total[i][j] > 1.0 )
                printf("XXXXX\n");
        }
        printf(" %.15f\n", total[i][n-1]);
        if ( total[i][n-1] > 1.0 )
            printf("XXXXX\n");
    }
    printf("\n");
}

void printProbabilities(void)

```

```

/*
    FUNCTION:      prints the selection probabilities as encountered by an
ant
    INPUT:        none
    OUTPUT:       none
    COMMENTS:     this computation assumes that no choice has been made yet.
*/
{
    long int i, j;
    double *p;
    double sum_prob;

    printf("Selection Probabilities, iteration: %ld\n", iteration);
    p = calloc( n, sizeof(double) );

    for (i=0; i < n; i++) {
        printf("From %ld: ", i);
        sum_prob = 0.;
        for ( j = 0 ; j < n ; j++) {
            if ( i == j )
                p[j] = 0.;
            else
                p[j] = total[i][j];
            sum_prob += p[j];
        }
        for ( j = 0 ; j < n ; j++) {
            p[j] = p[j] / sum_prob;
        }
        for ( j = 0 ; j < n-1 ; j++) {
            printf(" %.5f ", p[j]);
        }
        printf(" %.5f\n", p[n-1]);
        if (!(j % 26)) {
            printf("\n");
        }
        printf("\n");
    }
    printf("\n");
    free ( p );
}

void printTour( long int *t )
/*
    FUNCTION:      print the tour *t
    INPUT:        pointer to a tour
    OUTPUT:       none
*/
{
    long int i;

    printf("\n");
    for( i = 0 ; i <= n ; i++ ) {
        if (!(i%25))
            printf("\n");
        printf("%ld ", t[i]);
    }
}

```

```

    }
    printf("\n");
    printf("Tour length = %ld\n\n",compute_tour_length( t ));
}

void printTourFile( long int *t )
/*
    FUNCTION:      print the tour *t to comp.tsplibfile
    INPUT:         pointer to a tour
    OUTPUT:        none
*/
{
    long int    i;

    fprintf(comp_report,"begin solution\n");
    for( i = 0 ; i < n ; i++ ) {
    fprintf(comp_report,"%ld ", t[i]);
    }
    fprintf(comp_report,"\n");
    fprintf(comp_report,"Tour length %ld\n",compute_tour_length( t ));
    fprintf(comp_report,"end solution\n");
}

void checkTour( long int *t )
/*
    FUNCTION:      make a simple check whether tour *t can be feasible
    INPUT:         pointer to a tour
    OUTPUT:        none
*/
{
    long int    i, sum=0;

    for( i = 0 ; i < n ; i++ ) {
    sum += t[i];
    }
    if ( sum != (n-1) * n / 2 ) {
    fprintf(stderr,"Next tour must be flawed !!\n");
    printTour( t );
    exit(1);
    }
}

void write_params( void ).
/*
    FUNCTION:      writes chosen parameter settings in standard output and in
                  report files
    INPUT:         none
    OUTPUT:        none
*/
{
    /*
    printf("\nParameter-settings: \n\n");
    printf("max-tries %ld\n", max_tries);
    printf("max-tours %ld\n", max_tours);
    printf("optimum %ld\n", optimal);

```



```

printf("time %f\n", max_time);
printf("num-ants %ld\n", n_ants);
printf("num-neigh %ld\n", nn_ants);
printf("alpha %f\n", alpha);
printf("beta %f\n", beta);
printf("rho %f\n", rho);
printf("q_0 %f\n", q_0);
printf("branch-up %f\n", branch_fac);
printf("ls_flag %ld\n", ls_flag);
printf("nn_ls %ld\n", nn_ls);
printf("dlb_flag %ld\n", dlb_flag);
printf("as_flag %ld\n", as_flag);
printf("eas_flag %ld\n", eas_flag);
printf("ras_flag %ld\n", ras_flag);
printf("mmas_flag %ld\n", mmas_flag);
printf("bwas_flag %ld\n", bwas_flag);
printf("acs_flag %ld\n", acs_flag);
printf("\n");
*/
fprintf(report, "\nParameter-settings: \n\n");
fprintf(report, "max-tries %ld\n", max_tries);
fprintf(report, "max-tours %ld\n", max_tours);
fprintf(report, "optimum %ld\n", optimal);
fprintf(report, "time %f\n", max_time);
fprintf(report, "num-ants %ld\n", n_ants);
fprintf(report, "num-neigh %ld\n", nn_ants);
fprintf(report, "alpha %f\n", alpha);
fprintf(report, "beta %f\n", beta);
fprintf(report, "rho %f\n", rho);
fprintf(report, "q_0 %f\n", q_0);
fprintf(report, "branch-up %f\n", branch_fac);
fprintf(report, "ls_flag %ld\n", ls_flag);
fprintf(report, "nn_ls %ld\n", nn_ls);
fprintf(report, "dlb_flag %ld\n", dlb_flag);
fprintf(report, "as_flag %ld\n", as_flag);
fprintf(report, "eas_flag %ld\n", eas_flag);
fprintf(report, "ras_flag %ld\n", ras_flag);
fprintf(report, "mmas_flag %ld\n", mmas_flag);
fprintf(report, "bwas_flag %ld\n", bwas_flag);
fprintf(report, "acs_flag %ld\n", acs_flag);
fprintf(report, "\n");
fprintf(comp_report, "%s", PROG_ID_STR);
fprintf(comp_report, "\nParameter-settings: \n\n");
fprintf(comp_report, "max-tries %ld\n", max_tries);
fprintf(comp_report, "max-tours %ld\n", max_tours);
fprintf(comp_report, "optimum %ld\n", optimal);
fprintf(comp_report, "time %f\n", max_time);
fprintf(comp_report, "num-ants %ld\n", n_ants);
fprintf(comp_report, "num-neigh %ld\n", nn_ants);
fprintf(comp_report, "alpha %f\n", alpha);
fprintf(comp_report, "beta %f\n", beta);
fprintf(comp_report, "rho %f\n", rho);
fprintf(comp_report, "q_0 %f\n", q_0);
fprintf(comp_report, "branch-up %f\n", branch_fac);
fprintf(comp_report, "ls_flag %ld\n", ls_flag);

```

```
fprintf(comp_report, "nn_ls %ld\n", nn_ls);  
fprintf(comp_report, "dlb_flag %ld\n", dlb_flag);  
fprintf(comp_report, "as_flag %ld\n", as_flag);  
fprintf(comp_report, "eas_flag %ld\n", eas_flag);  
fprintf(comp_report, "ras_flag %ld\n", ras_flag);  
fprintf(comp_report, "mmas_flag %ld\n", mmas_flag);  
fprintf(comp_report, "bwas_flag %ld\n", bwas_flag);  
fprintf(comp_report, "acs_flag %ld\n", acs_flag);  
fprintf(comp_report, "\n");  
}
```

ls.h

/*

AAAA CCCC OOOO TTTTT SSSS PPPP
AA AA CC OO OO TT SS PP PP
AAAAA CC OO OO TT SSSS PPPP
AA AA CC OO OO TT SS PP
AA AA CCCC OOOO TT SSSS PP

ACO algorithms for the TSP #####
#####

Version: 2.0
File: ls.h
Author: Sammy D'Souza
Purpose: modifications for PC
Check: README and gpl.txt
Copyright (C) 2007 Sammy D'Souza

Version: 1.0
File: ls.h
Author: Thomas Stuetzle
Purpose: header file for local search routines
Check: README and gpl.txt
Copyright (C) 1999 Thomas Stuetzle

*/

/******

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

*****/

extern long int ls_flag;

extern long int nn_ls;

extern long int dlb_flag;

void two_opt_first(**long int** *tour);

void two_h_opt_first(**long int** *tour);

long int three_opt_first(**long int** *tour);

ls.c

/*

```

AAAA   CCCC   OOOO   TTTTTT   SSSSS   PPPPP
AA AA  CC    OO OO   TT    SS     PP  PP
AAAAAA  CC    OO OO   TT    SSSS   PPPPP
AA AA  CC    OO OO   TT     SS   PP
AA AA   CCCC   OOOO   TT    SSSSS   PP

```

```

#####
#####      ACO algorithms for the TSP      #####
#####

```

```

Version: 2.0
File:    ls.c
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza

```

```

Version: 1.0
File:    ls.c
Author:  Thomas Stuetzle
Purpose: implementation of local search routines
Check:   README and gpl.txt
Copyright (C) 1999 Thomas Stuetzle

```

*/

/******

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

*****/

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <limits.h>
```

```
#include "InOut.h"
#include "TSP.h"
#include "ants.h"
#include "utilities.h"
```

```
long int ls_flag;          /* indicates whether and which local search is used
*/
```

```
long int nn_ls;          /* maximal depth of nearest neighbour lists used in
the
```

```
local search */
```

```
long int dlb_flag = TRUE; /* flag indicating whether don't look bits are used.
I recommend
```

```
to always use it if local search is applied */
```

```
long int * generate_random_permutation( long int n )
```

```
/*
```

```
FUNCTION: generate a random permutation of the integers 0 .. n-1
```

```
INPUT: length of the array
```

```
OUTPUT: pointer to the random permutation
```

```
(SIDE)EFFECTS: the array holding the random permutation is allocated in
this
```

```
function. Don't forget to free again the memory!
```

```
COMMENTS: only needed by the local search procedures
```

```
*/
```

```
{
```

```
long int i, help, node, tot_assigned = 0;
```

```
double rnd;
```

```
long int *r;
```

```
r = malloc(n * sizeof(int));
```

```
for ( i = 0 ; i < n; i++)
```

```
    r[i] = i;
```

```
for ( i = 0 ; i < n ; i++ ) {
```

```
    /* find (randomly) an index for a free unit */
```

```
    rnd = ran01 ( &seed );
```

```
    node = (long int) (rnd * (n - tot_assigned));
```

```
    assert( i + node < n );
```

```
    help = r[i];
```

```
    r[i] = r[i+node];
```

```
    r[i+node] = help;
```

```
    tot_assigned++;
```

```
}
```

```

    return r;
}

void two_opt_first( long int *tour )
/*
    FUNCTION:      2-opt a tour
    INPUT:         pointer to the tour that undergoes local optimization
    OUTPUT:        none
    (SIDE)EFFECTS: tour is 2-opt
    COMMENTS:      the neighbourhood is scanned in random order (this need
                    not be the best possible choice). Concerning the speed-ups
used
                    here consult, for example, Chapter 8 of
                    Holger H. Hoos and Thomas Stuetzle,
                    Stochastic Local Search---Foundations and Applications,
                    Morgan Kaufmann Publishers, 2004.
                    or some of the papers online available from David S. Johnson.
*/
{
    long int c1, c2;           /* cities considered for an exchange */
    long int s_c1, s_c2;      /* successor cities of c1 and c2 */
    long int p_c1, p_c2;      /* predecessor cities of c1 and c2 */
    long int pos_c1, pos_c2;   /* positions of cities c1, c2 */
    long int i, j, h, l;
    long int improvement_flag, improve_node, help, n_improves = 0,
n_exchanges=0;
    long int h1=0, h2=0, h3=0, h4=0;
    long int radius;          /* radius of nn-search */
    long int gain = 0;
    long int *random_vector;
    long int *pos;            /* positions of cities in tour */
    long int *dlb;            /* vector containing don't look bits */

    pos = malloc(n * sizeof(long int));
    dlb = malloc(n * sizeof(long int));
    for ( i = 0 ; i < n ; i++ ) {
pos[tour[i]] = i;
dlb[i] = FALSE;
    }

    improvement_flag = TRUE;
    random_vector = generate_random_permutation( n );

    while ( improvement_flag ) {

improvement_flag = FALSE;

for ( l = 0 ; l < n; l++ ) {

    c1 = random_vector[l];
    DEBUG ( assert ( c1 < n && c1 >= 0); )
    if ( dlb_flag && dlb[c1] )
        continue;
    improve_node = FALSE;
    pos_c1 = pos[c1];

```

```

s_c1 = tour[pos_c1+1];
radius = instance.distance[c1][s_c1];

/* First search for c1's nearest neighbours, use successor of c1 */
for ( h = 0 ; h < nn_ls ; h++ ) {
c2 = instance.nn_list[c1][h]; /* exchange partner, determine its position */
if ( radius > instance.distance[c1][c2] ) {
    s_c2 = tour[pos[c2]+1];
    gain = - radius + instance.distance[c1][c2] +
instance.distance[s_c1][s_c2] - instance.distance[c2][s_c2];
    if ( gain < 0 ) {
h1 = c1; h2 = s_c1; h3 = c2; h4 = s_c2;
improve_node = TRUE;
goto exchange2opt;
    }
}
else
    break;
}
/* Search one for next c1's h-nearest neighbours, use predecessor c1 */
if ( pos_c1 > 0 )
p_c1 = tour[pos_c1-1];
else
p_c1 = tour[n-1];
radius = instance.distance[p_c1][c1];
for ( h = 0 ; h < nn_ls ; h++ ) {
c2 = instance.nn_list[c1][h]; /* exchange partner, determine its position
*/
if ( radius > instance.distance[c1][c2] ) {
    pos_c2 = pos[c2];
    if ( pos_c2 > 0 )
p_c2 = tour[pos_c2-1];
    else
p_c2 = tour[n-1];
    if ( p_c2 == c1 )
continue;
    if ( p_c1 == c2 )
continue;
    gain = - radius + instance.distance[c1][c2] +
instance.distance[p_c1][p_c2] - instance.distance[p_c2][c2];
    if ( gain < 0 ) {
h1 = p_c1; h2 = c1; h3 = p_c2; h4 = c2;
improve_node = TRUE;
goto exchange2opt;
    }
}
else
    break;
}
if ( improve_node ) {
exchange2opt:
n_exchanges++;
improvement_flag = TRUE;
dlb[h1] = FALSE; dlb[h2] = FALSE;
dlb[h3] = FALSE; dlb[h4] = FALSE;
}

```



```

/* Now perform move */
if ( pos[h3] < pos[h1] ) {
    help = h1; h1 = h3; h3 = help;
    help = h2; h2 = h4; h4 = help;
}
if ( pos[h3] - pos[h2] < n / 2 + 1 ) {
    /* reverse inner part from pos[h2] to pos[h3] */
    i = pos[h2]; j = pos[h3];
    while ( i < j ) {
        c1 = tour[i];
        c2 = tour[j];
        tour[i] = c2;
        tour[j] = c1;
        pos[c1] = j;
        pos[c2] = i;
        i++; j--;
    }
}
else {
    /* reverse outer part from pos[h4] to pos[h1] */
    i = pos[h1]; j = pos[h4];
    if ( j > i )
        help = n - ( j - i ) + 1;
    else
        help = ( i - j ) + 1;
    help = help / 2;
    for ( h = 0 ; h < help ; h++ ) {
        c1 = tour[i];
        c2 = tour[j];
        tour[i] = c2;
        tour[j] = c1;
        pos[c1] = j;
        pos[c2] = i;
        i--; j++;
        if ( i < 0 )
            i = n-1;
        if ( j >= n )
            j = 0;
    }
    tour[n] = tour[0];
}
} else {
    dlb[c1] = TRUE;
}
}
if ( improvement_flag ) {
    n_improves++;
}
}
free( random_vector );
free( dlb );
free( pos );
}

```

```

void two_h_opt_first( long int *tour )
/*
    FUNCTION:      2-h-opt a tour
    INPUT:         pointer to the tour that undergoes local optimization
    OUTPUT:        none
    (SIDE)EFFECTS: tour is 2-h-opt
    COMMENTS:      for details on 2-h-opt see J. L. Bentley. Fast algorithms
for geometric
                    traveling salesman problems. ORSA Journal on Computing,
                    4(4):387--411, 1992.
                    The neighbourhood is scanned in random order (this need
                    not be the best possible choice). Concerning the speed-ups
used
                    here consult, for example, Chapter 8 of
                    Holger H. Hoos and Thomas Stuetzle,
                    Stochastic Local Search---Foundations and Applications,
                    Morgan Kaufmann Publishers, 2004.
                    or some of the papers online available from David S. Johnson.
*/
{

    long int c1, c2;          /* cities considered for an exchange */
    long int s_c1, s_c2;     /* successors of c1 and c2          */
    long int p_c1, p_c2;     /* predecessors of c1 and c2        */
    long int pos_c1, pos_c2; /* positions of cities c1, c2      */
    long int i, j, h, l;
    long int improvement_flag, improve_node;
    long int h1=0, h2=0, h3=0, h4=0, h5=0, help;
    long int radius;         /* radius of nn-search */
    long int gain = 0;
    long int *random_vector;
    long int two_move, node_move;

    long int *pos;           /* positions of cities in tour */
    long int *dlb;           /* vector containing don't look bits */

    pos = malloc(n * sizeof(long int));
    dlb = malloc(n * sizeof(long int));
    for ( i = 0 ; i < n ; i++ ) {
pos[tour[i]] = i;
dlb[i] = FALSE;
    }

    improvement_flag = TRUE;
    random_vector = generate_random_permutation( n );

    while ( improvement_flag ) {

improvement_flag = FALSE; two_move = FALSE; node_move = FALSE;

for ( l = 0 ; l < n; l++ ) {

        c1 = random_vector[l];
        DEBUG ( assert ( c1 < n && c1 >= 0); )
        if ( dlb_flag && dlb[c1] )

```

```

    continue;
    improve_node = FALSE;
    pos_c1 = pos[c1];
    s_c1 = tour[pos_c1+1];
    radius = instance.distance[c1][s_c1];

    /* First search for c1's nearest neighbours, use successor of c1 */
    for ( h = 0 ; h < nn_ls ; h++ ) {
c2 = instance.nn_list[c1][h]; /* exchange partner, determine its position */
if ( radius > instance.distance[c1][c2] ) {
    pos_c2 = pos[c2];
    s_c2 = tour[pos_c2+1];
    gain = - radius + instance.distance[c1][c2] +
instance.distance[s_c1][s_c2] - instance.distance[c2][s_c2];
    if ( gain < 0 ) {
h1 = c1; h2 = s_c1; h3 = c2; h4 = s_c2;
improve_node = TRUE; two_move = TRUE; node_move = FALSE;
goto exchange;
    }
    if ( pos_c2 > 0 )
p_c2 = tour[pos_c2-1];
    else
p_c2 = tour[n-1];
    gain = - radius + instance.distance[c1][c2] +
instance.distance[c2][s_c1]
+ instance.distance[p_c2][s_c2] - instance.distance[c2][s_c2]
- instance.distance[p_c2][c2];
    if ( c2 == s_c1 )
gain = 0;
    if ( p_c2 == s_c1 )
gain = 0;

    gain = 0;

    if ( gain < 0 ) {
h1 = c1; h2 = s_c1; h3 = c2; h4 = p_c2; h5 = s_c2;
improve_node = TRUE; node_move = TRUE; two_move = FALSE;
goto exchange;
    }
}
else
    break;
}
/* Second search for c1's nearest neighbours, use predecessor c1 */
if ( pos_c1 > 0 )
p_c1 = tour[pos_c1-1];
else
p_c1 = tour[n-1];
radius = instance.distance[p_c1][c1];
for ( h = 0 ; h < nn_ls ; h++ ) {
c2 = instance.nn_list[c1][h]; /* exchange partner, determine its position
*/
if ( radius > instance.distance[c1][c2] ) {
    pos_c2 = pos[c2];
    if ( pos_c2 > 0 )

```

```

p_c2 = tour[pos_c2-1];
    else
p_c2 = tour[n-1];
    if ( p_c2 == c1 )
continue;
    if ( p_c1 == c2 )
continue;
    gain = - radius + instance.distance[c1][c2] +
instance.distance[p_c1][p_c2] - instance.distance[p_c2][c2];
    if ( gain < 0 ) {
h1 = p_c1; h2 = c1; h3 = p_c2; h4 = c2;
improve_node = TRUE; two_move = TRUE; node_move = FALSE;
goto exchange;
    }
    s_c2 = tour[pos[c2]+1];
    gain = - radius + instance.distance[c2][c1] +
instance.distance[p_c1][c2]
+ instance.distance[p_c2][s_c2] - instance.distance[c2][s_c2]
- instance.distance[p_c2][c2];
    if ( p_c1 == c2 )
gain = 0;
    if ( p_c1 == s_c2 )
gain = 0;

    if ( gain < 0 ) {
h1 = p_c1; h2 = c1; h3 = c2; h4 = p_c2; h5 = s_c2;
improve_node = TRUE; node_move = TRUE; two_move = FALSE;
goto exchange;
    }
}
else
    break;
}
exchange:
    if (improve_node) {
    if ( two_move ) {
        improvement_flag = TRUE;
        dlb[h1] = FALSE; dlb[h2] = FALSE;
        dlb[h3] = FALSE; dlb[h4] = FALSE;
        /* Now perform move */
        if ( pos[h3] < pos[h1] ) {
            help = h1; h1 = h3; h3 = help;
            help = h2; h2 = h4; h4 = help;
        }
        if ( pos[h3] - pos[h2] < n / 2 + 1 ) {
            /* reverse inner part from pos[h2] to pos[h3] */
            i = pos[h2]; j = pos[h3];
            while ( i < j ) {
                c1 = tour[i];
                c2 = tour[j];
                tour[i] = c2;
                tour[j] = c1;
                pos[c1] = j;
                pos[c2] = i;
                i++; j--;
            }
        }
    }
}

```

```

}
}
else {
/* reverse outer part from pos[h4] to pos[h1] */
i = pos[h1]; j = pos[h4];
if ( j > i )
    help = n - ( j - i ) + 1;
else
    help = ( i - j ) + 1;
help = help / 2;
for ( h = 0 ; h < help ; h++ ) {
    c1 = tour[i];
    c2 = tour[j];
    tour[i] = c2;
    tour[j] = c1;
    pos[c1] = j;
    pos[c2] = i;
    i--; j++;
    if ( i < 0 )
        i = n-1;
    if ( j >= n )
        j = 0;
}
tour[n] = tour[0];
}
} else if ( node_move ) {
    improvement_flag = TRUE;
    dlb[h1] = FALSE; dlb[h2] = FALSE; dlb[h3] = FALSE;
    dlb[h4] = FALSE; dlb[h5] = FALSE;
    /* Now perform move */
    if ( pos[h3] < pos[h1] ) {
        help = pos[h1] - pos[h3];
        i = pos[h3];
        for ( h = 0 ; h < help ; h++ ) {
            c1 = tour[i+1];
            tour[i] = c1;
            pos[c1] = i;
            i++;
        }
        tour[i] = h3;
        pos[h3] = i;
        tour[n] = tour[0];
    } else {
        /* pos[h3] > pos[h1] */
        help = pos[h3] - pos[h1];
        /* if ( help < n / 2 + 1 ) { */
        i = pos[h3];
        for ( h = 0 ; h < help - 1 ; h++ ) {
            c1 = tour[i-1];
            tour[i] = c1;
            pos[c1] = i;
            i--;
        }
        tour[i] = h3;
        pos[h3] = i;
    }
}

```

```

    tour[n] = tour[0];
    /*      } */
  }
} else {
    fprintf(stderr, "this should never occur, 2-h-opt!!\n");
    exit(0);
}
two_move = FALSE; node_move = FALSE;
} else {
    dlb[c1] = TRUE;
}
}
}
free( random_vector );
free( dlb );
free( pos );
}

void three_opt_first( long int *tour )

/*
    FUNCTION:      3-opt the tour
    INPUT:         pointer to the tour that is to optimize
    OUTPUT:        none
    (SIDE)EFFECTS: tour is 3-opt
    COMMENT:       this is certainly not the best possible implementation of
a 3-opt
                                local search algorithm. In addition, it is very lengthy;
the main
    reason herefore is that awkward way of making an exchange, where
it is tried to copy only the shortest possible part of a tour.
Whoever improves the code regarding speed or solution quality, please
drop me the code at stuetzle no@spam informatik.tu-darmstadt.de
The neighbourhood is scanned in random order (this need
                                not be the best possible choice). Concerning the speed-ups
used
    here consult, for example, Chapter 8 of
    Holger H. Hoos and Thomas Stuetzle,
    Stochastic Local Search---Foundations and Applications,
    Morgan Kaufmann Publishers, 2004.
    or some of the papers online available from David S. Johnson.
*/
{
    /* In case a 2-opt move should be performed, we only need to store opt2_move
= TRUE,
    as h1, .. h4 are used in such a way that they store the indices of the
correct move */

    long int  c1, c2, c3;          /* cities considered for an exchange */
    long int  s_c1, s_c2, s_c3;   /* successors of these cities      */
    long int  p_c1, p_c2, p_c3;   /* predecessors of these cities    */
    long int  pos_c1, pos_c2, pos_c3; /* positions of cities c1, c2, c3 */
*/
    long int  i, j, h, g, l;

```

```

    long int    improvement_flag, help;
    long int    h1=0, h2=0, h3=0, h4=0, h5=0, h6=0; /* memorize cities involved
in a move */
    long int    diffs, diffp;
    long int    between = FALSE;
    long int    opt2_flag; /* = TRUE: perform 2-opt move, otherwise none or
3-opt move */
    long int    move_flag; /*
        move_flag = 0 --> no 3-opt move
        move_flag = 1 --> between_move (c3 between c1 and c2)
        move_flag = 2 --> not_between with successors of c2 and c3
        move_flag = 3 --> not_between with predecessors of c2 and c3
        move_flag = 4 --> cyclic move
    */
    long int    gain, move_value, radius, add1, add2;
    long int    decrease_breaks; /* Stores decrease by breaking two edges (a,b)
(c,d) */
    long int    val[3];
    long int    n1, n2, n3;
    long int    *pos; /* positions of cities in tour */
    long int    *dlb; /* vector containing don't look bits */
    long int    *h_tour; /* help vector for performing exchange move */
    long int    *hh_tour; /* help vector for performing exchange move */
    long int    *random_vector;

    pos = malloc(n * sizeof(long int));
    dlb = malloc(n * sizeof(long int));
    h_tour = malloc(n * sizeof(long int));
    hh_tour = malloc(n * sizeof(long int));

    for ( i = 0 ; i < n ; i++ ) {
pos[tour[i]] = i;
dlb[i] = FALSE;
    }
    improvement_flag = TRUE;
    random_vector = generate_random_permutation( n );

    while ( improvement_flag ) {
move_value = 0;
improvement_flag = FALSE;

    for ( l = 0 ; l < n ; l++ ) {

        c1 = random_vector[l];
        if ( dlb_flag && dlb[c1] )
            continue;
        opt2_flag = FALSE;

        move_flag = 0;
        pos_c1 = pos[c1];
        s_c1 = tour[pos_c1+1];
        if (pos_c1 > 0)
            p_c1 = tour[pos_c1-1];
        else
            p_c1 = tour[n-1];
    }

```

```

    h = 0;      /* Search for one of the h-nearest neighbours */
    while ( h < nn_ls ) {

c2   = instance.nn_list[c1][h]; /* second city, determine its position */
pos_c2 = pos[c2];
s_c2 = tour[pos_c2+1];
if (pos_c2 > 0)
    p_c2 = tour[pos_c2-1];
else
    p_c2 = tour[n-1];

diffs = 0; diffp = 0;

radius = instance.distance[c1][s_c1];
add1   = instance.distance[c1][c2];

/* Here a fixed radius neighbour search is performed */
if ( radius > add1 ) {
    decrease_breaks = - radius - instance.distance[c2][s_c2];
    diffs = decrease_breaks + add1 + instance.distance[s_c1][s_c2];
    diffp = - radius - instance.distance[c2][p_c2] +
    instance.distance[c1][p_c2] + instance.distance[s_c1][c2];
}
else
    break;
if ( p_c2 == c1 ) /* in case p_c2 == c1 no exchange is possible */
    diffp = 0;
if ( (diffs < move_value) || (diffp < move_value) ) {
    improvement_flag = TRUE;
    if (diffs <= diffp) {
        h1 = c1; h2 = s_c1; h3 = c2; h4 = s_c2;
        move_value = diffs;
        opt2_flag = TRUE; move_flag = 0;
        /*      goto exchange; */
    } else {
        h1 = c1; h2 = s_c1; h3 = p_c2; h4 = c2;
        move_value = diffp;
        opt2_flag = TRUE; move_flag = 0;
        /*      goto exchange; */
    }
}
/* Now perform the innermost search */
g = 0;
while (g < nn_ls) {

    c3   = instance.nn_list[s_c1][g];
    pos_c3 = pos[c3];
    s_c3 = tour[pos_c3+1];
    if (pos_c3 > 0)
    p_c3 = tour[pos_c3-1];
    else
    p_c3 = tour[n-1];

    if ( c3 == c1 ) {

```



```

g++;
continue;
}
else {
add2 = instance.distance[s_c1][c3];
/* Perform fixed radius neighbour search for innermost search */
if ( decrease_breaks + add1 < add2 ) {

    if ( pos_c2 > pos_c1 ) {
if ( pos_c3 <= pos_c2 && pos_c3 > pos_c1 )
    between = TRUE;
else
    between = FALSE;
}
else if ( pos_c2 < pos_c1 )
if ( pos_c3 > pos_c1 || pos_c3 < pos_c2 )
    between = TRUE;
else
    between = FALSE;
else {
printf(" Strange !!, pos_1 %ld == pos_2 %ld, \n",pos_c1,pos_c2);
}

    if ( between ) {
/* We have to add edges (c1,c2), (c3,s_c1), (p_c3,s_c2) to get
    valid tour; it's the only possibility */

gain = decrease_breaks - instance.distance[c3][p_c3] +
    add1 + add2 +
    instance.distance[p_c3][s_c2];

/* check for improvement by move */
if ( gain < move_value ) {
    improvement_flag = TRUE; /* g = neigh_ls + 1; */
    move_value = gain;
    opt2_flag = FALSE;
    move_flag = 1;
    /* store nodes involved in move */
    h1 = c1; h2 = s_c1; h3 = c2; h4 = s_c2; h5 = p_c3; h6 = c3;
    goto exchange;
}
}
else { /* not between(pos_c1,pos_c2,pos_c3) */

/* We have to add edges (c1,c2), (s_c1,c3), (s_c2,s_c3) */

gain = decrease_breaks - instance.distance[c3][s_c3] +
    add1 + add2 +
    instance.distance[s_c2][s_c3];

if ( pos_c2 == pos_c3 ) {
    gain = 20000;
}

/* check for improvement by move */

```

```

if ( gain < move_value ) {
    improvement_flag = TRUE; /* g = neigh_ls + 1; */
    move_value = gain;
    opt2_flag = FALSE;
    move_flag = 2;
    /* store nodes involved in move */
    h1 = c1; h2 = s_c1; h3 = c2; h4 = s_c2; h5 = c3; h6 = s_c3;
    goto exchange;
}

/* or add edges (c1,c2), (s_c1,c3), (p_c2,p_c3) */
gain = - radius - instance.distance[p_c2][c2]
      - instance.distance[p_c3][c3] +
      add1 + add2 +
      instance.distance[p_c2][p_c3];

if ( c3 == c2 || c2 == c1 || c1 == c3 || p_c2 == c1 ) {
    gain = 2000000;
}

if ( gain < move_value ) {
    improvement_flag = TRUE;
    move_value = gain;
    opt2_flag = FALSE;
    move_flag = 3;
    h1 = c1; h2 = s_c1; h3 = p_c2; h4 = c2; h5 = p_c3; h6 = c3;
    goto exchange;
}

/* Or perform the 3-opt move where no subtour inversion is necessary
   i.e. delete edges (c1,s_c1), (c2,p_c2), (c3,s_c3) and
   add edges (c1,c2), (c3,s_c1), (p_c2,s_c3) */

gain = - radius - instance.distance[p_c2][c2] -
      instance.distance[c3][s_c3]
      + add1 + add2 + instance.distance[p_c2][s_c3];

/* check for improvement */
if ( gain < move_value ) {
    improvement_flag = TRUE;
    move_value = gain;
    opt2_flag = FALSE;
    move_flag = 4;
    improvement_flag = TRUE;
    /* store nodes involved in move */
    h1 = c1; h2 = s_c1; h3 = p_c2; h4 = c2; h5 = c3; h6 = s_c3;
    goto exchange;
}
}
else
    g = nn_ls + 1;
}
g++;
}

```

```

h++;
}
if ( move_flag || opt2_flag ) {
exchange:
move_value = 0;

/* Now make the exchange */
if ( move_flag ) {
    dlb[h1] = FALSE; dlb[h2] = FALSE; dlb[h3] = FALSE;
    dlb[h4] = FALSE; dlb[h5] = FALSE; dlb[h6] = FALSE;
    pos_c1 = pos[h1]; pos_c2 = pos[h3]; pos_c3 = pos[h5];

    if ( move_flag == 4 ) {

if ( pos_c2 > pos_c1 )
    n1 = pos_c2 - pos_c1;
else
    n1 = n - (pos_c1 - pos_c2);
if ( pos_c3 > pos_c2 )
    n2 = pos_c3 - pos_c2;
else
    n2 = n - (pos_c2 - pos_c3);
if ( pos_c1 > pos_c3 )
    n3 = pos_c1 - pos_c3;
else
    n3 = n - (pos_c3 - pos_c1);

/* n1: length h2 - h3, n2: length h4 - h5, n3: length h6 - h1 */
val[0] = n1; val[1] = n2; val[2] = n3;
/* Now order the partial tours */
h = 0;
help = LONG_MIN;
for ( g = 0; g <= 2; g++) {
    if ( help < val[g] ) {
        help = val[g];
        h = g;
    }
}

/* order partial tours according length */
if ( h == 0 ) {
    /* copy part from pos[h4] to pos[h5]
    direkt kopiert: Teil von pos[h6] to pos[h1], it
    remains the part from pos[h2] to pos[h3] */
    j = pos[h4];
    h = pos[h5];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
        i++;
        j++;
    }
    if ( j >= n )
        j = 0;
    h_tour[i] = tour[j];
}
}
}

```

```

nl++;
}

/* First copy partial tour 3 in new position */
j = pos[h4];
i = pos[h6];
tour[j] = tour[i];
pos[tour[i]] = j;
while ( i != pos_c1) {
i++;
if ( i >= n )
    i = 0;
j++;
if ( j >= n )
    j = 0;
tour[j] = tour[i];
pos[tour[i]] = j;
}

/* Now copy stored part from h_tour */
j++;
if ( j >= n )
j = 0;
for ( i = 0; i < nl ; i++ ) {
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
j++;
if ( j >= n )
    j = 0;
}
tour[n] = tour[0];
}
else if ( h == 1 ) {

/* copy part from pos[h6] to pos[h1]
direkt kopiert: Teil von pos[h2] to pos[h3], it
remains the part from pos[h4] to pos[h5] */
j = pos[h6];
h = pos[h1];
i = 0;
h_tour[i] = tour[j];
nl = 1;
while ( j != h) {
i++;
j++;
if ( j >= n )
    j = 0;
h_tour[i] = tour[j];
nl++;
}

/* First copy partial tour 3 in new position */
j = pos[h6];
i = pos[h2];
tour[j] = tour[i];

```

```

    pos[tour[i]] = j;
    while ( i != pos_c2) {
i++;
if ( i >= n )
    i = 0;
j++;
if ( j >= n )
    j = 0;
tour[j] = tour[i];
pos[tour[i]] = j;
    }

    /* Now copy stored part from h_tour */
    j++;
    if ( j >= n )
j = 0;
    for ( i = 0; i < n1 ; i++ ) {
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
j++;
if ( j >= n )
    j = 0;
    }
    tour[n] = tour[0];
}
else if ( h == 2 ) {
    /* copy part from pos[h2] to pos[h3]
    direkt kopiert: Teil von pos[h4] to pos[h5], it
    remains the part from pos[h6] to pos[h1] */
    j = pos[h2];
    h = pos[h3];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
i++;
j++;
if ( j >= n )
    j = 0;
h_tour[i] = tour[j];
n1++;
    }

    /* First copy partial tour 3 in new position */
    j = pos[h2];
    i = pos[h4];
    tour[j] = tour[i];
    pos[tour[i]] = j;
    while ( i != pos_c3) {
i++;
if ( i >= n )
    i = 0;
j++;
if ( j >= n )
    j = 0;

```

```

tour[j] = tour[i];
pos[tour[i]] = j;
}

/* Now copy stored part from h_tour */
j++;
if ( j >= n )
j = 0;
for ( i = 0; i < n1 ; i++ ) {
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
j++;
if ( j >= n )
j = 0;
}
tour[n] = tour[0];
}
}
else if ( move_flag == 1 ) {

if ( pos_c3 < pos_c2 )
n1 = pos_c2 - pos_c3;
else
n1 = n - (pos_c3 - pos_c2);
if ( pos_c3 > pos_c1 )
n2 = pos_c3 - pos_c1 + 1;
else
n2 = n - (pos_c1 - pos_c3 + 1);
if ( pos_c2 > pos_c1 )
n3 = n - (pos_c2 - pos_c1 + 1);
else
n3 = pos_c1 - pos_c2 + 1;

/* n1: length h6 - h3, n2: length h5 - h2, n2: length h1 - h3 */
val[0] = n1; val[1] = n2; val[2] = n3;
/* Now order the partial tours */
h = 0;
help = LONG_MIN;
for ( g = 0; g <= 2; g++ ) {
if ( help < val[g] ) {
help = val[g];
h = g;
}
}
/* order partial tours according length */

if ( h == 0 ) {

/* copy part from pos[h5] to pos[h2]
(inverted) and from pos[h4] to pos[h1] (inverted)
it remains the part from pos[h6] to pos[h3] */
j = pos[h5];
h = pos[h2];
i = 0;
h_tour[i] = tour[j];

```

```

    n1 = 1;
    while ( j != h ) {
i++;
j--;
if ( j < 0 )
    j = n-1;
h_tour[i] = tour[j];
n1++;
    }

    j = pos[h1];
    h = pos[h4];
    i = 0;
    hh_tour[i] = tour[j];
    n2 = 1;
    while ( j != h ) {
i++;
j--;
if ( j < 0 )
    j = n-1;
hh_tour[i] = tour[j];
n2++;
    }

    j = pos[h4];
    for ( i = 0; i < n2 ; i++ ) {
tour[j] = hh_tour[i];
pos[hh_tour[i]] = j;
j++;
if ( j >= n )
    j = 0;
    }

    /* Now copy stored part from h_tour */
    for ( i = 0; i < n1 ; i++ ) {
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
j++;
if ( j >= n )
    j = 0;
    }
    tour[n] = tour[0];
}
else if ( h == 1 ) {

    /* copy part from h3 to h6 (wird inverted) erstellen : */
    j = pos[h3];
    h = pos[h6];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
i++;
j--;
if ( j < 0 )

```

```

        j = n-1;
h_tour[i] = tour[j];
nl++;
    }

    j = pos[h6];
    i = pos[h4];

    tour[j] = tour[i];
    pos[tour[i]] = j;
    while ( i != pos_c1) {
i++;
j++;
if ( j >= n)
    j = 0;
if ( i >= n)
    i = 0;
tour[j] = tour[i];
pos[tour[i]] = j;
    }

    /* Now copy stored part from h_tour */
    j++;
    if ( j >= n )
j = 0;
    i = 0;
    tour[j] = h_tour[i];
    pos[h_tour[i]] = j;
    while ( j != pos_c1 ) {
j++;
if ( j >= n )
    j = 0;
i++;
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
    }
    tour[n] = tour[0];
}

else if ( h == 2 ) {

    /* copy part from pos[h2] to pos[h5] and
    from pos[h3] to pos[h6] (inverted), it
    remains the part from pos[h4] to pos[h1] */
    j = pos[h2];
    h = pos[h5];
    i = 0;
    h_tour[i] = tour[j];
    nl = 1;
    while ( j != h ) {
i++;
j++;
if ( j >= n )
    j = 0;
h_tour[i] = tour[j];

```



```

n1++;
}
j = pos_c2;
h = pos[h6];
i = 0;
hh_tour[i] = tour[j];
n2 = 1;
while ( j != h ) {
i++;
j--;
if ( j < 0 )
j = n-1;
hh_tour[i] = tour[j];
n2++;
}

j = pos[h2];
for ( i = 0; i < n2 ; i++ ) {
tour[j] = hh_tour[i];
pos[hh_tour[i]] = j;
j++;
if ( j >= n )
j = 0;
}

/* Now copy stored part from h_tour */
for ( i = 0; i < n1 ; i++ ) {
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
j++;
if ( j >= n )
j = 0;
}
tour[n] = tour[0];
}
}
else if ( move_flag == 2 ) {

if ( pos_c3 < pos_c1 )
n1 = pos_c1 - pos_c3;
else
n1 = n - (pos_c3 - pos_c1);
if ( pos_c3 > pos_c2 )
n2 = pos_c3 - pos_c2;
else
n2 = n - (pos_c2 - pos_c3);
if ( pos_c2 > pos_c1 )
n3 = pos_c2 - pos_c1;
else
n3 = n - (pos_c1 - pos_c2);

val[0] = n1; val[1] = n2; val[2] = n3;
/* Determine which is the longest part */
h = 0;
help = LONG_MIN;

```

```

for ( g = 0; g <= 2; g++) {
    if ( help < val[g] ) {
        help = val[g];
        h = g;
    }
}
/* order partial tours according length */

if ( h == 0 ) {

    /* copy part from pos[h3] to pos[h2]
       (inverted) and from pos[h5] to pos[h4], it
       remains the part from pos[h6] to pos[h1] */
    j = pos[h3];
    h = pos[h2];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
        i++;
        j--;
        if ( j < 0 )
            j = n-1;
        h_tour[i] = tour[j];
        n1++;
    }

    j = pos[h5];
    h = pos[h4];
    i = 0;
    hh_tour[i] = tour[j];
    n2 = 1;
    while ( j != h ) {
        i++;
        j--;
        if ( j < 0 )
            j = n-1;
        hh_tour[i] = tour[j];
        n2++;
    }

    j = pos[h2];
    for ( i = 0; i < n1 ; i++ ) {
        tour[j] = h_tour[i];
        pos[h_tour[i]] = j;
        j++;
        if ( j >= n )
            j = 0;
    }

    for ( i = 0; i < n2 ; i++ ) {
        tour[j] = hh_tour[i];
        pos[hh_tour[i]] = j;
        j++;
        if ( j >= n )

```

```

        j = 0;
    }
    tour[n] = tour[0];
    /*      getchar(); */
}
else if ( h == 1 ) {

    /* copy part from pos[h2] to pos[h3] and
       from pos[h1] to pos[h6] (inverted), it
       remains the part from pos[h4] to pos[h5] */
    j = pos[h2];
    h = pos[h3];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
    i++;
    j++;
    if ( j >= n )
        j = 0;
    h_tour[i] = tour[j];
    n1++;
    }

    j = pos[h1];
    h = pos[h6];
    i = 0;
    hh_tour[i] = tour[j];
    n2 = 1;
    while ( j != h ) {
    i++;
    j--;
    if ( j < 0 )
        j = n-1;
    hh_tour[i] = tour[j];
    n2++;
    }
    j = pos[h6];
    for ( i = 0; i < n1 ; i++ ) {
    tour[j] = h_tour[i];
    pos[h_tour[i]] = j;
    j++;
    if ( j >= n )
        j = 0;
    }
    for ( i = 0; i < n2 ; i++ ) {
    tour[j] = hh_tour[i];
    pos[hh_tour[i]] = j;
    j++;
    if ( j >= n )
        j = 0;
    }
    tour[n] = tour[0];
}

```

```

else if ( h == 2 ) {

    /* copy part from pos[h1] to pos[h6]
       (inverted) and from pos[h4] to pos[h5],
       it remains the part from pos[h2] to
       pos[h3] */
    j = pos[h1];
    h = pos[h6];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
        i++;
        j--;
        if ( j < 0 )
            j = n-1;
        h_tour[i] = tour[j];
        n1++;
    }

    j = pos[h4];
    h = pos[h5];
    i = 0;
    hh_tour[i] = tour[j];
    n2 = 1;
    while ( j != h ) {
        i++;
        j++;
        if ( j >= n )
            j = 0;
        hh_tour[i] = tour[j];
        n2++;
    }

    j = pos[h4];
    /* Now copy stored part from h_tour */
    for ( i = 0; i < n1 ; i++ ) {
        tour[j] = h_tour[i];
        pos[h_tour[i]] = j;
        j++;
        if ( j >= n )
            j = 0;
    }

    /* Now copy stored part from hh_tour */
    for ( i = 0; i < n2 ; i++ ) {
        tour[j] = hh_tour[i];
        pos[hh_tour[i]] = j;
        j++;
        if ( j >= n )
            j = 0;
    }
    tour[n] = tour[0];
}
}

```

```

    else if ( move_flag == 3 ) {

if ( pos_c3 < pos_c1 )
    n1 = pos_c1 - pos_c3;
else
    n1 = n - (pos_c3 - pos_c1);
if ( pos_c3 > pos_c2 )
    n2 = pos_c3 - pos_c2;
else
    n2 = n - (pos_c2 - pos_c3);
if ( pos_c2 > pos_c1 )
    n3 = pos_c2 - pos_c1;
else
    n3 = n - (pos_c1 - pos_c2);
/* n1: length h6 - h1, n2: length h4 - h5, n2: length h2 - h3 */

val[0] = n1; val[1] = n2; val[2] = n3;
/* Determine which is the longest part */
h = 0;
help = LONG_MIN;
for ( g = 0; g <= 2; g++) {
    if ( help < val[g] ) {
        help = val[g];
        h = g;
    }
}
/* order partial tours according length */

if ( h == 0 ) {

    /* copy part from pos[h2] to pos[h3]
       (inverted) and from pos[h4] to pos[h5]
       it remains the part from pos[h6] to pos[h1] */
    j = pos[h3];
    h = pos[h2];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
        i++;
        j--;
        if ( j < 0 )
            j = n-1;
        h_tour[i] = tour[j];
        n1++;
    }

    j = pos[h2];
    h = pos[h5];
    i = pos[h4];
    tour[j] = h4;
    pos[h4] = j;
    while ( i != h ) {
        i++;
        if ( i >= n )

```

```

        i = 0;
    j++;
    if ( j >= n )
        j = 0;
    tour[j] = tour[i];
    pos[tour[i]] = j;
    }
    j++;
    if ( j >= n )
    j = 0;
    for ( i = 0; i < n1 ; i++ ) {
    tour[j] = h_tour[i];
    pos[h_tour[i]] = j;
    j++;
    if ( j >= n )
        j = 0;
    }
    tour[n] = tour[0];
}
else if ( h == 1 ) {

    /* copy part from pos[h3] to pos[h2]
       (inverted) and from pos[h6] to pos[h1],
       it remains the part from pos[h4] to pos[h5] */
    j = pos[h3];
    h = pos[h2];
    i = 0;
    h_tour[i] = tour[j];
    n1 = 1;
    while ( j != h ) {
    i++;
    j--;
    if ( j < 0 )
        j = n-1;
    h_tour[i] = tour[j];
    n1++;
    }

    j = pos[h6];
    h = pos[h1];
    i = 0;
    hh_tour[i] = tour[j];
    n2 = 1;
    while ( j != h ) {
    i++;
    j++;
    if ( j >= n )
        j = 0;
    hh_tour[i] = tour[j];
    n2++;
    }

    j = pos[h6];
    for ( i = 0; i < n1 ; i++ ) {
    tour[j] = h_tour[i];

```

```

pos[h_tour[i]] = j;
j++;
if ( j >= n )
    j = 0;
}

for ( i = 0 ; i < n2 ; i++ ) {
tour[j] = hh_tour[i];
pos[hh_tour[i]] = j;
j++;
if ( j >= n )
    j = 0;
}
tour[n] = tour[0];
}

else if ( h == 2 ) {

/* copy part from pos[h4] to pos[h5]
   (inverted) and from pos[h6] to pos[h1] (inverted)
   it remains the part from pos[h2] to pos[h3] */
j = pos[h5];
h = pos[h4];
i = 0;
h_tour[i] = tour[j];
n1 = 1;
while ( j != h ) {
i++;
j--;
if ( j < 0 )
    j = n-1;
h_tour[i] = tour[j];
n1++;
}

j = pos[h1];
h = pos[h6];
i = 0;
hh_tour[i] = tour[j];
n2 = 1;
while ( j != h ) {
i++;
j--;
if ( j < 0 )
    j = n-1;
hh_tour[i] = tour[j];
n2++;
}

j = pos[h4];
/* Now copy stored part from h_tour */
for ( i = 0 ; i < n1 ; i++ ) {
tour[j] = h_tour[i];
pos[h_tour[i]] = j;
j++;
}
}

```

```

    if ( j >= n )
        j = 0;
    }
    /* Now copy stored part from h_tour */
    for ( i = 0; i < n2 ; i++ ) {
        tour[j] = hh_tour[i];
        pos[hh_tour[i]] = j;
        j++;
        if ( j >= n )
            j = 0;
    }
    tour[n] = tour[0];
}
}
else {
printf(" Some very strange error must have occurred !!!\n\n");
exit(0);
}
}
if (opt2_flag) {

    /* Now perform move */
    dlb[h1] = FALSE; dlb[h2] = FALSE;
    dlb[h3] = FALSE; dlb[h4] = FALSE;
    if ( pos[h3] < pos[h1] ) {
        help = h1; h1 = h3; h3 = help;
        help = h2; h2 = h4; h4 = help;
    }
    if ( pos[h3]-pos[h2] < n / 2 + 1 ) {
        /* reverse inner part from pos[h2] to pos[h3] */
        i = pos[h2]; j = pos[h3];
        while ( i < j ) {
            c1 = tour[i];
            c2 = tour[j];
            tour[i] = c2;
            tour[j] = c1;
            pos[c1] = j;
            pos[c2] = i;
            i++; j--;
        }
    }
    else {
        /* reverse outer part from pos[h4] to pos[h1] */
        i = pos[h1]; j = pos[h4];
        if ( j > i )
            help = n - (j - i) + 1;
        else
            help = (i - j) + 1;
        help = help / 2;
        for ( h = 0 ; h < help ; h++ ) {
            c1 = tour[i];
            c2 = tour[j];
            tour[i] = c2;
            tour[j] = c1;
            pos[c1] = j;

```



```

        pos[c2] = i;
        i--; j++;
        if ( i < 0 )
            i = n - 1;
        if ( j >= n )
            j = 0;
    }
    tour[n] = tour[0];
}
}
}
else {
    dlb[c1] = TRUE;
}
}
}
free( random_vector );
free( h_tour );
free( hh_tour );
free( pos );
free( dlb );
}
}

```

parse.h

```
/* This file has been generated with opag 0.6.4. */

#ifndef HDR_PARSE
#define HDR_PARSE 1

struct options {

/* Set to 1 if option --tries (-r) has been specified. */
unsigned int opt_tries : 1;

/* Set to 1 if option --tours (-s) has been specified. */
unsigned int opt_tours : 1;

/* Set to 1 if option --time (-t) has been specified. */
unsigned int opt_time : 1;

/* Set to 1 if option --tsplibfile (-i) has been specified. */
unsigned int opt_tsplibfile : 1;

/* Set to 1 if option --optimum (-o) has been specified. */
unsigned int opt_optimum : 1;

/* Set to 1 if option --ants (-m) has been specified. */
unsigned int opt_ants : 1;

/* Set to 1 if option --nnants (-g) has been specified. */
unsigned int opt_nnants : 1;

/* Set to 1 if option --alpha (-a) has been specified. */
unsigned int opt_alpha : 1;

/* Set to 1 if option --beta (-b) has been specified. */
unsigned int opt_beta : 1;

/* Set to 1 if option --rho (-e) has been specified. */
unsigned int opt_rho : 1;

/* Set to 1 if option --q0 (-q) has been specified. */
unsigned int opt_q0 : 1;

/* Set to 1 if option --elitistants (-c) has been specified. */
unsigned int opt_elitistants : 1;

/* Set to 1 if option --rasranks (-f) has been specified. */
unsigned int opt_rasranks : 1;

/* Set to 1 if option --nnls (-k) has been specified. */
unsigned int opt_nnls : 1;

/* Set to 1 if option --localsearch (-l) has been specified. */
unsigned int opt_localsearch : 1;

/* Set to 1 if option --dlb (-d) has been specified. */
```

```

unsigned int opt_dlb : 1;

/* Set to 1 if option --as (-u) has been specified. */
unsigned int opt_as : 1;

/* Set to 1 if option --eas (-v) has been specified. */
unsigned int opt_eas : 1;

/* Set to 1 if option --ras (-w) has been specified. */
unsigned int opt_ras : 1;

/* Set to 1 if option --mmas (-x) has been specified. */
unsigned int opt_mmas : 1;

/* Set to 1 if option --bwas (-y) has been specified. */
unsigned int opt_bwas : 1;

/* Set to 1 if option --acs (-z) has been specified. */
unsigned int opt_acs : 1;

/* Set to 1 if option --help (-h) has been specified. */
unsigned int opt_help : 1;

/* Argument to option --tries (-r). */
const char *arg_tries;

/* Argument to option --tours (-s). */
const char *arg_tours;

/* Argument to option --time (-t). */
const char *arg_time;

/* Argument to option --tsplibfile (-i). */
const char *arg_tsplibfile;

/* Argument to option --optimum (-o). */
const char *arg_optimum;

/* Argument to option --ants (-m). */
const char *arg_ants;

/* Argument to option --nnants (-g). */
const char *arg_nnants;

/* Argument to option --alpha (-a). */
const char *arg_alpha;

/* Argument to option --beta (-b). */
const char *arg_beta;

/* Argument to option --rho (-e). */
const char *arg_rho;

/* Argument to option --q0 (-q). */
const char *arg_q0;

```

```

/* Argument to option --elitistants (-c). */
const char *arg_elitistants;

/* Argument to option --rasranks (-f). */
const char *arg_rasranks;

/* Argument to option --nnls (-k). */
const char *arg_nnls;

/* Argument to option --localsearch (-l). */
const char *arg_localsearch;

/* Argument to option --dlb (-d). */
const char *arg_dlb;

};

/* Parse command line options. Return index of first non-option argument,
   or -1 if an error is encountered. */
extern int parse_options (struct options *options, const char *program_name, int
argc, char **argv);

extern void check_out_of_range ( double value, double MIN, double MAX, char
*optionName );

extern int parse_commandline (int argc, char *argv []);

#endif

```

parse.c

```
/* This file has been generated with opag 0.6.4. */
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <limits.h>
#include <stdlib.h>

#include "InOut.h"
#include "utilities.h"
#include "ants.h"
#include "ls.h"

#ifndef STR_ERR_UNKNOWN_LONG_OPT
# define STR_ERR_UNKNOWN_LONG_OPT "%s: unrecognized option '--%s'\n"
#endif

#ifndef STR_ERR_LONG_OPT_AMBIGUOUS
# define STR_ERR_LONG_OPT_AMBIGUOUS "%s: option '--%s' is ambiguous\n"
#endif

#ifndef STR_ERR_MISSING_ARG_LONG
# define STR_ERR_MISSING_ARG_LONG "%s: option '--%s' requires an argument\n"
#endif

#ifndef STR_ERR_UNEXPEC_ARG_LONG
# define STR_ERR_UNEXPEC_ARG_LONG "%s: option '--%s' doesn't allow an
argument\n"
#endif

#ifndef STR_ERR_UNKNOWN_SHORT_OPT
# define STR_ERR_UNKNOWN_SHORT_OPT "%s: unrecognized option '-%c'\n"
#endif

#ifndef STR_ERR_MISSING_ARG_SHORT
# define STR_ERR_MISSING_ARG_SHORT "%s: option '-%c' requires an argument\n"
#endif

#define STR_HELP_TRIES \
    " -r, --tries          # number of independent trials\n"

#define STR_HELP_TOURS \
    " -s, --tours          # number of steps in each trial\n"

#define STR_HELP_TIME \
    " -t, --time           # maximum time for each trial\n"

#define STR_HELP_TSPLIBFILE \
    " -i, --tsplibfile     f inputfile (TSPLIB format necessary)\n"

#define STR_HELP_OPTIMUM \
    " -o, --optimum        # stop if tour better or equal optimum is found\n"

#define STR_HELP_ANTS \
```

```

" -m, --ants          # number of ants\n"
#define STR_HELP_NNANTS \
" -g, --nnants        # nearest neighbours in tour construction\n"
#define STR_HELP_ALPHA \
" -a, --alpha         # alpha (influence of pheromone trails)\n"
#define STR_HELP_BETA \
" -b, --beta          # beta (influence of heuristic information)\n"
#define STR_HELP_RHO \
" -e, --rho           # rho: pheromone trail evaporation\n"
#define STR_HELP_Q0 \
" -q, --q0            # q0: prob. of best choice in tour construction\n"
#define STR_HELP_ELITISTANTS \
" -c, --elitistants  # number of elitist ants\n"
#define STR_HELP_RASRANKS \
" -f, --rasranks     # number of ranks in rank-based Ant System\n"
#define STR_HELP_NNLS \
" -k, --nnls         # No. of nearest neighbors for local search\n"
#define STR_HELP_LOCALSEARCH \
" -l, --localsearch  0: no local search  1: 2-opt  2: 2.5-opt  3:
3-opt\n"
#define STR_HELP_DLB \
" -d, --dlb          1 use don't look bits in local search\n"
#define STR_HELP_AS \
" -u, --as           apply basic Ant System\n"
#define STR_HELP_EAS \
" -v, --eas         -apply elitist Ant System\n"
#define STR_HELP_RAS \
" -w, --ras         apply rank-based version of Ant System\n"
#define STR_HELP_MMAS \
" -x, --mmas        apply MAX-MIN ant system\n"
#define STR_HELP_BWAS \
" -y, --bwas        apply best-worst ant system\n"
#define STR_HELP_ACS \
" -z, --acs         apply ant colony system\n"
#define STR_HELP_HELP \
" -h, --help        display this help text and exit\n"
#define STR_HELP \

```

```

STR_HELP_TRIES \
STR_HELP_TOURS \
STR_HELP_TIME \
STR_HELP_TSPLIBFILE \
STR_HELP_OPTIMUM \
STR_HELP_ANTS \
STR_HELP_NNANTS \
STR_HELP_ALPHA \
STR_HELP_BETA \
STR_HELP_RHO \
STR_HELP_Q0 \
STR_HELP_ELITISTANTS \
STR_HELP_RASRANKS \
STR_HELP>NNLS \
STR_HELP_LOCALSEARCH \
STR_HELP_DLB \
STR_HELP_AS \
STR_HELP_EAS \
STR_HELP_RAS \
STR_HELP_MMAS \
STR_HELP_BWAS \
STR_HELP_ACS \
STR_HELP_HELP

struct options {

/* Set to 1 if option --tries (-r) has been specified. */
unsigned int opt_tries : 1;

/* Set to 1 if option --tours (-s) has been specified. */
unsigned int opt_tours : 1;

/* Set to 1 if option --time (-t) has been specified. */
unsigned int opt_time : 1;

/* Set to 1 if option --tsplibfile (-i) has been specified. */
unsigned int opt_tsplibfile : 1;

/* Set to 1 if option --optimum (-o) has been specified. */
unsigned int opt_optimum : 1;

/* Set to 1 if option --ants (-m) has been specified. */
unsigned int opt_ants : 1;

/* Set to 1 if option --nnants (-g) has been specified. */
unsigned int opt_nnants : 1;

/* Set to 1 if option --alpha (-a) has been specified. */
unsigned int opt_alpha : 1;

/* Set to 1 if option --beta (-b) has been specified. */
unsigned int opt_beta : 1;

/* Set to 1 if option --rho (-e) has been specified. */
unsigned int opt_rho : 1;

```

```

/* Set to 1 if option --q0 (-q) has been specified. */
unsigned int opt_q0 : 1;

/* Set to 1 if option --elitistants (-c) has been specified. */
unsigned int opt_elitistants : 1;

/* Set to 1 if option --rasranks (-f) has been specified. */
unsigned int opt_rasranks : 1;

/* Set to 1 if option --nnls (-k) has been specified. */
unsigned int opt_nnls : 1;

/* Set to 1 if option --localsearch (-l) has been specified. */
unsigned int opt_localsearch : 1;

/* Set to 1 if option --dlb (-d) has been specified. */
unsigned int opt_dlb : 1;

/* Set to 1 if option --as (-u) has been specified. */
unsigned int opt_as : 1;

/* Set to 1 if option --eas (-v) has been specified. */
unsigned int opt_eas : 1;

/* Set to 1 if option --ras (-w) has been specified. */
unsigned int opt_ras : 1;

/* Set to 1 if option --mmas (-x) has been specified. */
unsigned int opt_mmas : 1;

/* Set to 1 if option --bwas (-y) has been specified. */
unsigned int opt_bwas : 1;

/* Set to 1 if option --acs (-z) has been specified. */
unsigned int opt_acs : 1;

/* Set to 1 if option --help (-h) has been specified. */
unsigned int opt_help : 1;

/* Argument to option --tries (-r). */
const char *arg_tries;

/* Argument to option --tours (-s). */
const char *arg_tours;

/* Argument to option --time (-t). */
const char *arg_time;

/* Argument to option --tsplibfile (-i). */
const char *arg_tsplibfile;

/* Argument to option --optimum (-o). */
const char *arg_optimum;

```



```

/* Argument to option --ants (-m). */
const char *arg_ants;

/* Argument to option --nnants (-g). */
const char *arg_nnants;

/* Argument to option --alpha (-a). */
const char *arg_alpha;

/* Argument to option --beta (-b). */
const char *arg_beta;

/* Argument to option --rho (-e). */
const char *arg_rho;

/* Argument to option --q0 (-q). */
const char *arg_q0;

/* Argument to option --elitistants (-c). */
const char *arg_elitistants;

/* Argument to option --rasranks (-f). */
const char *arg_rasranks;

/* Argument to option --nnls (-k). */
const char *arg_nnls;

/* Argument to option --localsearch (-l). */
const char *arg_localsearch;

/* Argument to option --dlb (-d). */
const char *arg_dlb;
};

int PC_myProc;

/* Parse command line options. Return index of first non-option argument,
or -1 if an error is encountered. */
int parse_options (struct options *const options, const char *const
program_name, const int argc, char **const argv)
{
    static const char *const optstr_tries = "tries";
    static const char *const optstr_tours = "tours";
    static const char *const optstr_time = "time";
    static const char *const optstr_tsplibfile = "tsplibfile";
    static const char *const optstr_optimum = "optimum";
    static const char *const optstr_ants = "ants";
    static const char *const optstr_nnants = "nnants";
    static const char *const optstr_alpha = "alpha";
    static const char *const optstr_beta = "beta";
    static const char *const optstr_rho = "rho";
    static const char *const optstr_q0 = "q0";
    static const char *const optstr_elitistants = "elitistants";
    static const char *const optstr_rasranks = "rasranks";

```

```

static const char *const optstr_nnls = "nnls";
static const char *const optstr_localsearch = "localsearch";
static const char *const optstr_dlb = "dlb";
static const char *const optstr_as = "as";
static const char *const optstr_eas = "eas";
static const char *const optstr_ras = "ras";
static const char *const optstr_mmas = "mmas";
static const char *const optstr_bwas = "bwas";
static const char *const optstr_acs = "acs";
static const char *const optstr_help = "help";
int i = 0;
options->opt_tries = 0;
options->opt_tours = 0;
options->opt_time = 0;
options->opt_tsplibfile = 0;
options->opt_optimum = 0;
options->opt_ants = 0;
options->opt_nnants = 0;
options->opt_alpha = 0;
options->opt_beta = 0;
options->opt_rho = 0;
options->opt_q0 = 0;
options->opt_elitistants = 0;
options->opt_rasranks = 0;
options->opt_nnls = 0;
options->opt_localsearch = 0;
options->opt_dlb = 0;
options->opt_as = 0;
options->opt_eas = 0;
options->opt_ras = 0;
options->opt_mmas = 0;
options->opt_bwas = 0;
options->opt_acs = 0;
options->opt_help = 0;
options->arg_tries = 0;
options->arg_tours = 0;
options->arg_time = 0;
options->arg_tsplibfile = 0;
options->arg_optimum = 0;
options->arg_ants = 0;
options->arg_nnants = 0;
options->arg_alpha = 0;
options->arg_beta = 0;
options->arg_rho = 0;
options->arg_q0 = 0;
options->arg_elitistants = 0;
options->arg_rasranks = 0;
options->arg_nnls = 0;
options->arg_localsearch = 0;
options->arg_dlb = 0;
while (++i < argc)
{
    const char *option = argv [i];
    if (*option != '-')
        return i;
}

```

```

else if (*++option == '\0')
    return i;
else if (*option == '-')
{
    const char *argument;
    size_t option_len;
    ++option;
    if ((argument = strchr (option, '=')) == option)
        goto error_unknown_long_opt;
    else if (argument == 0)
        option_len = strlen (option);
    else
        option_len = argument++ - option;
    switch (*option)
    {
    case '\0':
        return i + 1;
    case 'a':
        if (strncmp (option + 1, optstr__acs + 1, option_len - 1) == 0)
        {
            if (option_len <= 1)
                goto error_long_opt_ambiguous;
            if (argument != 0)
            {
                option = optstr__acs;
                goto error_unexpec_arg_long;
            }
            options->opt_acs = 1;
            break;
        }
    else if (strncmp (option + 1, optstr__alpha + 1, option_len - 1) == 0)
    {
        if (option_len <= 1)
            goto error_long_opt_ambiguous;
        if (argument != 0)
            options->arg_alpha = argument;
        else if (++i < argc)
            options->arg_alpha = argv [i];
        else
        {
            option = optstr__alpha;
            goto error_missing_arg_long;
        }
        options->opt_alpha = 1;
        break;
    }
    else if (strncmp (option + 1, optstr__ants + 1, option_len - 1) == 0)
    {
        if (option_len <= 1)
            goto error_long_opt_ambiguous;
        if (argument != 0)
            options->arg_ants = argument;
        else if (++i < argc)
            options->arg_ants = argv [i];
        else

```

```

    {
        option = optstr__ants;
        goto error_missing_arg_long;
    }
    options->opt_ants = 1;
    break;
}
else if (strncmp (option + 1, optstr__as + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
    {
        option = optstr__as;
        goto error_unexpec_arg_long;
    }
    options->opt_as = 1;
    break;
}
goto error_unknown_long_opt;
case 'b':
if (strncmp (option + 1, optstr__beta + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_beta = argument;
    else if (++i < argc)
        options->arg_beta = argv [i];
    else
    {
        option = optstr__beta;
        goto error_missing_arg_long;
    }
    options->opt_beta = 1;
    break;
}
else if (strncmp (option + 1, optstr__bwas + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
    {
        option = optstr__bwas;
        goto error_unexpec_arg_long;
    }
    options->opt_bwas = 1;
    break;
}
goto error_unknown_long_opt;
case 'd':
if (strncmp (option + 1, optstr__dlb + 1, option_len - 1) == 0)
{
    if (argument != 0)
        options->arg_dlb = argument;
}

```

```

else if (++i < argc)
    options->arg_dlb = argv [i];
else
    {
    option = optstr__dlb;
    goto error_missing_arg_long;
    }
options->opt_dlb = 1;
break;
}
goto error_unknown_long_opt;
case 'e':
if (strncmp (option + 1, optstr__eas + 1, option_len - 1) == 0)
    {
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        {
        option = optstr__eas;
        goto error_unexpec_arg_long;
        }
    options->opt_eas = 1;
    break;
    }
else if (strncmp (option + 1, optstr__elitistants + 1, option_len - 1)
== 0)
    {
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_elitistants = argument;
    else if (++i < argc)
        options->arg_elitistants = argv [i];
    else
        {
        option = optstr__elitistants;
        goto error_missing_arg_long;
        }
    options->opt_elitistants = 1;
    break;
    }
goto error_unknown_long_opt;
case 'h':
if (strncmp (option + 1, optstr__help + 1, option_len - 1) == 0)
    {
    if (argument != 0)
        {
        option = optstr__help;
        goto error_unexpec_arg_long;
        }
    options->opt_help = 1;
    return i + 1;
    }
goto error_unknown_long_opt;
case 'l':

```

```

if (strncmp (option + 1, optstr__localsearch + 1, option_len - 1) == 0)
{
    if (argument != 0)
        options->arg_localsearch = argument;
    else if (++i < argc)
        options->arg_localsearch = argv [i];
    else
    {
        option = optstr__localsearch;
        goto error_missing_arg_long;
    }
    options->opt_localsearch = 1;
    break;
}
goto error_unknown_long_opt;
case 'm':
if (strncmp (option + 1, optstr__mmas + 1, option_len - 1) == 0)
{
    if (argument != 0)
    {
        option = optstr__mmas;
        goto error_unexpect_arg_long;
    }
    options->opt_mmas = 1;
    break;
}
goto error_unknown_long_opt;
case 'n':
if (strncmp (option + 1, optstr__nnants + 1, option_len - 1) == 0)
{
    if (option_len <= 2)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_nnants = argument;
    else if (++i < argc)
        options->arg_nnants = argv [i];
    else
    {
        option = optstr__nnants;
        goto error_missing_arg_long;
    }
    options->opt_nnants = 1;
    break;
}
else if (strncmp (option + 1, optstr__nnls + 1, option_len - 1) == 0)
{
    if (option_len <= 2)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_nnls = argument;
    else if (++i < argc)
        options->arg_nnls = argv [i];
    else
    {
        option = optstr__nnls;

```

```

        goto error_missing_arg_long;
    }
    options->opt_nnl8 = 1;
    break;
}
goto error_unknown_long_opt;
case 'o':
    if (strncmp (option + 1, optstr__optimum + 1, option_len - 1) == 0)
    {
        if (argument != 0)
            options->arg_optimum = argument;
        else if (++i < argc)
            options->arg_optimum = argv [i];
        else
        {
            option = optstr__optimum;
            goto error_missing_arg_long;
        }
        options->opt_optimum = 1;
        break;
    }
    goto error_unknown_long_opt;
case 'q':
    if (strncmp (option + 1, optstr__q0 + 1, option_len - 1) == 0)
    {
        if (argument != 0)
            options->arg_q0 = argument;
        else if (++i < argc)
            options->arg_q0 = argv [i];
        else
        {
            option = optstr__q0;
            goto error_missing_arg_long;
        }
        options->opt_q0 = 1;
        break;
    }
    goto error_unknown_long_opt;
case 'r':
    if (strncmp (option + 1, optstr__ras + 1, option_len - 1) == 0)
    {
        if (option_len < 3)
            goto error_long_opt_ambiguous;
        if (argument != 0)
        {
            option = optstr__ras;
            goto error_unexpect_arg_long;
        }
        options->opt_ras = 1;
        break;
    }
    else if (strncmp (option + 1, optstr__rasranks + 1, option_len - 1) ==
0)
    {
        if (option_len <= 3)

```

```

    goto error_long_opt_ambiguous;
if (argument != 0)
    options->arg_rasranks = argument;
else if (++i < argc)
    options->arg_rasranks = argv [i];
else
{
    option = optstr__rasranks;
    goto error_missing_arg_long;
}
options->opt_rasranks = 1;
break;
}
else if (strncmp (option + 1, optstr__rho + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_rho = argument;
    else if (++i < argc)
        options->arg_rho = argv [i];
    else
    {
        option = optstr__rho;
        goto error_missing_arg_long;
    }
    options->opt_rho = 1;
    break;
}
goto error_unknown_long_opt;
case 't':
if (strncmp (option + 1, optstr__time + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_time = argument;
    else if (++i < argc)
        options->arg_time = argv [i];
    else
    {
        option = optstr__time;
        goto error_missing_arg_long;
    }
    options->opt_time = 1;
    break;
}
else if (strncmp (option + 1, optstr__tours + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_tours = argument;
    else if (++i < argc)
        options->arg_tours = argv [i];
}

```



```

    else
    {
        option = optstr__tours;
        goto error_missing_arg_long;
    }
    options->opt_tours = 1;
    break;
}
else if (strncmp (option + 1, optstr__tries + 1, option_len - 1) == 0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_tries = argument;
    else if (++i < argc)
        options->arg_tries = argv [i];
    else
    {
        option = optstr__tries;
        goto error_missing_arg_long;
    }
    options->opt_tries = 1;
    break;
}
else if (strncmp (option + 1, optstr__tsplibfile + 1, option_len - 1) ==
0)
{
    if (option_len <= 1)
        goto error_long_opt_ambiguous;
    if (argument != 0)
        options->arg_tsplibfile = argument;
    else if (++i < argc)
        options->arg_tsplibfile = argv [i];
    else
    {
        option = optstr__tsplibfile;
        goto error_missing_arg_long;
    }
    options->opt_tsplibfile = 1;
    break;
}
default:
error_unknown_long_opt:
    fprintf (stderr, STR_ERR_UNKNOWN_LONG_OPT, program_name, option);
    return -1;
error_long_opt_ambiguous:
    fprintf (stderr, STR_ERR_LONG_OPT_AMBIGUOUS, program_name, option);
    return -1;
error_missing_arg_long:
    fprintf (stderr, STR_ERR_MISSING_ARG_LONG, program_name, option);
    return -1;
error_unexpec_arg_long:
    fprintf (stderr, STR_ERR_UNEXPEC_ARG_LONG, program_name, option);
    return -1;
}

```

```

}
else
do
{
switch (*option)
{
case 'a':
if (option [1] != '\0')
options->arg_alpha = option + 1;
else if (++i < argc)
options->arg_alpha = argv [i];
else
goto error_missing_arg_short;
option = "\0";
options->opt_alpha = 1;
break;
case 'b':
if (option [1] != '\0')
options->arg_beta = option + 1;
else if (++i < argc)
options->arg_beta = argv [i];
else
goto error_missing_arg_short;
option = "\0";
options->opt_beta = 1;
break;
case 'c':
if (option [1] != '\0')
options->arg_elitistants = option + 1;
else if (++i < argc)
options->arg_elitistants = argv [i];
else
goto error_missing_arg_short;
option = "\0";
options->opt_elitistants = 1;
break;
case 'd':
if (option [1] != '\0')
options->arg_dlb = option + 1;
else if (++i < argc)
options->arg_dlb = argv [i];
else
goto error_missing_arg_short;
option = "\0";
options->opt_dlb = 1;
break;
case 'e':
if (option [1] != '\0')
options->arg_rho = option + 1;
else if (++i < argc)
options->arg_rho = argv [i];
else
goto error_missing_arg_short;
option = "\0";
options->opt_rho = 1;

```

```

break;
case 'f':
    if (option [1] != '\0')
        options->arg_rasranks = option + 1;
    else if (++i < argc)
        options->arg_rasranks = argv [i];
    else
        goto error_missing_arg_short;
    option = "\0";
    options->opt_rasranks = 1;
    break;
case 'g':
    if (option [1] != '\0')
        options->arg_nnants = option + 1;
    else if (++i < argc)
        options->arg_nnants = argv [i];
    else
        goto error_missing_arg_short;
    option = "\0";
    options->opt_nnants = 1;
    break;
case 'h':
    options->opt_help = 1;
    return i + 1;
case 'i':
    if (option [1] != '\0')
        options->arg_tsplibfile = option + 1;
    else if (++i < argc)
        options->arg_tsplibfile = argv [i];
    else
        goto error_missing_arg_short;
    option = "\0";
    options->opt_tsplibfile = 1;
    break;
case 'k':
    if (option [1] != '\0')
        options->arg_nnls = option + 1;
    else if (++i < argc)
        options->arg_nnls = argv [i];
    else
        goto error_missing_arg_short;
    option = "\0";
    options->opt_nnls = 1;
    break;
case 'l':
    if (option [1] != '\0')
        options->arg_localssearch = option + 1;
    else if (++i < argc)
        options->arg_localssearch = argv [i];
    else
        goto error_missing_arg_short;
    option = "\0";
    options->opt_localssearch = 1;
    break;
case 'm':

```

```

if (option [1] != '\0')
    options->arg_ants = option + 1;
else if (++i < argc)
    options->arg_ants = argv [i];
else
    goto error_missing_arg_short;
option = "\0";
options->opt_ants = 1;
break;
case 'o':
if (option [1] != '\0')
    options->arg_optimum = option + 1;
else if (++i < argc)
    options->arg_optimum = argv [i];
else
    goto error_missing_arg_short;
option = "\0";
options->opt_optimum = 1;
break;
case 'q':
if (option [1] != '\0')
    options->arg_q0 = option + 1;
else if (++i < argc)
    options->arg_q0 = argv [i];
else
    goto error_missing_arg_short;
option = "\0";
options->opt_q0 = 1;
break;
case 'r':
if (option [1] != '\0')
    options->arg_tries = option + 1;
else if (++i < argc)
    options->arg_tries = argv [i];
else
    goto error_missing_arg_short;
option = "\0";
options->opt_tries = 1;
break;
case 's':
if (option [1] != '\0')
    options->arg_tours = option + 1;
else if (++i < argc)
    options->arg_tours = argv [i];
else
    goto error_missing_arg_short;
option = "\0";
options->opt_tours = 1;
break;
case 't':
if (option [1] != '\0')
    options->arg_time = option + 1;
else if (++i < argc)
    options->arg_time = argv [i];
else

```

```

        goto error_missing_arg_short;
    option = "\0";
    options->opt_time = 1;
    break;
case 'u':
    options->opt_as = 1;
    break;
case 'v':
    options->opt_eas = 1;
    break;
case 'w':
    options->opt_ras = 1;
    break;
case 'x':
    options->opt_mmas = 1;
    break;
case 'y':
    options->opt_bwas = 1;
    break;
case 'z':
    options->opt_acs = 1;
    break;
default:
    fprintf (stderr, STR_ERR_UNKNOWN_SHORT_OPT, program_name, *option);
    return -1;
error_missing_arg_short:
    fprintf (stderr, STR_ERR_MISSING_ARG_SHORT, program_name, *option);
    return -1;
}
} while (++option != '\0');
}
return i;
}

void check_out_of_range ( double value, double MIN, double MAX, char *optionName
)
/*
    FUNCTION: check whether parameter values are within allowed range
    INPUT:    none
    OUTPUT:   none
    COMMENTS: none
*/
{
    if ((value<MIN)|| (value>MAX)){
        if(0 == PC_myProc) fprintf(stderr, "Error: Option '%s' out of
range\n", optionName);
        exit(1);
    }
}

int PC_parse_commandline (int myProc, int argc, char *argv [])
{
    PC_myProc = myProc;

    int i;

```

```

const char *programe;
struct options options;

programe = argv [0] != NULL && *(argv [0]) != '\0'
? argv [0]
: "acotsp";

i = parse_options (&options, programe, argc, argv);

if (i < 2)
{
fprintf (stderr, "No options are specified\n");
fprintf (stderr, "Try '%s --help' for more information.\n",
programe);
exit(1);
}

if (options.opt_help)
{
if(0 == PC_myProc) printf ("Usage: %s [OPTION]... [ARGUMENT]...\n\n"
"Options:\n" STR_HELP, programe);
exit(0);
}

/* if(0 == PC_myProc) puts ("\t OPTIONS:"); */

if ( options.opt_time ) {
max_time = atof(options.arg_time);
/* if(0 == PC_myProc) fputs (" -t --time ", stdout);
if (options.arg_time != NULL)
if(0 == PC_myProc) printf ("with argument \'%.3f\'\n", max_time); */
check_out_of_range( max_time, 0.0, 86400., "max_time (seconds)");
} else {
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: time limit is set to default
%.3f seconds\n", max_time); */
}

if ( options.opt_tries ) {
max_tries = atol(options.arg_tries);
/* if(0 == PC_myProc) fputs (" -r --tries ", stdout);
if (options.arg_tries != NULL)
if(0 == PC_myProc) printf ("with argument \'%ld\'\n", max_tries); */
check_out_of_range( max_tries, 1, MAXIMUM_NO_TRIES, "max_tries (tries)");
} else {
/* max_tries = 10; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: number of trials is set to
default %ld\n", max_tries); */
}

if ( options.opt_tours ) {
max_tours = atol(options.arg_tours);
/* if(0 == PC_myProc) fputs (" -s --tours ", stdout);
if (options.arg_tries != NULL)
if(0 == PC_myProc) printf ("with argument \'%ld\'\n", max_tours); */
check_out_of_range( max_tours, 1, LONG_MAX, "max_tries (tries)");
}

```

```

    } else {
        /* max_tours = 100; */
        /* if(0 == PC_myProc) fprintf(stderr, "\tNote: maximum number of tours is set
to default %ld\n", max_tours); */
    }

    if ( options.opt_optimum )
    {
        optimal = atol(options.arg_optimum);
        /* if(0 == PC_myProc) fputs ( " -o --optimum ", stdout);
if (options.arg_optimum != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", optimal); */
    } else {
        /* optimal = 1; */
        /* if(0 == PC_myProc) fprintf(stderr, "\tNote: optimal solution value is set to
default %ld\n", optimal); */
    }

    if ( options.opt_tsplibfile )
    {
        strncpy( name_buf, options.arg_tsplibfile, strlen( options.arg_tsplibfile ) );
        /* if(0 == PC_myProc) fputs ( " -i --tsplibfile ", stdout);
if (options.arg_tsplibfile != NULL)
    if(0 == PC_myProc) printf ("with argument \'%s\'\n", name_buf ); */
    }

    if ( options.opt_ants ) {
        n_ants = atol(options.arg_ants);
        /* if(0 == PC_myProc) fputs ( " -m --ants ", stdout);
if (options.arg_ants != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", n_ants); */
        check_out_of_range( n_ants, 1, MAX_ANTS-1, "n_ants");
    } else {
        /* n_ants = 25; */
        /* if(0 == PC_myProc) fprintf(stderr, "\tNote: number of ants is set to
default %ld\n", n_ants); */
    }

    if ( options.opt_nnants ) {
        nn_ants = atol(options.arg_nnants);
        /* if(0 == PC_myProc) fputs ( " -m --ants ", stdout);
if (options.arg_ants != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", nn_ants); */
        check_out_of_range( n_ants, 1, 100, "nn_ants");
    } else {
        /* nn_ants = 20; */
        /* if(0 == PC_myProc) fprintf(stderr, "\tNote: number of nearest neighbours
in tour construction is set to default %ld\n", nn_ants); */
    }

    if ( options.opt_alpha ) {
        alpha = atof(options.arg_alpha);
        /* if(0 == PC_myProc) fputs ( " -a --alpha ", stdout);
if (options.arg_alpha != NULL)
    if(0 == PC_myProc) printf ("with argument \'%f\'\n", alpha); */
    }

```

```

check_out_of_range( alpha, 0., 100., "alpha");
    } else {
/* alpha = 1.0; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: alpha is set to default %f\n",
alpha); */
    }

    if ( options.opt_beta ) {
beta = atof(options.arg_beta);
/* if(0 == PC_myProc) fputs (" -b --beta ", stdout);
if (options.arg_beta != NULL)
    if(0 == PC_myProc) printf ("with argument \'%f\'\n", beta); */
check_out_of_range( beta, 0., 100., "beta");
    } else {
/* beta = 2.0; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: beta is set to default %f\n",
beta); */
    }

    if ( options.opt_rho ) {
rho = atof(options.arg_rho);
/* if(0 == PC_myProc) fputs (" -e --rho ", stdout);
if (options.arg_rho != NULL)
    if(0 == PC_myProc) printf ("with argument \'%f\'\n", rho); */
check_out_of_range( rho, 0., 1., "rho");
    } else {
/* rho = 0.5; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: rho is set to default %f\n",
rho); */
    }

    if ( options.opt_q0 ) {
q_0 = atof(options.arg_q0);
/* if(0 == PC_myProc) fputs (" -q --q0 ", stdout);
if (options.arg_q0 != NULL)
    if(0 == PC_myProc) printf ("with argument \'%f\'\n", q_0); */
check_out_of_range( q_0, 0., 1., "q0");
    } else {
/* q_0 = 0.0; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: q_0 is set to default %f\n",
q_0); */
    }

    if ( options.opt_elitistants ) {
elitist_ants = atol(options.arg_elitistants);
/* if(0 == PC_myProc) fputs (" -m --ants ", stdout);
if (options.arg_elitistants != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", elitist_ants); */
check_out_of_range( n_ants, 0, LONG_MAX, "elitistants");
    } else {
/* elitist_ants = 100; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: number of elitist ants is set
to default %ld\n", elitist_ants); */
    }

```



```

    if ( options.opt_rasranks ) {
ras_ranks = atol(options.arg_rasranks);
/* if(0 == PC_myProc) fputs (" -m --ants ", stdout);
if (options.arg_rasranks != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", ras_ranks); */
check_out_of_range( n_ants, 0, LONG_MAX, "rasranks");
    } else {
/* ras_ranks = 6; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: number of ranks is set to
default %ld\n", ras_ranks); */
    }

    if ( options.opt_nnls ) {
nn_ls = atol(options.arg_nnls);
/* if(0 == PC_myProc) fputs (" -k --nnls ", stdout);
if (options.arg_nnls != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", nn_ls); */
check_out_of_range( n_ants, 0, LONG_MAX, "nnls");
    } else {
/* nn_ls = 20; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: number nearest neighbours in
local search is set to default %ld\n", nn_ls); */
    }

    if ( options.opt_localsearch ) {
ls_flag = atol(options.arg_localsearch);
/* if(0 == PC_myProc) fputs (" -l --localsearch ", stdout);
if (options.arg_localsearch != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", ls_flag); */
check_out_of_range( n_ants, 0, LONG_MAX, "ls_flag");
    } else {
/* ls_flag = 3; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: local search flag is set to
default 3 (3-opt)\n"); */
    }

    if ( options.opt_dlb ) {
dlb_flag = atol(options.arg_dlb);
/* if(0 == PC_myProc) fputs (" -d --dlb ", stdout);
if (options.arg_dlb != NULL)
    if(0 == PC_myProc) printf ("with argument \'%ld\'\n", dlb_flag); */
check_out_of_range( dlb_flag, 0, 1, "dlb_flag");
    } else {
/* dlb_flag = TRUE; */
/* if(0 == PC_myProc) fprintf(stderr, "\tNote: dlb flag is set to default 1
(use don't look bits)\n"); */
    }

    if ( options.opt_as ) {
as_flag = TRUE;
eas_flag = FALSE; ras_flag = FALSE; mmas_flag = FALSE;
bwas_flag = FALSE; acs_flag = FALSE;
/* if(0 == PC_myProc) fprintf(stderr, "as_flag is set to 1, run Ant
System\n"); */
/* } else { */

```

```

/*   if(0 == PC_myProc) fprintf(stderr, "\tNote: eas_flag is set to default 0
(don't run Ant System)\n"); */
}

    if ( options.opt_eas ) {
        eas_flag = TRUE;
        as_flag = FALSE; ras_flag = FALSE; mmas_flag = FALSE;
        bwas_flag = FALSE; acs_flag = FALSE;
        /*   if(0 == PC_myProc) fprintf(stderr, "eas_flag is set to 1, run Elitist Ant
System\n"); */
        /*   } else { */
        /*   if(0 == PC_myProc) fprintf(stderr, "\tNote: eas_flag is set to default 0
(don't run Elitist Ant System)\n"); */
        }

        if ( options.opt_ras ) {
            ras_flag = TRUE;
            as_flag = FALSE; eas_flag = FALSE; mmas_flag = FALSE;
            bwas_flag = FALSE; acs_flag = FALSE;
            /*   if(0 == PC_myProc) fprintf(stderr, "ras_flag is set to 1, run rank-based
Ant System\n"); */
            /*   } else { */
            /*   if(0 == PC_myProc) fprintf(stderr, "\tNote: ras_flag is set to default 0
(don't run rank-based Ant System)\n"); */
            }

            if ( options.opt_mmas ) {
                mmas_flag = TRUE;
                as_flag = FALSE; eas_flag = FALSE; ras_flag = FALSE;
                bwas_flag = FALSE; acs_flag = FALSE;
                /*   if(0 == PC_myProc) fprintf(stderr, "mmas_flag is set to 1, run MAX-MIN Ant
System\n"); */
                /*   } else { */
                /*   if(0 == PC_myProc) fprintf(stderr, "\tNote: mmas_flag is set to default 1
(run MAX-MIN Ant System!)\n"); */
                }

                if ( options.opt_bwas ) {
                    bwas_flag = TRUE;
                    as_flag = FALSE; eas_flag = FALSE; mmas_flag = FALSE;
                    ras_flag = FALSE; acs_flag = FALSE;
                    /*   if(0 == PC_myProc) fprintf(stderr, "bwas_flag is set to 1, run Best-Worst
Ant System\n"); */
                    /*   } else { */
                    /*   if(0 == PC_myProc) fprintf(stderr, "\tNote: bwas_flag is set to default 0
(don't run Best-Worst Ant System)\n"); */
                    }

                    if ( options.opt_acs ) {
                        acs_flag = TRUE;
                        as_flag = FALSE; eas_flag = FALSE; mmas_flag = FALSE;
                        ras_flag = FALSE; bwas_flag = FALSE;
                        /*   if(0 == PC_myProc) fprintf(stderr, "acs_flag is set to 1, run Ant Colony
System\n"); */
                        /*   } else { */

```

```

/* if(0 == PC_myProc) fprintf(stderr, "\tNote: acs_flag is set to default 0
(don't run Ant Colony System)\n"); */
}

/* if(0 == PC_myProc) puts ("Non-option arguments:"); */

while (i < argc) {
fprintf (stderr, " '%s'\n", argv [i++]);
fprintf (stderr, "\nThere were non-option arguments\n");
fprintf (stderr, "I suspect there is something wrong, maybe wrong option name;
exit\n");
exit(1);
}

return 0;
}

```

sos.h

```
/* SOS : Streams, Overlap and Shortcut system */
/* Ernesto Gomez 3/02 */

/* includes */

#include "mpi.h"
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <string.h>

#define SOS_Version .1
#define SOS_MPI

/* ===== standard defines ===== */

#define TRUE 1
#define FALSE 0
#define NIL 0
#define NONE -1
#define BYTE unsigned char

/* systemSize definitions */
#define MAXFUN 128
#define MAXVAR 1024
int MAXSTAT=128; /* allow reset of MAXSTAT */
#define MAXNODE 128
#define MSGSIZE 64

/* number of calls to dosospoll before status gets checked
   outside the queue loop */
#define STAT 30

/* states */

#define Start 0
#define MayS 1
#define MayR 2
#define MayA 3
#define MustS 4
#define MustR 5
#define MustA 6
#define ShortCut 7
#define ShRoot 8
#define Quiet 9
#define Error 10

/* use MPI datatypes if SOS_MPI is defined
   MPI_FLOAT - complex is 2 items of this
   MPI_DOUBLE - dcomplex is 2 items of this
   MPI_INTEGER
   MPI_LONG
```

```

MPI_CHAR

- otherwise :

#define SOS_FLOAT 1
#define SOS_DOUBLE 2
#define SOS_INTEGER 3
#define SOS_LONG 4
#define SOS_CHAR 5

*/

/* messages */

#define NOM -1 /* No Message */
#define N 0 /* don't join */
#define J 1 /* join */
#define S 2 /* Send */
#define R 3 /* Receive */
#define A 4 /* collective - (All) */
#define SP 12 /* final send phase of All operation */
#define RH 5 /* Right Hand - variable is read */
#define LH 6 /* Left Hand - variable is written */
#define CS 7 /* Clear to Send data _(received) sent by receiver to sender */
#define MSG 8 /* data Message _(received) */
#define Z 9 /* Zero count - finished */
#define C 10 /* shortCut */
#define DC 11 /* Define shortCut */

/* automaton - overlap+shortcut
   use: SOSfsm[old-state][message] -> new-state */

/* change - assuming A is a broadcast, then will receive one
   message and then only has to send - so goes over from mayA
   to mayS => entry is 1 for MS,MayA */

static int SOSfsm[11][13] = {
/* msg N J S R A RH LH CS MS Z C DC SP states */
0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 7, 8, 0, /* 0 Start */
10,10,1, 1, 1, 1, 4, 1, 10,0, 7, 8, 1, /* 1 MayS */
10,10,2, 2, 2, 5, 2, 2, 2, 0, 7, 8, 2, /* 2 MayR */
10,10,3, 3, 3, 6, 5, 3, 3, 0, 7, 8, 1, /* 3 MayA */
10,10,4, 4, 4, 4, 4, 4, 10,0, 7, 10,4, /* 4 MustS */
10,10,5, 5, 5, 5, 5, 5, 5, 0, 7, 10,5, /* 5 MustR */
10,10,6, 6, 6, 6, 5, 6, 6, 0, 7, 10,4, /* 6 MustA */
9, 2, 10,10,10,10,10,7, 10,10,10,10,7, /* 7 ShortCut */
8, 8, 10,10,10,10,10,8, 10,1, 8, 10,8, /* 8 ShRoot */
0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 7, 8, 0, /* 9 Quiet - End */
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 /* 10 Error */
};

/* making entry LH,MayR = 2 instead of 5 allos system to resolve deadlock
   dynamically - igonres write/write conflict?? */

/* blocking flag -> 1 for blocking states, 0 to allow exit */

```

```

static int SOSblock[11] = {
    0, /* 0 Start */
    0, /* 1 MayS */
    0, /* 2 MayR */
    0, /* 3 MayA */
    1, /* 4 MustS */
    1, /* 5 MustR */
    1, /* 6 MustA */
    1, /* 7 ShortCut */
    1, /* 8 ShRoot */
    0, /* 9 Quiet - End */
    0, /* 10 Error */
};

/* global definitions */

typedef union SOSitemType {
    int *intP;
    float *floatP;
    double *doubleP;
    long *longP;
    char *charP;
} SOSitem;

typedef struct SOSfunArrayItem {
    long *funPT;
    int funType;
} SOSfunI;

typedef struct SOSvarArrayItem {
    BYTE *varPT; /* generic pointer to everything */
    long size; /* size in bytes */
    int type;
    int fsm;
    int status[MAXNODE]; /* keep only local status here */
    MPI_Request handle[MAXNODE];
    MPI_Comm channel; /* if not MPI, replace with tag */
    int age;
    int refcount;
    int staticv;
    short mess[6]; /* status message create buffer for each */
} SOSvar;

typedef struct SOSvarStackItem {
    int varHandle;
    int pending;
    struct SOSvarStackItem *next;
} SOSvarStack;

/* communication queue entry */
typedef struct SOSqItem {
    int ivar; /* variable index */
    int send; /* S for send, CS->R for receive */
}

```

```

int to;
int from;
int fun;      /* function number for gather, -1 for none */
BYTE *work; /* workspace */
int sel; /* function extra parameter */
struct SOSqItem *nextCom; /* next in same communication */
struct SOSqItem *prevCom; /* previous in same communication */
struct SOSqItem *next; /* next in queue */
struct SOSqItem *prev; /* previous in queue */
int iremote; /* variable index at remote node */
int flag; /* misc. code */
int group; /* flag indicating order doesn't matter */
long size;
long off;
} SOSq;

typedef struct SOSloopItem {
    int level;
    struct SOSloopItem *next;
} SOSl;

static SOSfunI SOSfuns[MAXFUN]; /* functions array */
static SOSvar SOSvars[MAXVAR]; /* vars array */

static struct SOSqItem *SOSqu=NULL; /* queue */
static struct SOSvarStackItem *SOSvarS=NULL; /* var stack */

static int ME, SOSv; /* SOS thisnode and next var */
/* stream global data */
static int SOSlevel[MAXNODE]; /* array of nesting level, init to 0 */
static int SOSstream[MAXNODE]; /* array of indices in stream, init to 0..MAX */
/* this array is different for each stream;
   the index is the position in the stream and
   the content is the global (MPI) id */
static int SOSwork[MAXNODE]; /* work array for SOS splits */
static int SOSworkP; /* handle for SOS array */
static int SOSloop[MAXNODE]; /* loop nesting array */
static struct SOSloopItem *SOSls=NULL; /* loop level stack */
static int SOSsplitF=FALSE; /* split in progress flag */
static int SOSsplitC=0; /* split in progress streamcount */
static int SOSsplitS; /* split path selector */

static int MIPScnt; /* stream count */
static int MIPStop; /* stream high index */
static int MIPSbot; /* stream low index */

/* MPI globals */
static MPI_Status *MPIstate;
static MPI_Request MPIhandle;
static MPI_Comm SOSchannel;
static MPI_Comm SOSdata;

/* timer globals */
static int SOSbegin;
#define SOStime (MPI_Wtime()-SOSbegin)/1000

```

```

/* Planguage globals */
static int thisnode,numnode,cubedim;
static short SOSmess[6];
static MPI_Request SOShandle=MPI_REQUEST_NULL;

typedef struct SOSmessStack {
    short mess[6];
    struct SOSmessStack *next;
} SOSmessageStack ;

static struct SOSmessStack *SOSmessS=NULL;
static struct SOSmessStack *SOSmessW=NULL;

/* standard message
    mess[0] sending node
    mess[1] variable index at message receiver
    mess[2] message code
    mess[3] variable index at message source
    mess[4] variable size
    mess[5] variable offset
*/

/* Global change flags */
static int SOSchange=FALSE;
static int SOSforce=FALSE;
static int SOSreceive=0;
/* global top variable and function */
static int lastVar;
static int lastFun;
/* global dummy variable, index and size - .98*/
static int SOSdP, ONE=1, ZER=0;
static long SOSd;
/* set values of function handles to match assignment
    in sosinitfun */
static int SOSADD2=0;
static int SOSMUL2=1;
static int SOSMAX2=2;
static int SOSMIN2=3;
static int SOSADD1=4;
static int SOSMUL1=5;
static int SOSMAX1=6;
static int SOSMIN1=7;
static int SOSADDI=8;
static int SOSMULI=9;
static int SOSADDL=10;
static int SOSMULL=11;
static int SOSVOR=12;
static int SOSVAND=13;

/* signal control params */
/* poll repeat interval, microseconds */

static int SOS_REPEAT=100;

```



```
static sigset_t SOSset;
static sigset_t SOSoldSet;
static sigset_t SOSstestSet;

/* shortcut flag */
static int SOSshort=NONE;
static int SOSshortVar=NONE;
static int SOSshortLevel=0;

/* timer */
static long sosalarmcount;

/*
#define TIMER ITIMER_REAL
#define ALARM SIGALRM
*/
#define TIMER ITIMER_VIRTUAL
#define ALARM SIGVTALRM

#include "sosdefs.h"
```

sosdefs.h

```

/* stream functions.
   PI is process number
   SI is id within stream
   PI is a predicate value that selects a stream
   D is a decrement number for stream end,
   indicates how many nesting levels merge
*/

int sosstreamstart_(int *P); /* joins processes with common P */
int sosstreamend_(int *D); /* merges D nesting levels of streams */
int sosstreamlevel_(int *L); /* returns nesting level of this stream */
int sosstreammember_(int *PN, int *result); /* TRUE if PN is in this stream */
int sosme_(int *SI); /* SI of this process */
int sosid_(int *PN, int *SI); /* PN -> SI */
int sospn_(int *SI, int *PN); /* SI -> PN */
int sosstreamcount_(int *count); /* returns number of pocesses in stream */
int sosstreamtop_(int *top);
int sosstreambottom_(int *bottom);
int sosgetstream_(int *stream);
int sossplit_(int *P, int *nu);
int sosendsplit_();
int soscount_();
int *sosstream_();
int sostop_();
int sosbot_();

#define SPLITCHECK if(SOSsplitF) return NONE
#define DOSPLIT if(SOSsplitF) sosendsplit_()

/* overlap - interface */

int sospoint2point_(int *ivarRH, int *ivarLH, long *Count, long *OffRH,
    long *OffLH, int *SOStype, int *fromP, int *toP);
int sosgroup2group_(void *FromBuffer, void *ToBuffer, long *Count, long *Off,
    int *SOStype, int *FromPlist, int *ToPlist, int *ListCount);
int sospoint2group_(void *FromBuffer, void *ToBuffer, long *Count, long *Off,
    int *SOStype, int *FromP, int *ToPlist, int *ListCount);
int sosbroadcast_(int *ivarRH, int *ivarLH, long *Count, long *Off,
    int *SOStype, int *FromP);
int sospointreduce_(int *ivarRH, int *ivarLH, long *Count, long *Off,
    int *SOStype, int *FunP, int *sel, int *rootP);
int sosallreduce_(int *ivarRH, int *ivarLH, long *Count, long *Off,
    int *SOStype, int *FunP, int *sel);
int sosgroupreduce_(void *FromBuffer, void *ToBuffer, long *Count, long *Off,
    int *SOStype, int *RootPlist, int *FunP);
int sosdorh_(int *ivar);
int sosdolh_(int *ivar);

/* overlap - internal */

/* shortcut */
int sosshortcut_(int *ivar, int *DStream);
/* assert shortcut */

```

```

int sosshortcutstart_(int *PN);
    /* sets DC local, sends C */
int sosshortcutend_();
    /* drops stream nesting level by 1 -> SOSstreamEnd */
int sosshortcuttest_();
    /* checks if we're in shortcut, returns PN of shortcutter or -1 */

/* init */

int sosinit_(int *argc, char ***argv);
int sosinitfun_();
int sosaddvar_(int *buffer, long* size, int* type, int* iVar, int* staticv);
int sosaddfun_(int (*fun)(), int *iVar);
int sosremovevar_(int *iVar);
int soslastvar_(int *iVar); /* returns handle of top var */
int sossizes_(int *iVar); /* returns size in bytes of var */

int sospopvar_(); /* mark for remove all variables with index >= top of
varstack, pops */
int sospushvar_(); /* saves value of next var to allocate */
int dosospopvar_(); /* actually removes vars */
int popLevel; /* global level for popvar */

/* variable insertion and deletion */
int addVar(void *buffer, long *size, int *type, int *staticv);
int SOSindex(int staticv); /* returns index for new var insertion */
int findVar(void *buffer); /* returns index of variable */
/* removeVar is not safe inside any stream <> initial stream */
int removeVar(int iVar); /* removes variable */
int selectRemoveVar(); /* selects var for removal */
struct SOSqItem *scanQueue_(int iVar); /* returns item in queue that references
iVar */

/* polling */

int getStatus(); /* get all pending status msgs */
int sospoll_();
/* sospoll fsa processing:
sets: MSG, CS, Z
uses: RH, LH, C, DC
sends: CS
*/

/* signals */
int dosospoll_();
void sosalarm_();
int setsosalarm_(int *repeat);
int sosblock_();
int sosunblock_();

/* overlap queue */

int setPoint2Point( int iVarLH, int iVarRH, int fromP, int toP,
long Size, long OffRH, long OffLH );
/* sets S, R */

```

```

int setBroadcast(int from, int ivarRH, int ivarLH, long Size, long Off);
/* sets S, A - calls makeBroadcast */
int setReduce(int ivarRH, int ivarLH, int ifun, int sel, long Size, long Off);
/* sets A - calls makeReduce */
struct SOSqItem *makeBroadcast(int ivar, int root, int *group,
    struct SOSqItem *prev, long Size, long Off);
struct SOSqItem *makeReduce(int ivar, int root, int *group, int ifun, int sel,
    long Size, long Off);
/* utility overlap functs, called by other functs */
struct SOSqItem *makeQItem(int ivar, int sendF, int Nto, int Nfrom, int Nfun,
    long Size, long Off, int iremote);
int offset(int i, int off, int size); /* maps i to use root=off */

/* finite automata */
int stepFSA(int ivar, int message);
/* defs to acces fsa */
#define getFSA( ivar ) SOSvars[ivar].fsm
#define getBlock( ivar ) SOSblock[ SOSvars[ivar].fsm ]
#define RefCount( ivar ) SOSvars[ivar].refcount
#define IncRef( ivar ) SOSvars[ivar].refcount++
#define DecRef( ivar ) SOSvars[ivar].refcount--

/* utility */
int log2ceil(int ncube);
struct SOSqItem *removeItem(struct SOSqItem *q);
void printQueue(int id, struct SOSqItem *s);
void printCom(int id, struct SOSqItem *s);
int sosclear_();
int sosclearstack_();
int SOSdoMess(); /* process status message */
int SOScheckMessStack(int iVar ); /* check for waiting message */

int SOScount(); /* count this stream */
int SOScountS(int level); /* count this stream and up */
int SOSme();

/* message passing */
int doSend(int ivar, int ito, long size, long offset, int iwhat);
int doIrecv(int ivar, int ifrom, long offset, long size );
int doIrecv2(int ivar, int ifrom, BYTE *buffer, long size );
int msgTransfer(void *mess, int ifrom, int ito, int size);
int stateSend(int ivar, int ito);
void sendCS(struct SOSqItem *q); /* make CS entry send message and step to
receive */

/* timing */
double sosclock_();
long soscounter_();

/* stac.h - doubly linked stack defines */

/* a stack item is any structure that contains data and
a pointers to an item of the same type called "next"
and "prev".

```

usage: declare the stack to be a pointer to a stack item structure and set its value to NIL.
then, initialize data items as needed, and PUSH them on the stack to create, or pop to empty them.

search routines depend on the actual data structures and may not be generalized.

```

*/
#define NIL 0

/* init stack, place item on it */
#define PUSH( s , i ) i->next=s; if( s ) s->prev=i; s=i ; SOSchange=1

/* init stack, place item on it - standard */
#define SPUSH( s , i ) i->next=s; s=i

/* remove item from stack top and return pointer to it */
#define POP( s , i ) i=s; s=s->next; i->next=NIL; if( s ) s->prev=NIL;

/* remove item from stack top and return pointer to it */
#define SPOP( s , i ) if(s) { i=s; s=s->next; i->next=NIL; } else { i=s; }

/* get pointer to next item on stack */
#define NEXT( s , i ) i=s->next

/* get pointer to previous item on stack */
#define PREV( s , i ) i=s->prev;

/* true if stack is empty */
#define EOS(s) (s==NIL)

/* sample stack item definition
struct stacItem {
    char *value;
    struct stacItem *next;
    struct stacItem *prev;
};
*/

/*
#define SOSBLOCK sigprocmask(SIG_BLOCK, &SOSset, NULL);
#define SOSCLEAR sigprocmask(SIG_UNBLOCK, &SOSset, NULL);
#define SOSCLEAR sigprocmask(SIG_SETMASK, &SOSset, NULL);
#define SOSCLEAR sigprocmask(SIG_SETMASK, &SOSoldSet, NULL);
*/

#define SOSBLOCK sigprocmask(SIG_BLOCK, &SOSset, NULL);
#define SOSCLEAR sigprocmask(SIG_UNBLOCK, &SOSset, NULL);
#define SOS POLL sospoll_();
#define DOSOSPOLL raise(ALARM);
/*
#define DOSOSCLEAR sigprocmask(SIG_SETMASK, &SOSoldSet, NULL);

```

```
#define SOSBLOCK  
#define SOSCLEAR setsosalarm_(&zer);  
#define SOS POLL raise(SIGVTALRM);  
*/
```

sos-new.h

```
/* SOS : Streams, Overlap and Shortcut system */
/* Ernesto Gomez 3/02 */

/* includes */

#include "mpi.h"
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <string.h>

#define SOS_Version .1
#define SOS_MPI

/* ===== standard defines ===== */

#define TRUE 1
#define FALSE 0
#define NIL 0
#define NONE -1
#define BYTE unsigned char

/* systemSize definitions */
#define MAXFUN 128
#define MAXVAR 1024
#define MAXNODE 64
#define MSGSIZE 64

/* number of calls to dosospoll before status gets checked
   outside the queue loop */
#define STAT 30

/* states */

#define Start 0
#define MayS 1
#define MayR 2
#define MayA 3
#define MustS 4
#define MustR 5
#define MustA 6
#define ShortCut 7
#define ShRoot 8
#define Quiet 9
#define Error 10

/* use MPI datatypes if SOS_MPI is defined
   MPI_FLOAT - complex is 2 items of this
   MPI_DOUBLE - dcomplex is 2 items of this
   MPI_INTEGER
   MPI_LONG
   MPI_CHAR
```

```

- otherwise :

#define SOS_FLOAT 1
#define SOS_DOUBLE 2
#define SOS_INTEGER 3
#define SOS_LONG 4
#define SOS_CHAR 5

*/

/* messages */

#define NOM -1 /* No Message */
#define N 0 /* don't join */
#define J 1 /* join */
#define S 2 /* Send */
#define R 3 /* Receive */
#define A 4 /* collective - (All) */
#define SP 12 /* final send phase of All operation */
#define RH 5 /* Right Hand - variable is read */
#define LH 6 /* Left Hand - variable is written */
#define CS 7 /* Clear to Send data _(received) sent by receiver to sender */
#define MSG 8 /* data Message _(received) */
#define Z 9 /* Zero count - finished */
#define C 10 /* shortCut */
#define DC 11 /* Define shortCut */

/* automaton - overlap+shortcut
   use: SOSfsm[old-state][message] -> new-state */

/* change - assuming A is a broadcast, then will receive one
   message and then only has to send - so goes over from mayA
   to mayS => entry is 1 for MS,MayA */

static int SOSfsm[11][13] = {
/* msg N J S R A RH LH CS MS Z C DC SP      states */
  0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 7, 8, 0, /* 0 Start */
  10,10,1, 1, 1, 1, 4, 1, 10,0, 7, 8, 1, /* 1 MayS */
  10,10,2, 2, 2, 5, 2, 2, 2, 0, 7, 8, 2, /* 2 MayR */
  10,10,3, 3, 3, 6, 5, 3, 3, 0, 7, 8, 1, /* 3 MayA */
  10,10,4, 4, 4, 4, 4, 4, 10,0, 7, 10,4, /* 4 MustS */
  10,10,5, 5, 5, 5, 5, 5, 5, 0, 7, 10,5, /* 5 MustR */
  10,10,6, 6, 6, 6, 5, 6, 6, 0, 7, 10,4, /* 6 MustA */
  9, 2, 10,10,10,10,10,7, 10,10,10,10,7, /* 7 ShortCut */
  8, 8, 10,10,10,10,10,8, 10,1, 8, 10,8, /* 8 ShRoot */
  0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 7, 8, 0, /* 9 Quiet - End */
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 /* 10 Error */
};

/* making entry LH,MayR = 2 instead of 5 allos system to resolve deadlock
   dynamically - igonres write/write conflict?? */

/* blocking flag -> 1 for blocking states, 0 to allow exit */

```



```

static int SOSblock[11] = {
    0, /* 0 Start */
    0, /* 1 MayS */
    0, /* 2 MayR */
    0, /* 3 MayA */
    1, /* 4 MustS */
    1, /* 5 MustR */
    1, /* 6 MustA */
    1, /* 7 ShortCut */
    1, /* 8 ShRoot */
    0, /* 9 Quiet - End */
    0, /* 10 Error */
};

/* global definitions */

typedef union SOSitemType {
    int *intP;
    float *floatP;
    double *doubleP;
    long *longP;
    char *charP;
} SOStype;

typedef struct SOSfunArrayItem {
    long *funPT;
    int funType;
} SOSfunI;

typedef struct SOSvarArrayItem {
    BYTE *varPT; /* generic pointer to everything */
    long size; /* size in bytes */
    int type;
    int fsm; /* finite state machine */
    int status[MAXNODE]; /* keep only local status here ?*/
    MPI_Request handle[MAXNODE];
    MPI_Comm channel; /* if not MPI, replace with tag */
    int age; /* use by sosindex and selectremove var */
    int refcount;
    int staticv;
    short mess[6]; /* status message create buffer for each*/
} SOSvar;

typedef struct SOSvarStackItem {
    int varHandle;
    int pending;
    struct SOSvarStackItem *next;
} SOSvarStack;

/* communication queue entry */
typedef struct SOSqItem {
    int ivar; /* variable index */
    int send; /* S for send, CS->R for receive*/
    int to;

```

```

int from;
int fun;      /* function number for gather, -1 for none */
BYTE *work; /* workspace */
int sel; /* function extra parameter */
struct SOSqItem *nextCom; /* next in same communication */
struct SOSqItem *prevCom; /* previous in same communication */
struct SOSqItem *next; /* next in queue */
struct SOSqItem *prev; /* previous in queue */
int iremote; /* variable index at remote node */
int flag; /* misc. code */
int group; /* flag indicating order doesn't matter */
long size;
long off;
} SOSq;

typedef struct SOSloopItem {
    int level;
    struct SOSloopItem *next;
} SOSl;

static SOSfunI SOSfuns[MAXFUN]; /* functions array */
static SOSvar SOSvars[MAXVAR]; /* vars array */

static struct SOSqItem *SOSqu=NULL; /* queue */
static struct SOSvarStackItem *SOSvarS=NULL; /* var stack */

static int ME, SOSv; /* SOS thisnode and next var */
/* stream global data */
static int SOSlevel[MAXNODE]; /* array of nesting level, init to 0 */
static int SOSstream[MAXNODE]; /* array of indices in stream, init to 0..MAX */
/* this array is different for each stream;
   the index is the position in the stream and
   the content is the global (MPI) id */
static int SOSwork[MAXNODE]; /* work array for SOS splits */
static int SOSworkP; /* handle for SOS array */
static int SOSloop[MAXNODE]; /* loop nesting array */
static struct SOSloopItem *SOSls=NULL; /* loop level stack */
static int SOSsplitF=FALSE; /* split in progress flag */
static int SOSsplitC=0; /* split in progress streamcount */
static int SOSsplitS; /* split path selector */

static int MIPScnt; /* stream count */
static int MIPStop; /* stream high index */
static int MIPSbot; /* stream low index */

/* MPI globals */
static MPI_Status *MPIstate;
static MPI_Request MPIhandle;
static MPI_Comm SOSchannel;
static MPI_Comm SOSdata;

/* timer globals */
static int SOSbegin;
#define SOSStime (MPI_Wtime()-SOSbegin)/1000

```

```

/* Planguage globals */
static int thisnode,numnode,cubedim;
static short SOSmess[6];
static MPI_Request SOShandle=MPI_REQUEST_NULL;

typedef struct SOSmessStack {
    short mess[6];
    struct SOSmessStack *next;
} SOSmessageStack ;

static struct SOSmessStack *SOSmessS=NIL;
static struct SOSmessStack *SOSmessW=NIL;

/* standard message
    mess[0] sending node
    mess[1] variable index at message receiver
    mess[2] message code
    mess[3] variable index at message source
    mess[4] variable size
    mess[5] variable offset
*/

/* Global change flags */
static int SOSchange=FALSE;
static int SOSforce=FALSE;
static int SOSreceive=0;
/* global top variable and function */
static int lastVar;
static int lastFun;
/* global dummy variable, index and size - .98*/
static int SOSdP,ONE=1;
static long SOSd;
/* set values of function handles to match assignment
    in sosinitfun */
static int SOSADD2=0;
static int SOSMUL2=1;
static int SOSMAX2=2;
static int SOSMIN2=3;
static int SOSADD1=4;
static int SOSMUL1=5;
static int SOSMAX1=6;
static int SOSMIN1=7;
static int SOSADDI=8;
static int SOSMULI=9;
static int SOSADDL=10;
static int SOSMULL=11;
static int SOSVOR=12;
static int SOSVAND=13;

/* signal control params */
/* poll repeat interval, microseconds */

static int SOS_REPEAT=100;
static sigset_t SOSset;

```

```
static sigset_t SOSoldSet;
static sigset_t SOSTestSet;

/* shortcut flag */
static int SOSshort=NONE;
static int SOSshortVar=NONE;
static int SOSshortLevel=0;

/* timer */
static long sosalarmcount;

/*
#define TIMER ITIMER_REAL
#define ALARM SIGALRM
*/
#define TIMER ITIMER_VIRTUAL
#define ALARM SIGVTALRM

#include "sosdefs.h"
```

TSP.h

/*

```
AAAA  CCCC  OOOO  TTTTT  SSSSS  PPPPP
AA AA  CC   OO  OO  TT   SS   PP  PP
AAAAAA  CC   OO  OO  TT   SSSS  PPPPP
AA AA  CC   OO  OO  TT   SS   PP
AA AA  CCCC  OOOO  TT   SSSSS  PP
```

```
#####
#####      ACO algorithms for the TSP      #####
#####
```

```
Version: 2.0
File:    TSP.h
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0
File:    TSP.h
Author:  Thomas Stuetzle
Purpose: TSP related procedures, distance computation, neighbour lists
Check:   README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle
```

*/

/*****

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

```
*****/

#define RRR          6378.388
#ifndef PI           /* as in stroustrup */
#define PI          3.14159265358979323846
#endif

struct point {
    double x;
    double y;
};

struct problem{
    char      name[LINE_BUF_LEN];          /* instance name */
    char      edge_weight_type[LINE_BUF_LEN]; /* selfexplanatory */
    long int  optimum;                     /* optimal tour length if known,
otherwise a bound */
    long int  n;                           /* number of cities */
    long int  n_near;                       /* number of nearest neighbors */
    struct point *nodeptr;                 /* array of structs containing
coordinates of nodes */
    long int  **distance;                  /* distance matrix: distance[i][j] gives
distance
between city i und j */
    long int  **nn_list;                   /* nearest neighbor list; contains for
each node i a
sorted list of n_near nearest
neighbors */
};

extern struct problem instance;

long int n;                               /* number of cities in the instance to be solved */

long int (*distance)(long int, long int); /* pointer to function returning
distance */

long int round_distance (long int i, long int j);

long int ceil_distance (long int i, long int j);

long int geo_distance (long int i, long int j);

long int att_distance (long int i, long int j);

long int compute_tour_length( long int *t );

long int **compute_distances(void);

long int ** compute_nn_lists ( void );
```

TSP.c

/*

```

AAAA   CCCC   OOOO   TTTTTT   SSSSS   PPPPP
AA  AA  CC    OO  OO   TT    SS    PP  PP
AAAAAA  CC    OO  OO   TT    SSSS   PPPPP
AA  AA  CC    OO  OO   TT    SS    PP
AA  AA  CCCC   OOOO   TT    SSSS   PP

```

```

#####
#####      ACO algorithms for the TSP      #####
#####

```

```

Version: 2.0
File:    TSP.c
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza

```

```

Version: 1.0
File:    TSP.c
Author:  Thomas Stuetzle
Purpose: TSP related procedures, distance computation, neighbour lists
Check:   README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle

```

*/

/*****

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

```
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <assert.h>

#include "InOut.h"
#include "TSP.h"
#include "ants.h"
#include "ls.h"
#include "utilities.h"

#define M_PI 3.14159265358979323846264

long int n;          /* number of cities in the instance to be solved */

struct problem instance;

static double dtrunc (double x)
{
    int k;

    k = (int) x;
    x = (double) k;
    return x;
}

long int (*distance)(long int, long int); /* function pointer */

/*
    FUNCTION: the following four functions implement different ways of
              computing distances for TSPLIB instances
    INPUT:    two node indices
    OUTPUT:   distance between the two nodes
*/

long int round_distance (long int i, long int j)
/*
    FUNCTION: compute Euclidean distances between two nodes rounded to next
              integer for TSPLIB instances
    INPUT:    two node indices
    OUTPUT:   distance between the two nodes
    COMMENTS: for the definition of how to compute this distance see TSPLIB
*/
{
    double xd = instance.nodeptr[i].x - instance.nodeptr[j].x;
    double yd = instance.nodeptr[i].y - instance.nodeptr[j].y;
    double r  = sqrt(xd*xd + yd*yd) + 0.5;
}
```



```

    return (long int) r;
}

long int ceil_distance (long int i, long int j)
/*
    FUNCTION: compute ceiling distance between two nodes rounded to next
              integer for TSPLIB instances
    INPUT:    two node indices
    OUTPUT:   distance between the two nodes
    COMMENTS: for the definition of how to compute this distance see TSPLIB
*/
{
    double xd = instance.nodeptr[i].x - instance.nodeptr[j].x;
    double yd = instance.nodeptr[i].y - instance.nodeptr[j].y;
    double r = sqrt(xd*xd + yd*yd) + 0.000000001;

    return (long int)r;
}

long int geo_distance (long int i, long int j)
/*
    FUNCTION: compute geometric distance between two nodes rounded to next
              integer for TSPLIB instances
    INPUT:    two node indices
    OUTPUT:   distance between the two nodes
    COMMENTS: adapted from concorde code
              for the definition of how to compute this distance see TSPLIB
*/
{
    double deg, min;
    double lati, latj, longi, longj;
    double q1, q2, q3;
    long int dd;
    double x1 = instance.nodeptr[i].x, x2 = instance.nodeptr[j].x,
    y1 = instance.nodeptr[i].y, y2 = instance.nodeptr[j].y;

    deg = dtrunc (x1);
    min = x1 - deg;
    lati = M_PI * (deg + 5.0 * min / 3.0) / 180.0;
    deg = dtrunc (x2);
    min = x2 - deg;
    latj = M_PI * (deg + 5.0 * min / 3.0) / 180.0;

    deg = dtrunc (y1);
    min = y1 - deg;
    longi = M_PI * (deg + 5.0 * min / 3.0) / 180.0;
    deg = dtrunc (y2);
    min = y2 - deg;
    longj = M_PI * (deg + 5.0 * min / 3.0) / 180.0;

    q1 = cos (longi - longj);
    q2 = cos (lati - latj);
    q3 = cos (lati + latj);
    dd = (int) (6378.388 * acos (0.5 * ((1.0 + q1) * q2 - (1.0 - q1) * q3)) +
1.0);
}

```

```

    return dd;
}

long int att_distance (long int i, long int j)
/*
    FUNCTION: compute ATT distance between two nodes rounded to next
              integer for TSPLIB instances
    INPUT:    two node indices
    OUTPUT:   distance between the two nodes
    COMMENTS: for the definition of how to compute this distance see TSPLIB
*/
{
    double xd = instance.nodeptr[i].x - instance.nodeptr[j].x;
    double yd = instance.nodeptr[i].y - instance.nodeptr[j].y;
    double rij = sqrt ((xd * xd + yd * yd) / 10.0);
    double tij = dtrunc (rij);
    long int dij;

    if (tij < rij)
        dij = (int) tij + 1;
    else
        dij = (int) tij;
    return dij;
}

long int ** compute_distances(void)
/*
    FUNCTION: computes the matrix of all intercity distances
    INPUT:    none
    OUTPUT:   pointer to distance matrix, has to be freed when program stops
*/
{
    long int    i, j;
    long int    **matrix;

    if((matrix = malloc(sizeof(long int) * n * n +
        sizeof(long int *) * n )) == NULL){
        fprintf(stderr, "Out of memory, exit.");
        exit(1);
    }
    for ( i = 0 ; i < n ; i++ ) {
        matrix[i] = (long int *) (matrix + n) + i*n;
        for ( j = 0 ; j < n ; j++ ) {
            matrix[i][j] = distance(i, j);
        }
    }
    return matrix;
}

long int ** compute_nn_lists( void )
/*
    FUNCTION: computes nearest neighbor lists of depth nn for each city
    INPUT:    none
    OUTPUT:   pointer to the nearest neighbor lists
*/

```

```

*/
{
    long int i, node, nn;
    long int *distance_vector;
    long int *help_vector;
    long int **m_nnear;

/*    TRACE ( printf("\n computing nearest neighbor lists, "); ) */

    nn = MAX(nn_ls, nn_ants);
    if ( nn >= n )
nn = n - 1;
    DEBUG ( assert( n > nn ); )

/*    TRACE ( printf("nn = %ld ... \n", nn); ) */

    if((m_nnear = malloc(sizeof(long int) * n * nn
        + n * sizeof(long int *))) == NULL){
exit(EXIT_FAILURE);
    }
    distance_vector = calloc(n, sizeof(long int));
    help_vector = calloc(n, sizeof(long int));

    for ( node = 0 ; node < n ; node++ ) { /* compute cnd-sets for all node */
m_nnear[node] = (long int *) (m_nnear + n) + node * nn;

    for ( i = 0 ; i < n ; i++ ) { /* Copy distances from nodes to the others */
        distance_vector[i] = instance.distance[node][i];
        help_vector[i] = i;
    }
    distance_vector[node] = LONG_MAX; /* city is not nearest neighbour */
    sort2(distance_vector, help_vector, 0, n-1);
    for ( i = 0 ; i < nn ; i++ ) {
        m_nnear[node][i] = help_vector[i];
    }
    }
    free(distance_vector);
    free(help_vector);
/*    TRACE ( printf("\n    .. done\n"); ) */
    return m_nnear;
}

long int compute_tour_length( long int *t )
/*
    FUNCTION: compute the tour length of tour t
    INPUT:    pointer to tour t
    OUTPUT:   tour length of tour t
*/
{
    int i;
    long int tour_length = 0;

    for ( i = 0 ; i < n ; i++ ) {
tour_length += instance.distance[t[i]][t[i+1]];
    }
}

```

```
    return tour_length;  
}
```

unix_timer.h

/*

```
AAAA  CCCC  OOOO  TTTTT  SSSS  PPPP
AA AA  CC   OO  OO  TT   SS   PP  PP
AAAAAA  CC   OO  OO  TT   SSSS  PPPP
AA AA  CC   OO  OO  TT   SS   PP
AA AA  CCCC  OOOO  TT   SSSS  PP
```

```
#####
#####      ACO algorithms for the TSP      #####
#####
```

```
Version: 2.0
File:    times.h
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0
File:    times.h
Author:  Thomas Stuetzle
Purpose: routines for measuring elapsed time (CPU or real)
Check:   README.txt and legal.txt
```

*/

```
int time_expired();

void start_timers();

double elapsed_time();

typedef enum type_timer {REAL, VIRTUAL} TIMER_TYPE;
```

unix_timer.c

/*

```
AAAA  CCCC  OOOO  TTTTT  SSSSS  PPPPP
AA AA  CC   OO  OO  TT   SS   PP  PP
AAAAAA  CC   OO  OO  TT   SSSS  PPPPP
AA AA  CC   OO  OO  TT   SS   PP
AA AA  CCCC  OOOO  TT   SSSSS  PP
```

```
#####
#####      ACO algorithms for the TSP      #####
#####
```

```
Version: 2.0
File:    times.c
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0
File:    times.c
Author:  Thomas Stuetzle
Purpose: routines for measuring elapsed time (CPU or real)
Check:   README.txt and legal.txt
```

*/

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/resource.h>
```

```
#include "timer.h"
```

```
static struct rusage res;
static struct timeval tp;
static double virtual_time, real_time;
```

```
void start_timers()
```

/*

```
FUNCTION:      virtual and real time of day are computed and stored to
                allow at later time the computation of the elapsed time
                (virtual or real)
INPUT:         none
OUTPUT:        none
(SIDE)EFFECTS: virtual and real time are computed
```

*/

{

```
getrusage( RUSAGE_SELF, &res );
virtual_time = (double) res.ru_utime.tv_sec +
               (double) res.ru_stime.tv_sec +
               (double) res.ru_utime.tv_usec / 1000000.0 +
               (double) res.ru_stime.tv_usec / 1000000.0;
```

```
gettimeofday( &tp, NULL );
```

```

    real_time =    (double) tp.tv_sec +
                  (double) tp.tv_usec / 1000000.0;
}

double elapsed_time( type )
    TIMER_TYPE type;
/*
    FUNCTION:      return the time used in seconds (virtual or real,
depending on type)
    INPUT:         TIMER_TYPE (virtual or real time)
    OUTPUT:        seconds since last call to start_timers (virtual or real)
    (SIDE)EFFECTS: none
*/
{
    if (type == REAL) {
        gettimeofday( &tp, NULL );
        return( (double) tp.tv_sec +
                (double) tp.tv_usec / 1000000.0
                - real_time );
    }
    else {
        getrusage( RUSAGE_SELF, &res );
        return( (double) res.ru_utime.tv_sec +
                (double) res.ru_stime.tv_sec +
                (double) res.ru_utime.tv_usec / 1000000.0 +
                (double) res.ru_stime.tv_usec / 1000000.0
                - virtual_time );
    }
}

```

utilities.h

/*

```
AAAA   CCCC   OOOO   TTTTT   SSSSS   PPPPP
AA AA  CC    OO OO   TT    SS     PP  PP
AAAAAA  CC    OO OO   TT    SSSS   PPPPP
AA AA  CC    OO OO   TT     SS    PP
AA AA  CCCC   OOOO   TT    SSSSS   PP
```

```
#####
#####      ACO algorithms for the TSP      #####
#####
```

```
Version: 2.0
File:    utilities.h
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0
File:    utilities.h
Author:  Thomas Stuetzle
Purpose: some additional useful procedures
Check:   README.txt and legal.txt
Copyright (C) 2002 Thomas Stuetzle
```

*/

/*****

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

*****/

```
#define INFTY          LONG_MAX
#define MAXIMUM_NO_TRIES 100
```

```
#define TRUE 1
#define FALSE 0
```

/ general macros */*

```
#define MAX(x,y)      ((x)>=(y)?(x):(y))
#define MIN(x,y)      ((x)<=(y)?(x):(y))
```

```
#define DEBUG( x )
```

```
#define TRACE( x ) x
```

/ constants for a random number generator, for details see numerical recipes in C */*

```
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define MASK 123459876
```

```
extern long int seed;
double mean ( long int *values, long int max);
double meanr ( double *values, long int max );
double std_deviation ( long int *values, long int i, double mean );
double std_deviationr ( double *values, long int i, double mean );
long int best_of_vector ( long int *values, long int i );
long int worst_of_vector ( long int *values, long int i );
void swap ( long int v[], long int i, long int j );
void sort ( long int v[], long int left, long int right );
double quantil ( long int vector[], double q, long int numbers );
void swap2(long int v[], long int v2[], long int i, long int j);
void sort2(long int v[], long int v2[], long int left, long int right);
double ran01 ( long *idum );
long int random_number ( long *idum );
long int ** generate_int_matrix( long int n, long int m);
double ** generate_double_matrix( long int n, long int m);
```

utilities.c

/*

```
AAAA  CCCC  OOOO  TTTTT  SSSSS  PPPPP
AA AA  CC   OO  OO  TT   SS   PP  PP
AAAAAA  CC   OO  OO  TT   SSSS  PPPPP
AA AA  CC   OO  OO  TT   SS   PP
AA AA  CCCC  OOOO  TT   SSSS  PP
```

```
#####
#####      ACO algorithms for the TSP      #####
#####
```

```
Version: 2.0
File:    utilities.c
Author:  Sammy D'Souza
Purpose: modifications for PC
Check:   README and gpl.txt
Copyright (C) 2007 Sammy D'Souza
```

```
Version: 1.0
File:    utilities.c
Author:  Thomas Stuetzle
Purpose: some additional useful procedures
Check:   README and gpl.txt
Copyright (C) 2002 Thomas Stuetzle
```

*/

/*****

Program's name: acotsp

Ant Colony Optimization algorithms (AS, ACS, EAS, RAS, MMAS, BWAS) for the symmetric TSP

Copyright (C) 2004 Thomas Stuetzle

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

email: stuetzle no@spam informatik.tu-darmstadt.de
mail address: Universitaet Darmstadt
Fachbereich Informatik

Hochschulstr. 10
D-64283 Darmstadt

Germany

```
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

/* #define USE_MPI */
/* #define SIMPLE_SPRNG */

#include "InOut.h"
#include "utilities.h"
#include "TSP.h"
#include "ants.h"
#include "timer.h"

/* #include "/home0/sdsouza/Nirvana/sprng/sprng2.0/include/sprng.h" */

long int seed = 12345678;

double mean( long int *values, long int max )
/*
    FUNCTION:      compute the average value of an integer array of length
max
    INPUT:        pointer to array, length of array
    OUTPUT:       average
    (SIDE)EFFECTS: none
*/
{
    long int j;
    double m;

    m = 0.;
    for ( j = 0 ; j < max ; j++ ) {
        m += (double)values[j];
    }
    m = m / (double)max;
    return m;
}

double meanr( double *values, long int max )
/*
    FUNCTION:      compute the average value of a floating number array of
length max
    INPUT:        pointer to array, length of array
    OUTPUT:       average
    (SIDE)EFFECTS: none
*/
{
    long int j;
    double m;
```

```

    m = 0.;
    for ( j = 0 ; j < max ; j++ ) {
        m += values[j];
    }
    m = m / (double)max;
    return m;
}

double std_deviation( long int *values, long int max, double mean )
/*
    FUNCTION:      compute the standard deviation of an integer array
    INPUT:         pointer to array, length of array, mean
    OUTPUT:        standard deviation
    (SIDE)EFFECTS: none
*/
{
    long int j;
    double dev = 0.;

    if (max <= 1)
        return 0.;
    for ( j = 0 ; j < max; j++ ) {
        dev += ((double)values[j] - mean) * ((double)values[j] - mean);
    }
    return sqrt(dev/((double)(max - 1)));
}

double std_deviationr( double *values, long int max, double mean )
/*
    FUNCTION:      compute the standard deviation of a floating number array
    INPUT:         pointer to array, length of array, mean
    OUTPUT:        standard deviation
    (SIDE)EFFECTS: none
*/
{
    long int j;
    double dev;

    if (max <= 1)
        return 0.;
    dev = 0.;
    for ( j = 0 ; j < max ; j++ ) {
        dev += ((double)values[j] - mean) * ((double)values[j] - mean);
    }
    return sqrt(dev/((double)(max - 1)));
}

long int best_of_vector( long int *values, long int l )
/*
    FUNCTION:      return the minimum value in an integer value
    INPUT:         pointer to array, length of array
    OUTPUT:        smallest number in the array
    (SIDE)EFFECTS: none
*/

```

```

{
    long int min, k;

    k = 0;
    min = values[k];
    for( k = 1 ; k < l ; k++ ) {
        if( values[k] < min ) {
            min = values[k];
        }
    }
    return min;
}

long int worst_of_vector( long int *values, long int l )
/*
    FUNCTION:      return the maximum value in an integer value
    INPUT:         pointer to array, length of array
    OUTPUT:        largest number in the array
    (SIDE)EFFECTS: none
*/
{
    long int max, k;

    k = 0;
    max = values[k];
    for( k = 1 ; k < l ; k++ ) {
        if( values[k] > max ) {
            max = values[k];
        }
    }
    return max;
}

double quantil(long int v[], double q, long int l)
/*
    FUNCTION:      return the q-quantil of an ordered integer array
    INPUT:         one array, desired quantil q, length of array
    OUTPUT:        q-quantil of array
    (SIDE)EFFECTS: none
*/
{
    long int i, j;
    double tmp;

    tmp = q * (double)l;
    if ((double)((long int)tmp) == tmp) {
        i = (long int)tmp;
        j = (long int)(tmp + 1.);
        return ((double)v[i-1] + (double)v[j-1]) / 2.;
    } else {
        i = (long int)(tmp + 1.);
        return v[i-1];
    }
}

```

```

void swap(long int v[], long int i, long int j)
/*
    FUNCTION:      auxiliary routine for sorting an integer array
    INPUT:         array, two indices
    OUTPUT:        none
    (SIDE)EFFECTS: elements at position i and j of array are swapped
*/
{
    long int tmp;

    tmp = v[i];
    v[i] = v[j];
    v[j] = tmp;
}

void sort(long int v[], long int left, long int right)
/*
    FUNCTION:      recursive routine (quicksort) for sorting an array
    INPUT:         one array, two indices
    OUTPUT:        none
    (SIDE)EFFECTS: elements at position i and j of the two arrays are swapped
*/
{
    long int k, last;

    if (left >= right)
        return;
    swap(v, left, (left + right)/2);
    last = left;
    for (k=left+1; k <= right; k++)
        if (v[k] < v[left])
            swap(v, ++last, k);
    swap(v, left, last);
    sort(v, left, last);
    sort(v, last+1, right);
}

void swap2(long int v[], long int v2[], long int i, long int j)
/*
    FUNCTION:      auxiliary routine for sorting an integer array
    INPUT:         two arrays, two indices
    OUTPUT:        none
    (SIDE)EFFECTS: elements at position i and j of the two arrays are swapped
*/
{
    long int tmp;

    tmp = v[i];
    v[i] = v[j];
    v[j] = tmp;
    tmp = v2[i];
    v2[i] = v2[j];
    v2[j] = tmp;
}

```

```

void sort2(long int v[], long int v2[], long int left, long int right)
/*
    FUNCTION:      recursive routine (quicksort) for sorting one array;
second
                  arrays does the same sequence of swaps
    INPUT:        two arrays, two indices
    OUTPUT:       none
    (SIDE)EFFECTS: elements at position i and j of the two arrays are swapped
*/
{
    long int k, last;

    if (left >= right)
        return;
    swap2(v, v2, left, (left + right)/2);
    last = left;
    for (k=left+1; k <= right; k++)
        if (v[k] < v[left])
            swap2(v, v2, ++last, k);
    swap2(v, v2, left, last);
    sort2(v, v2, left, last);
    sort2(v, v2, last+1, right);
}

/*
double ran01( long *idum )

    FUNCTION:      generate a random number that is uniformly distributed in
[0,1]
    INPUT:        pointer to variable with the current seed
    OUTPUT:       random number uniformly distributed in [0,1]
    (SIDE)EFFECTS: random number seed is modified (important, this has to be
done!)
    ORIGIN:       numerical recipes in C
*/
double ran01( long *idum )
{
    long k;
    double ans;

/*
    ans = sprng();
    TRACE( printf("%lf \t", ans); );
    return ans;
*/

    k =(*idum)/IQ;
    *idum = IA * (*idum - k * IQ) - IR * k;
    if (*idum < 0 ) *idum += IM;
    ans = AM * (*idum);
    return ans;
}

```

```

/*
double ran01( long * idum)
{
    return sprng();
}
*/

long int random_number( long *idum )
/*
    FUNCTION:      generate an integer random number
    INPUT:         pointer to variable containing random number seed
    OUTPUT:        integer random number uniformly distributed in
{0,2147483647}
    (SIDE)EFFECTS: random number seed is modified (important, has to be
done!)
    ORIGIN:        numerical recipes in C
*/
{
    long k;

    k =(*idum)/IQ;
    *idum = IA * (*idum - k * IQ) - IR * k;
    if (*idum < 0 ) *idum += IM;
    return *idum;
}

long int ** generate_int_matrix( long int n, long int m)
/*
    FUNCTION:      malloc a matrix and return pointer to it
    INPUT:         size of matrix as n x m
    OUTPUT:        pointer to matrix
    (SIDE)EFFECTS:
*/
{
    long int i;
    long int **matrix;

    if((matrix = malloc(sizeof(long int) * n * m +
        sizeof(long int *) * n )) == NULL){
        printf("Out of memory, exit.");
        exit(1);
    }
    for ( i = 0 ; i < n ; i++ ) {
        matrix[i] = (long int *) (matrix + n) + i*m;
    }

    return matrix;
}

double ** generate_double_matrix( long int n, long int m)
/*
    FUNCTION:      malloc a matrix and return pointer to it
    INPUT:         size of matrix as n x m
    OUTPUT:        pointer to matrix
    (SIDE)EFFECTS:
*/

```



```
*/
{
    long int i;
    double **matrix;

    if((matrix = malloc(sizeof(double) * n * m +
        sizeof(double *) * n )) == NULL){
        printf("Out of memory, exit.");
        exit(1);
    }
    for ( i = 0 ; i < n ; i++ ) {
        matrix[i] = (double *) (matrix + n) + i*m;
    }
    return matrix;
}
```

APPENDIX D
GLOSSARY OF TERMS

Ant Colony Optimization (ACO) A metaheuristic invented by Marco Dorigo and inspired by the foraging behavior of real ants. In ACO, a number of artificial ants build solutions to an optimization problem and exchange information on their quality via a communication scheme that is reminiscent of the one adopted by real ants.

Ant System (AS) The original algorithm in the ACO class. AS was inspired by the foraging behavior of ants in finding shortest paths to food sources, and was used to show probabilistic techniques for solving the TSP. The main characteristic was that pheromone values are updated by all the ants that have completed the tour.

Combinatorial Optimization (CO) A field of applied mathematics that aims to solve optimization problems that are believed to be hard in general, using techniques from combinatorics, linear programming, and the theory of algorithms. A typical CO problem involves finding values for discrete variables such that the optimal solution for a given objective function is found.

Distributed Computing A system of computation in which multiple computers or processors collaborate over a network to solve a common problem. The computers themselves are independent and may be heterogeneous in many aspects, however, this aggregation is transparent.

Evolutionary Computation (EC) Problem solving and computation techniques based in some form or other on the evolution of biological species in the natural world. Three large classes within EC are evolution strategies, evolutionary programming and genetic algorithms.

Metaheuristic A set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. In other words, a

metaheuristic is a general-purpose algorithmic framework that can be applied to different optimization problems with relatively few modifications. Simulated Annealing, Tabu Search, Iterated Local Search and Ant Colony Optimization are examples of metaheuristics.

MAX-MIN Ant System (MMAS) An improvement over the original ant system proposed by Sttzle and Hoos (2000). The most important difference in MMAS was that the pheromone value was bounded between a maximum and a minimum value and only the best ant updated the pheromone trails.

NP-hard An NP (nondeterministic, polynomial time) problem is one that can be solved in polynomial time by a nondeterministic Turing Machine. If the algorithm for solving it can be translated into one for solving any other NP problem, then we say the problem is NP-hard.

Overlapping A parallel communication protocol that tries to minimize the synchronization delay between data generation at one process and its consumption by another. Overlapping achieves this by scheduling communications dynamically and guaranteeing correct placement of these calls, blocking if necessary.

Shortcutting A parallel computing technique that exploits time irregularity and non-determinism. A computation may terminate faster on one node than the others; this node can then effectively shortcut the other nodes.

Single Program Multiple Data (SPMD) A parallel computing approach in which a single program executes differently on different processors. This is done either by referring to the processor *number* explicitly or through implicit mechanisms such as compiler directives.

Stigmergy Originally used to denote the class of mechanisms that mediate animal-animal interactions. Stigmergy now refers to a method of indirect communication

that takes place by modifying the environment.

Streams A set of programs executing the same control path. A parallel execution will always start and end as a single stream, but may split into different streams and possibly merge along the way depending on program logic.

Streams, Overlapping and Shortcutting (SOS) A library built on MPI and callable from Fortran or C that allows a programmer to incorporate Streams, Overlapping communications, and Shortcutting into a parallel program.

Traveling Salesman Problem (TSP) An NP-hard optimization problem in computer science that involves finding the minimum length Hamiltonian circuit of a complete weighted graph. In lay terms, the problem involves a salesman who wants to start from home, take a shortest tour through a set of cities, visit each city exactly once and return home. In a Symmetric TSP, the distance from node j to i is the same as that from node i to j .

TSPLIB A library of sample instances for the TSP maintained by Professor Gerhard Reinelt at the University of Heidelberg, Germany. The TSPLIB also contains problem instances for other problems, such as the Sequential Ordering Problem (SOP) and the Capacitated Vehicle Routing Problem (CVRP) and optimal solutions and values for some of these.

REFERENCES

- [1] Prasanna Balaprakash. Ant Colony Optimization under Uncertainty. Technical Report TR/IRIDIA/2005-028, Universit Libre de Bruxelles, 2005.
- [2] M. Birattari, G. Di Caro, and Marco Dorigo. For a Formal Foundation of the Ant Programming Approach to Combinatorial Optimization. Technical Report TR-H-301, ATR Human Information Processing Research Laboratories, Kyoto, Japan, 2000.
- [3] Soumen Chakrabarti, James Demmel, and Katherine A. Yelick. Modeling the Benefits of Mixed Data and Task Parallelism. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 74–83, 1995.
- [4] David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauer, Ramesh Subramonian, and Thorsten von Eicken. Logp: a practical model of parallel computation. *Commun. ACM*, 39(11):78–85, 1996.
- [5] M. Dorigo and L.M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):73–81, 1997.
- [6] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.
- [7] M. Dorigo and L.M. Gambardella. Ant Algorithms for Discrete Optimization.

- Technical Report IRIDIA/98-10, IRIDIA, Universite Libre de Bruxelles, Brussels, Belgium, 1998.
- [8] M. Dorigo, V. Maniezzo, and A. Coloni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 26(1):29–41, 1996.
- [9] Marco Dorigo and Thomas Stutzle. *Ant Colony Optimization*. MIT Press, 2004.
- [10] John R. Ellis. *Bulldog – A Compiler for VLIW Architectures*. MIT Press, 1986.
- [11] L.M. Gambardella and M. Dorigo. Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. In *Proc. Twelfth International Conference on Machine Learning, ML-95*, pages 252–260. Morgan Kaufmann Publishers, 1995.
- [12] Ernesto Gomez. *PC Reference Manual*. Available with the PC distribution, <http://csci.csusb.edu/egomez/cs624.html>.
- [13] Ernesto Gomez. *The SOS library*. <http://csci.csusb.edu/egomez/html-res/sos.html>.
- [14] Ernesto Gomez. *Single Program Task Parallelism*. PhD thesis, University of Chicago, March 2005.
- [15] Ernesto Gomez and L. Ridgway Scott. Overlapping and Shortcutting Techniques in Loosely Synchronous Irregular Problems. *Solving Irregularly Structured Problems in Parallel*, LNCS 1457:116–127, August 1998.
- [16] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing*. Addison-Wesley, 2nd edition, 2004.
- [17] Steven Homer and Marcus Peinado. Design and Performance of Parallel and Distributed Approximation Algorithms for Maxcut. *Journal of Parallel and Distributed Computing*, 46(1):48–61, 1997.

- [18] L. Ridgway Scott, Terry W. Clark, and Babak Bagheri. PFortran - a Parallel Extension of Fortran (the PFortran Reference Manual). Research Report UH/MD 124, Department of Mathematics, University of Houston, 1992.
- [19] L. Ridgway Scott, Terry W. Clark, and Babak Bagheri. *Pfortran: A Parallel Dialect of Fortran*. Fortran Forum. ACM Press, 1992.
- [20] L. Ridgway Scott, Terry W. Clark, and Babak Bagheri. *Scientific Parallel Computation*. MIT Press, 2004.
- [21] David Brian Sturgill. *Nagging: a general, fault-tolerant approach to parallel search pruning*. PhD thesis, Cornell University, Ithaca, NY, USA, 1997.
- [22] T. Stutzle, M. Dorigo, and M. Birattari. Ant colony optimization— artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, November 2006.
- [23] T. Stutzle and H.H. Hoos. MAX MIN Ant System. *Journal of Future Generation Computer Systems*, 16:889–914, 2000.
- [24] Thomas Stutzle. Parallelization strategies for ant colony optimization. *Parallel Problem Solving from Nature - PPSN V*, LNCS 1498:722–731, 1998.