Abstract of "Simulated Annealing Based Local Search for Sport Scheduling Problems" by Ioannis Vergados, Ph.D., Brown University, May 2007.

Sport Scheduling is an important area of Combinatorial Optimization of great practical and theoretical significance, that features some extremely challenging optimization problems. This thesis focuses on two important sport scheduling problems: the Break Minimization Problem (which arises in scheduling European soccer leagues) and the Traveling Tournament Problem (TTP), proposed by Easton, Nemhauser, and Trick (2001), which captures the essence of scheduling Major League Baseball in the United States. For the TTP, no exact solution has been found for most benchmarks, even by employing state-of-the-art combinatorial optimization tools. In this thesis, we develop two sophisticated simulated annealing schemes, BMSA and TTSA, and successfully apply them to Break Minimization and TTP, respectively, improving the best-known solutions to most benchmarks proposed in the literature. Then we embed TTSA in an original population-based framework featuring *diversification* and *intensification*, to significantly improve most TTSA results. The main contributions of this thesis are the following: we show that, contrary to common belief, Local Search *is* an effective method for Sport Scheduling; we design a sophisticated neighborhood that captures the problem's special structure; we successfully adapt tabu search ideas such as strategic oscillation, intensification and diversification into simulated annealing; and, we develop advanced local search schemes with original modelings and sophisticated metaheuristics, that produce the best currently known solutions on most benchmarks.

Simulated Annealing Based Local Search for Sport Scheduling Problems

by
Ioannis Vergados
Diploma, Computer Engineering and Informatics Department, University of Patras, Greece, 1999
Sc. M., Computer Engineering and Informatics Department, University of Patras, Greece, 2001
Sc. M., Brown University, Providence, RI, USA, 2002

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in the
Department of Computer Science at Brown University

Providence, Rhode Island
May 2007

This dissertation by Ioannis Vergados is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Pascal Van Hentenryck, Director

Recommended to the Graduate Council

Date _____ _____

Meinolf Sellmann, Reader

Date _____ _____

Michael Trick, Reader
(Tepper School of Business, Carnegie Mellon University)

Approved by the Graduate Council

Date _____ _____

Sheila Bonde
Dean of the Graduate School

Vita

Ioannis (Yannis) Vergados was born in Patras, Greece, on April 9th, 1976. After graduating from the 9th Lyceum of Patras in June 1994, he entered the Computer Engineering and Informatics Department of the Polytechnic School of the University of Patras. He received his Diploma from the Computer Engineering and Informatics Department in September 1999, with a GPA of 8.8/10, and a Sc.M. from the same department in 2001, under the supervision of Professor Christos Kaklamanis.

In September 2000, he got accepted into the Ph.D. program of the Computer Science Department of Brown University, from which he received a Sc.M. in Computer Science in 2002, under the supervision of Professor Pascal Van Hentenryck. While at Brown University, he served as a Teaching Assistant for CS181 (Computational Molecular Biology), CS157 (Design and Analysis of Algorithms), CS196 (Introduction to Combinatorial Optimization), and CS258 (Solving Hard Problems in Combinatorial Optimization). He was a recipient of the Paris Kanellakis Fellowship in 2004. His publications include:

C. Bartzis, I. Caragiannis, C. Kaklamanis, and I. Vergados. "Experimental Evaluation of Hot-Potato Routing Algorithms on 2-Dimensional Processor Arrays (Research Note)." In *Proceedings of EURO-PAR'00*, pp. 877–881, Munich, Germany, 2000. LNCS 1900, Springer-Verlag, 2000.

I. Caragiannis, C. Kaklamanis, and I. Vergados. "Greedy Dynamic Hot-Potato Routing on Arrays." In *Proceedings of ISPAN'00*, pp. 178–185, Dallas, TX, USA, 2000. IEEE Computer Society, 2000.

A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. "A Simulated Annealing Approach to the Traveling Tournament Problem." In *CP-AI-OR'03*, Montréal, Canada, 2003.

P. Van Hentenryck and Y. Vergados. "Minimizing Breaks in Sport Scheduling with Local Search." In *Proceedings of ICAPS'05*, pp. 22–29, Monterey, CA, USA, 2005.

P. Van Hentenryck and Y. Vergados. "Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealling." In *Proceedings of CP-AI-OR'06*, pp. 228–243, Cork, Ireland, 2006. LNCS 3990, Springer-Verlag, 2006.

P. Van Hentenryck, R. Bent, and Y. Vergados. "Online Stochastic Reservation Systems." In *Proceedings of CP-AI-OR'06*, pp. 212–227, Cork, Ireland, 2006. LNCS 3990, Springer-Verlag, 2006.

A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. "A Simulated Annealing Approach to the Traveling Tournament Problem." In *Journal of Scheduling*, 9(2):177–193, 2006.

Acknowledgments

In any work of this scale, there are always many people I would like to thank, so I hope to succeed in paying tribute to everybody. First of all, I don't think that it would have been possible to reach this happy ending of my long adventure in the Brown Computer Science Department, without the support of my advisor, Professor Pascal Van Hentenryck. I feel really honored to have collaborated with Pascal, for multiple reasons. It was he who believed in me and supported me at difficult moments. Most importantly, he gave me the freedom to pursue my own research interests, and he was always available to offer help and guidance.

I was really happy to have Professor Meinolf Sellmann in my committee. Meinolf's office door was always open, and he was always willing to discuss with me, both on strictly technical matters and on more general topics, often offering me enlightening views.

Professor Michael Trick only joined my committee about two years ago; however, his involvement in my thesis has been a great honor to me. With his deep knowledge of Optimization and Operations Research, and his pleasant personality, it has been a pleasure to collaborate with him. I wish to thank Mike for helping me significantly broaden my understanding in the area of Sport Scheduling.

In the course of the Ph.D. program, it was invaluable to be able to interact so freely with the Brown CS Professors. Professors in our department combine impressive academic credentials with an excellent sense of all other aspects of life, constantly demonstrating that research lies not so much in isolation as in active involvement. In this respect, I want to thank all the Computer Science faculty. However, I wish to make a special mention to the following professors, considering myself lucky for getting the opportunity to meet and work with them: Professor Philip Klein, who was my advisor for two years, and significantly contributed to my research development; Professor Eli Upfal, whose involvement with the CS optimization group made its meetings really exciting and productive; and, finally, Professors Franco Preparata and John Savage, who are not only outstanding scientists and teachers, but, more importantly, they are ideal role models for any young scientist.

I also wish to thank General and Mrs. Kanellakis for their great support and encouragement, at a very critical point of my studies at Brown.

With regards to my thesis research, big thanks go to my collaborators and co-authors, while at Brown: Aris Anagnostopoulos, Russell Bent and Laurent Michel, all of which are fun people to work with. Over the years, I got the chance to meet many prominent researchers in the area of Combinatorial Optimization, especially in the area of Sport Scheduling. I would like to cordially thank

Matthias Elf, Luca Di Gaspero, Irv Lustig, Rasmus Rasmussen, for our many fruitful discussions. I want to specially thank Matthias Elf for providing us with instances, solutions, and important ideas related to the Break Minimization problem. This work has been partially supported by NSF ITR Awards DMI-0121495 and ACI-0121497. I would also like to thank the many anonymous reviewers for their creative criticism, that contributed to improving this work.

The administrative and technical staff have been great in ensuring that everything rolls smoothly in the Department, and, for this, they deserve a big "thank you". It would have been really difficult to complete my thesis work without them.

Having reached this point of my studies, I can now appreciate the contribution of my teachers, starting from the very first grade of elementary school. Of course, this is too long a list of people to mention fully, but, to a smaller or greater degree, each of them made me realize the importance of learning and of devotion to academic pursuits, even in the face of hardships and adversities.

I would like to take this opportunity to thank my undergraduate advisor in Greece, Professor Christos Kaklamanis, for all his guidance and support in my first steps in research; Professor Ioannis Caragiannis and Dr. Evi Papaioannou, for the excellent research collaboration and, in general, for the very positive environment they maintained in our common lab with Professor Kaklamanis, during the time I was doing research for my Master's degree in the University of Patras; Professor Efstratios Gallopoulos, with whom I worked for the first time on a research project beyond regular course work, and from whom I also learned a lot; Professor Athanasios Tsakalidis, for the interesting discussions we had during my time in the University of Patras. These people taught me in many ways about research, and they also gave me invaluable help in taking the big step to pursue a Ph.D. in the United States.

Overall, studying at Brown was an amazing experience for me, especially given the many interesting people I got to meet. I would like to thank the following people, for making life at Brown and Providence enjoyable: Aris Anagnostopoulos, Yanif Ahmad, Ionuţ Aron, Don Carney, Cleopatra Christoforou, Socrates Dimitriadis, Ioanna Grypari, Paschalis Karageorgis, Yannis Katsoulis, Dimitris Kazazis, Yannis Kontoyiannis, Panayiotis Mertikopoulos, Dimitris Nikitopoulos, Olga Papaemmanouil, Manos Renieris, Stefan Roth, Nikos Triandopoulos, Yannis Tsochantaridis.

It is said that you truly appreciate real friendship, only in the light of distance. Having left from my home-country seven years ago, it is extremely meaningful to me to still be close to people, with whom we have been friends for longer than 15 years. Sofia, Monika, Mathie, Stefane, Louka, Rania, I am grateful that you have always been there, as a reference point, a shelter, a connecting line going through a carefree childhood, an enquiring adolescence, a unique college experience, and an always optimistic future...

I would also like to give a big hug to Carmen, my precious love, who brings balance and happiness to my life.

Finally, there are not enough ways for me to thank my parents, Froso and Nikos, my sister, Eva, and the rest of my family, for their endless and unconditional love and encouragement during all these years, and most importantly for the priceless moments we have spent together.

As a token of gratitude for all the sacrifices they have made and for everything they have imparted to me about life, I would like to dedicate this thesis to my beloved parents, Froso and Nikos.

Contents

List of Tables **xi**

List of Figures **xiii**

1 Related Work **3**

1.1 Local Search . . . 3

1.1.1 Local Search Algorithms . . . 3

1.1.2 Illustration . . . 5

1.1.3 Formalization . . . 7

1.1.4 Properties of Neighborhood . . . 8

1.2 Heuristics and Metaheuristics . . . 10

1.3 Heuristics . . . 11

1.3.1 Systematic Heuristics . . . 11

1.3.2 Random Walks . . . 12

1.4 Metaheuristics . . . 13

1.4.1 Iterated Local Search . . . 13

1.4.2 Simulated Annealing . . . 14

1.4.3 Variable Neighborhood Search . . . 15

1.4.4 Tabu Search . . . 16

1.5 Cooperative Parallel Search . . . 19

1.6 Sport Scheduling . . . 21

1.6.1 Break Minimization Problem . . . 21

1.6.2 Traveling Tournament Problem . . . 22

2 The Break Minimization Problem **24**

2.1 Problem Description . . . 24

2.2 Neighborhood Definition . . . 26

2.3 The BMSA Algorithm . . . 27

2.3.1 The Metropolis Heuristic . . . 27

2.3.2 The Simulated Annealing Meta-Heuristic . . . 28

2.3.3 Analogy with Statistical Physics . . . 28
2.3.4 Estimating the Statistical Measures . . . 28
2.3.5 The Initial Temperature . . . 29
2.3.6 Updating the Temperature . . . 29
2.3.7 Early Phase Termination . . . 29
2.3.8 Early Termination of the Cooling Process and Reheats . . . 30
2.3.9 The Simulated Annealing Algorithm (BMSA) . . . 30
2.4 Experimental Results with BMSA . . . 32
2.4.1 The Instances . . . 32
2.4.2 Experimental Setting for BMSA . . . 33
2.4.3 Quality of the Schedules . . . 33
2.4.4 Performance of the Algorithm . . . 34
2.4.5 Restarting . . . 35
2.4.6 Quality under Strict Time Constraints . . . 35

3 The Traveling Tournament Problem 37
3.1 The Basic TTP Problem . . . 37
3.1.1 Overall Design of Local Search . . . 38
3.1.2 Initial Solutions . . . 39
3.1.3 The Neighborhood . . . 41
3.1.4 Simulated Annealing . . . 44
3.1.5 The Objective Function . . . 45
3.1.6 Strategic Oscillation . . . 46
3.1.7 Reheats . . . 46
3.2 Experimental Results on the Basic TTP . . . 47
3.2.1 Quality and Performance of TTSA . . . 47
3.2.2 Impact of the Components . . . 50
3.2.3 Solution Quality over Time . . . 51
3.2.4 Fast Cooling . . . 51
3.2.5 Best Solutions Since the Beginning of this Research . . . 54
3.3 Comparison with Break Minimization . . . 55
3.4 Variants of the TTP . . . 55
3.4.1 Definition of TTP Variants . . . 56
3.4.2 Handling Mirroring . . . 57
3.4.3 Handling Different Distance Metrics . . . 58
3.4.4 Algorithmic Refinements . . . 59
3.5 Experimental Results on TTP Variants . . . 60
3.5.1 Mirrored Instances . . . 61
3.5.2 Non-Mirrored Instances . . . 61
3.6 Understanding the Neighborhood Structure . . . 64

3.6.1 Using No Mirrored Moves . . . 65
3.6.2 Using Only Mirrored Moves . . . 65
3.7 Time Considerations . . . 70
3.8 Lower Bounds . . . 70

4 Population-Based Simulated Annealing **73**
4.1 The PBSA Algorithm . . . 74
4.2 Experimental Results with PBSA . . . 76
4.2.1 PBSA from High-Quality Solutions . . . 76
4.2.2 PBSA from Scratch . . . 79
4.2.3 TTSA versus PBSA . . . 81
4.2.4 The Effect of Macro-Diversification . . . 81
4.3 Connections with Related Work . . . 81
4.3.1 Cooperative Parallel Search . . . 81
4.3.2 Memetic Algorithms and Scatter Search . . . 84
4.4 Summary of All Results . . . 85

Bibliography **90**

List of Tables

2.1 Quality of the Schedules: Number of Breaks in MCMP and BMSA 33
2.2 Performance Comparison: Computation Times in CPU seconds of MCMP and BMSA 34
2.3 Performance of BMSA-R: Computation Times in CPU seconds 35
2.4 Quality of BMSA with Limited CPU Time: n=24 36
2.5 Quality of BMSA when Limited CPU Time: n=26 36
2.6 Quality of BMSA when Limited CPU Time: n=28 36

3.1 Solution Quality of TTSA on the TTP 49
3.2 Computation Times of TTSA on the TTP 49
3.3 Parameter Values for the TTSA Instances 49
3.4 Impact of TTSA Components on Solution Quality (14 Teams) 50
3.5 Parameter Values for Experiments on the Impact of the Components (14 Teams) . . 50
3.6 Impact of Full Moves on the Solution Quality of TTSA (14 Teams) 51
3.7 Solution Quality of TTSA and TTSA(FC) within 2 Hours on 16 Teams 53
3.8 Parameter Values for Experiments on Fast Cooling (16 Teams) 53
3.9 Timeline of Best-Known Solutions: TTSA solutions are shown in bold face 54
3.10 TTSA Parameter Values used in our May 2004 solutions ($n = 12, 14$) 54
3.11 Solution Quality and Solution Times for NLB Distances with Mirroring 61
3.12 Solution Quality and Solution Times for Constant Distances with Mirroring 62
3.13 Solution Quality and Solution Times for Circular Distances with Mirroring 62
3.14 Solution Quality and Solution Times for NLB Distances without Mirroring 63
3.15 Solution Quality and Solution Times for Constant Distances without Mirroring . . . 63
3.16 Solution Quality and Solution Times for Circular Distances without Mirroring 64
3.17 Summary of TTSA Results for Non-Mirrored Instances Relative to LB 71
3.18 Summary of TTSA Results for Mirrored Instances Relative to LB 72

4.1 Quality and Times in Seconds of PBSA for NLB Distances 78
4.2 Quality and Times in Seconds of PBSA for Circular Distances 78
4.3 Quality and Times in Seconds of PBSA for NFL Distances 78
4.4 Quality and Times in Seconds of PBSA from Scratch for NLB Distances 80
4.5 Quality and Times in Seconds of PBSA from Scratch for Circular Distances 80

4.6 Quality and Times in Seconds of PBSA from Scratch for NFL Distances 80
4.7 Summary of All Results for Non-Mirrored Instances Relative to LB 86
4.8 Summary of All Results for Mirrored Instances Relative to LB 87

List of Figures

1.1 The Local Search Move 4
1.2 The Basic Local Search Template 4
1.3 A Graph Partition of Cost 9 6
1.4 A Move from Neighborhood N 6
1.5 A Move from Neighborhood N' 9
1.6 The Generic Local Search Template 11
1.7 The Iterated Local Search Template 14
1.8 The Simulated Annealing Template 14
1.9 The Variable Neighborhood Search Template 15
1.10 The Generic Local Search Template Revisited 16
1.11 A Simple Template for the Intensification of a Local Search 19

2.1 The Metropolis Algorithm 27
2.2 The Metropolis Algorithm Revisited 31
2.3 The Simulated Annealing Algorithm (BMSA) 32

3.1 Generation of Random Initial Schedules 40
3.2 The Basic Simulated Annealing Algorithm 45
3.3 The Simulated Annealing Algorithm (TTSA) 48
3.4 Solution Quality over Time for 12 Teams 52
3.5 Solution Quality over Time for 14 Teams 52
3.6 Solution Quality over Time for TTSA and TTSA(FC) 53
3.7 Comparison of Using No Mirrored Moves for Circular Instances 66
3.8 Comparison of Using No Mirrored Moves for NLB Instances 67
3.9 Comparison of Using Only Mirrored Moves for Circular Instances 68
3.10 Comparison of Using Only Mirrored Moves for NLB Instances 69

4.1 Illustrating PBSA with $k = 4$ 75
4.2 PBSA-P: A Phase of PBSA 77
4.3 The Population-Based Simulated Annealing Algorithm PBSA 77
4.4 Evolution of the Objective on NLB-16 79

4.5 Comparison of Min-Cost Evolution for TTSA and PBSA on NLB-16 82
4.6 The Effect of Macro-Diversification (NLB-16) 83

Introduction

Sport league scheduling [15] has become an important class of combinatorial optimization applications as it represents significant sources of revenue for television networks and generates extremely challenging optimization problems.

The Constrained Break Minimization Problem is a classical application in sport scheduling that has been widely studied (e.g., [45, 43, 48, 17]). Given a schedule for a round-robin tournament without specifications of the home/away roles, the problem consists of finding out which team plays at home (respectively away) in each of the games in order to minimize the number of breaks in the schedule. Breaks are a common quality measure for schedules and arise when a team plays two consecutive games at home or two consecutive games away. Break minimization problems arise in many sport scheduling applications, including the scheduling of soccer leagues in Europe.

Sophisticated optimization approaches have been applied to the break minimization problem, including constraint programming [43], integer programming [48] and, more recently, an elegant reduction to a cut maximization problem [17]. The recent results indicate that problems with 28 teams can be solved optimally within reasonable times (e.g., about 8 minutes), although some instances may take significantly longer (e.g., about 30 minutes). Note that most of these solutions are rather involved conceptually and employ state-of-the-art constraint or mathematical programming tools.

In 2001, Easton, Nemhauser, and Trick [15] proposed the Traveling Tournament Problem (TTP) in an attempt to abstract the salient features of Major League Baseball (MLB) in the United States. Easton et.al [15] argue that, without an approach to the TTP, it is unlikely that suitable schedules can be obtained for the MLB. The key to the MLB schedule is a conflict between minimizing travel distances and feasibility constraints on the home/away patterns. Travel distances are a major issue in MLB due to the number of teams and to the fact that teams go on "road trips" to visit several opponents before returning home. The feasibility constraints in MLB restrict the number of successive games that can be played at home or away.

The TTP is an abstraction of the MLB intended to stimulate research in sport scheduling. A solution to the TTP is a double round-robin tournament which satisfies sophisticated feasibility constraints (e.g., no more than three away games in a row) and minimizes the total travel distances of the teams. While both minimizing the total distance traveled, and satisfying the feasibility constraints, are easy problems when considered in isolation, the combination of the two (which is

captured by the TTP) makes the problem particularly difficult; even instances with as few as 8 teams are hard to solve, requiring techniques used by both constraint programming and mathematical programming communities, and no exact solution has been found for most benchmarks, even by employing advanced combinatorial optimization techniques.

The main conceptual contribution of this thesis is the following: it shows that, contrary to common belief in the Sport Scheduling community (at least until the results of this work were presented), Local Search *is*, in fact, an effective method for tackling sport scheduling problems. This is evidenced by the fact that we have been able to produce the best currently known upper bounds on many problems and instances, through schemes exhibiting robustness and reasonable speed.

An important technical contribution of the thesis, pertaining to the TTP, is the design of a sophisticated neighborhood that captures the problem's special structure and exploits the sub-structures of round-robin schedules. Using the above mentioned neighborhood, we develop an advanced simulated annealing algorithm (TTSA) for the TTP, featuring original modelings and sophisticated metaheuristics. TTSA produces matching or improving best-known solutions to most of the benchmark instances proposed in the literature. For the Break Minimization problem, we create a simpler version of TTSA, BMSA, which is adapted to the problem's simpler structure. BMSA is shown to have very competitive performance, compared to state-of-art methods used for break minimization.

We then show how we can embed TTSA in a new, more general, population-based framework (PBSA), that uses simulated annealing in an unconventional way, by introducing *diversification* and *intensification* into the search. Note that, even though TTSA has proven to be highly successful on most TTP instances, there are still some instances on which it has had limited effectiveness. PBSA successfully handles almost all of those instances, producing improving upper bounds. Moreover, it further improves many of the best-known solutions computed using TTSA, even on instances in which no improvement had been found using any other method, in more than four years.

The ideas of diversification and intensification have previously been used in the context of Tabu Search; however, to our knowledge, they have not been used effectively in Simulated Annealing. Note that, even in our simpler schemes, we introduce features from Tabu Search, such as, for example, strategic oscillation. This underlines another important technical contribution of this thesis, namely the successful adaptation of ideas from Tabu Search into Simulated Annealing.

Chapter 1

Related Work

Before proceeding to the actual problems and solutions, we give an overview of some of the ideas that will prove useful in describing our approaches in subsequent chapters of this thesis. After introducing some central ideas of Local Search, we proceed to describe some more specific techniques used in practice, including well-known heuristics and meta-heuristics. We then present a brief survey of Cooperative Parallel Search techniques related to our populated-based simulated annealing scheme, and conclude the chapter with a short account of some of the research in the area of Sport Scheduling.

This chapter is not aimed at being exhaustive; its goal is simply to set the stage for the main work of the thesis. Readers interested in a more complete coverage of particular sections are encouraged to consult the references accompanying each section.

1.1 Local Search

We first present some general concepts about Local Search, putting into context the more specific techniques that will follow later in the thesis. Local search algorithms are first described generically in terms of a small set of concepts and operations. Sections 1.3 and 1.4 show how to instantiate the generic local search algorithm to obtain a variety of well-known heuristics and meta-heuristics. Thus this presentation stresses the commonalities underlying local search algorithms, not their differences. Note that Sections 1.1 through 1.4 of this Chapter draw heavily from the introductory chapters of [52]. Readers interested in more detailed presentations of local search should consult, for instance, [2, 39].

1.1.1 Local Search Algorithms

A local search algorithm typically starts from a solution (that is, an assignment of values to its decision variables) and moves from solutions to neighboring solutions in hope of improving a function f. The function f measures the quality of solutions to the problem at hand. In satisfiability problems, it typically provides a distance from the current solution to a feasible solution. In a pure optimization

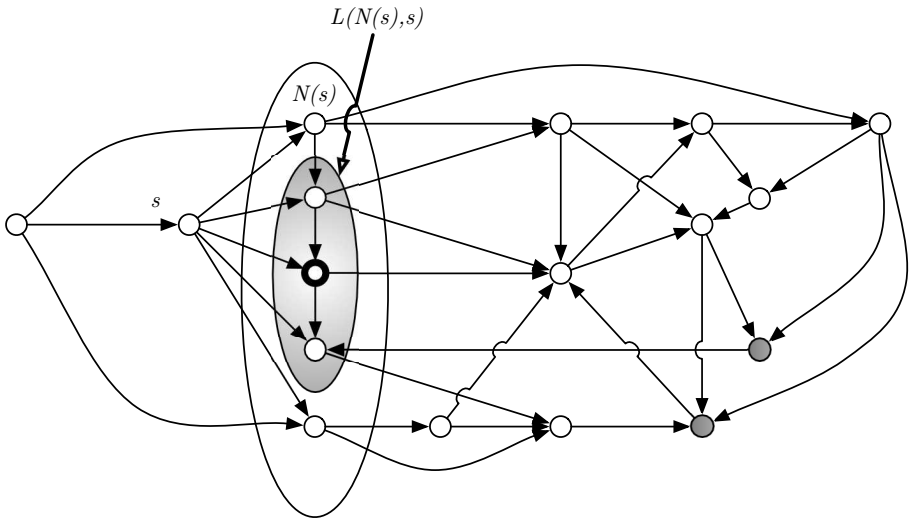


Figure 1.1: The Local Search Move

```
1.  function LocalSearch() {
2.      s := generateInitialSolution();
3.      s* := s;
4.      for k := 1 to MaxTrials do
5.          if satisfiable(s) ∧ f(s) < f(s*) then
6.              s* := s;
7.          s := S(L(N(s), s), s);
8.      return s*;
9.  }
```

Figure 1.2: The Basic Local Search Template

problem, it expresses the objective of the problem or a function of finer granularity that differentiates between solutions with the same objective value. In applications composed of both constraints and an objective, the function f may combine feasibility and optimality measures appropriate for the problem at hand. We generally assume the availability of a function f to be minimized.

The main operation of a local search algorithm amounts to moving from a solution s to one of its neighbors. The set of neighboring solutions of s, denoted by $N(s)$, is called the *neighborhood* of s. At a specific computation step, some of these neighbors may be *legal*, in which case they may be selected, or they may be *forbidden*. Once the legal neighbors are identified (by operation L), the local search selects one of them and decides whether to move to this neighbor or to stay at s (operation S). These concepts, which define the *moves* in local search, are illustrated in figure 1.1. It shows the solution s, its neighborhood $N(s)$, its set $L(N(s), s)$ of legal moves, and the selected solution (the bold thick circle).

Figure 1.2 depicts a simple generic local search template. The search starts from an initial state s (line 2) and performs a number of iterations (line 4). The move takes place in line 7 and consists

of selecting a new solution by composing operations N, L, and S:

$s := S(L(N(s), s), s)$;

Lines 5 and 6 simply keep the best solution s^* encountered so far.

Some local search algorithms feature very simple implementations of some of these operations. For instance, in some local search algorithms, all moves may be legal. In other algorithms, these operations may be rather complex and may rely on sophisticated data structures and algorithms as well as on randomization.

1.1.2 Illustration

To illustrate the local search schema, we now present a greedy local improvement algorithm in graph partitioning.

The Problem The graph-partitioning problem consists of finding a balanced partition of the vertices of a graph that minimizes the number of edges with one endpoint in each partition. More formally, a balanced partition of a graph $G = (V, E)$ is a pair $\langle P_1, P_2 \rangle$ such that $P_1 \cup P_2 = V$ and $|P_1| = |P_2|$. The cost of a partition $P = \langle P_1, P_2 \rangle$, denoted by $f(P)$, is the number of edges with one endpoint in each set:

$$f(\langle P_1, P_2 \rangle) = \#\{(v, w) \in E \mid v \in P_1 \ \& \ w \in P_2\}. \tag{1.1}$$

The set of feasible solutions, denoted by S, is the set of balanced partitions, and the set of optimal solutions, denoted by S^*, is specified as

$$\{s \in S \mid f(s) = \min_{p \in S} f(p)\}. \tag{1.2}$$

Figure 1.3 depicts a balanced partition of cost 9.

The Neighborhood The most natural move for graph partitioning consists of swapping two vertices, i.e, selecting a vertex a from P_1 and a vertex b from P_2 and assigning a to P_2 and b to P_1. Such a move is depicted in figure 1.4 and produces a partition of cost 5. More formally, the neighborhood function N is defined as

$$N(\langle P_1, P_2 \rangle) = \{\langle P_1 \setminus \{a\} \cup \{b\}, P_2 \setminus \{b\} \cup \{a\} \rangle \mid a \in P_1 \wedge b \in P_2\}. \tag{1.3}$$

The neighborhood N has a fundamental property: if the solution s is a balanced partition, all the solutions in $N(s)$ are balanced partitions as well. As a consequence, any local search starting from a balanced partition explores only balanced partitions and need not represent the balancing constraint explicitly. In other words, the local search algorithm considers only feasible solutions.

Legal Moves Local improvement algorithms require the neighbors to improve the objective value. As a consequence, the legal moves in the local improvement algorithm are specified by

$$L(N, s) = \{n \in N \mid f(n) < f(s)\}. \tag{1.4}$$

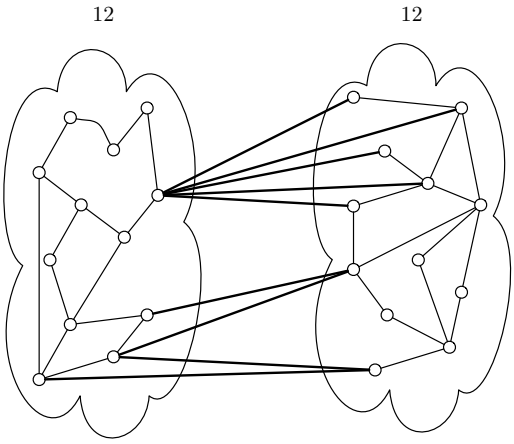


Figure 1.3: A Graph Partition of Cost 9

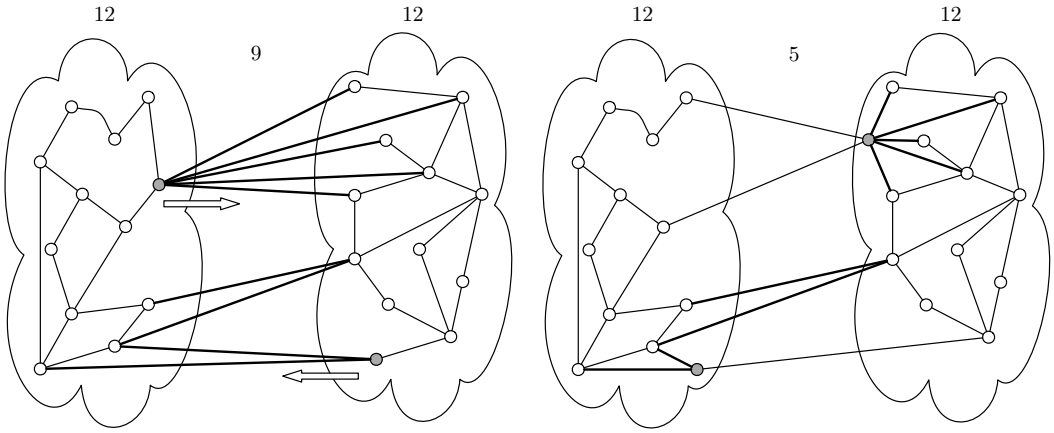


Figure 1.4: A Move from Neighborhood N

Selection It remains to specify how to choose the neighbor. Since the algorithm is greedy, the algorithm selects the best legal neighbor, that is, an element from

$$S(\mathcal{M}, s) = \{n \in \mathcal{M} \mid f(n) = \min_{s \in \mathcal{M}} f(s)\}. \tag{1.5}$$

1.1.3 Formalization

This sub-section formalizes the concepts introduced so far and defines a variety of concepts, which are referred to in later chapters. It assumes an underlying combinatorial optimization problem $\mathcal{P}$ of the form

$$\begin{array}{l} \min f(\vec{x}) \quad \text{subject to} \\ C_1(\vec{x}) \\ \vdots \\ C_n(\vec{x}) \end{array} \tag{1.6}$$

where $\vec{x}$ is a vector of n (discrete) decision variables, f is an objective function $\mathcal{N}^n \rightarrow \mathcal{N}$ that associates a performance measure with a variable assignment, and $C_1, \ldots, C_n$ are constraints defining the solution space. The concepts of solutions, feasible solutions, and optimal solutions have the following (natural) meanings.

Definition 1 *A solution to $\mathcal{P}$ is an assignment of values to the variables in $\vec{x}$. The set of solutions to $\mathcal{P}$ is denoted by $\mathcal{L}_{\mathcal{P}}$.*

Definition 2 *A feasible solution of $\mathcal{P}$ is a solution $\hat{x}$ that satisfies $C_1(\hat{x}) \wedge \ldots \wedge C_n(\hat{x})$. The set of all feasible solutions of $\mathcal{P}$ is denoted by $\tilde{\mathcal{L}_{\mathcal{P}}}$.*

Definition 3 *The set of optimal solutions to $\mathcal{P}$, denoted by ${\mathcal{L}_{\mathcal{P}}}^*$ is defined as*

$${\mathcal{L}_{\mathcal{P}}}^* = \{s \in \tilde{\mathcal{L}_{\mathcal{P}}} \mid f(s) = \min_{k \in \tilde{\mathcal{L}_{\mathcal{P}}}} f(k)\} \tag{1.7}$$

Many local search algorithms consider only solutions that satisfy some of the constraints. This set of solutions over which the algorithm is defined is called the search space.

Definition 4 *A search space for a combinatorial optimization problem $\mathcal{P}$ is a set $\hat{\mathcal{L}_{\mathcal{P}}}$ such that $\mathcal{L}_{\mathcal{P}} \subseteq \hat{\mathcal{L}_{\mathcal{P}}} \subseteq \mathcal{N}^n$. Elements of the set $\hat{\mathcal{L}_{\mathcal{P}}}$ often satisfy a subset of $\{C_1, \ldots, C_n\}$.*

The concepts of neighborhood, transition graph, and local optimality are central in local search, since they define how to move from solution to solution.

Definition 5 *A neighborhood is a pair $\langle \hat{\mathcal{L}_{\mathcal{P}}}, N \rangle$, where $\hat{\mathcal{L}_{\mathcal{P}}}$ is a search space and N is a mapping $N : \hat{\mathcal{L}_{\mathcal{P}}} \rightarrow 2^{\hat{\mathcal{L}_{\mathcal{P}}}}$ that defines, for each solution s, the set of adjacent solutions $N(s) \subseteq \hat{\mathcal{L}_{\mathcal{P}}}$. Whenever the relation $s \in N(j) \Leftrightarrow j \in N(s)$ holds, the neighborhood is said to be symmetric.*

Definition 6 *The transition graph $G(\hat{\mathcal{L}_{\mathcal{P}}}, N)$ associated to a neighborhood $\langle \hat{\mathcal{L}_{\mathcal{P}}}, N \rangle$ is the graph whose nodes are solutions in $\hat{\mathcal{L}_{\mathcal{P}}}$ and where an arc $a \rightarrow b$ exists if $b \in N(a)$. The reflexive and transitive closure of $\rightarrow$ is denoted by $\rightarrow^*$.*

A solution is locally optimal if none of its neighbors have a smaller cost. Note that local optimality is always defined with respect to a specific neighborhood function.

Definition 7 *A solution s in $\mathcal{L}_\mathcal{P}$ is locally optimal with respect to N if*

$$f(s) \leq \min_{i \in N(s)} f(i). \tag{1.8}$$

The set of locally optimal solutions with respect to N is denoted ${\mathcal{L}_\mathcal{P}}^+$.

One of the critical issues in local search is to escape local minima, which is why local search algorithms typically feature interesting legality and selection criteria.

Definition 8 *A legality condition L is a function $(2^{\hat{\mathcal{L}_\mathcal{P}}} \times \hat{\mathcal{L}_\mathcal{P}}) \rightarrow 2^{\hat{\mathcal{L}_\mathcal{P}}}$ that filters sets of solutions from the search space. A selection rule $S(\mathcal{M}, s)$ is a function $S : (2^{\hat{\mathcal{L}_\mathcal{P}}} \times \hat{\mathcal{L}_\mathcal{P}}) \rightarrow \hat{\mathcal{L}_\mathcal{P}}$ that picks an element s from $\mathcal{M}$ according to some strategy and decides to accept it or to select the current solution s instead.*

Definition 9 *A local search algorithm for $\mathcal{P}$ is a path*

$$s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k \tag{1.9}$$

in the transition graph $G(\hat{\mathcal{L}_\mathcal{P}}, N)$ for P such

$$s_{i+1} = S(L(N(s_i), s_i), s_i) \quad (1 \leq i \leq k). \tag{1.10}$$

Typically, such a local search produces a final computation state s_k that belongs to ${\mathcal{L}_\mathcal{P}}^+$ (for a given neighborhood function N). The role of heuristics and metaheuristics is to drive the search toward high-quality local optima and, ideally, those in ${\mathcal{L}_\mathcal{P}}^*$.

1.1.4 Properties of Neighborhood

The effectiveness of a local search algorithm critically depends upon its neighborhood. In what follows, we briefly review some fundamental properties of neighborhoods.

Neighborhood Size The size of a neighborhood $N : \hat{\mathcal{L}_\mathcal{P}} \rightarrow 2^{\hat{\mathcal{L}_\mathcal{P}}}$ for a solution s is the set of solutions in $N(s)$. Typically, large neighborhoods induce shorter paths to high-quality solutions but also require more time to explore. This trade-off between the length of the paths and the exploration is a key design decision. Sometimes it is preferable to select a linear neighborhood (in the size of the problem $\mathcal{P}$) over a quadratic neighborhood, even if the resulting path is longer. Sometimes, however, the resulting neighborhood is not large enough to produce high-quality solutions in reasonable time.

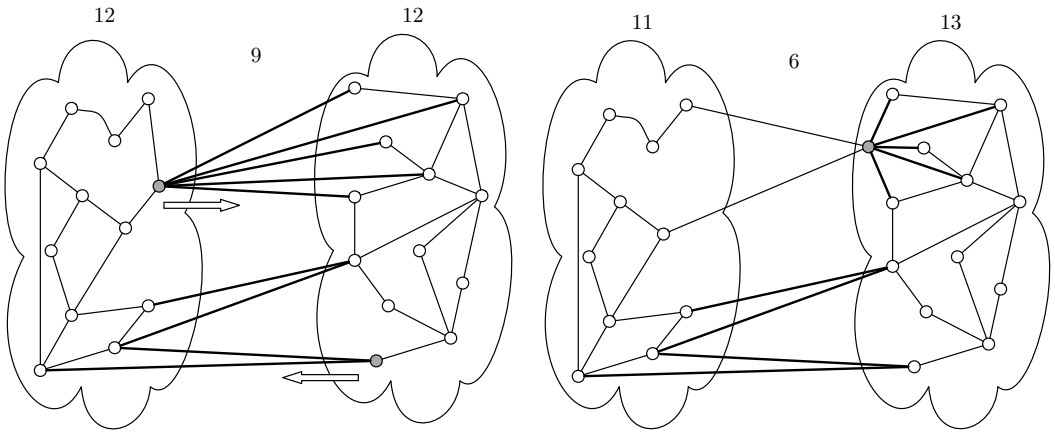


Figure 1.5: A Move from Neighborhood N'

Neighborhood Connectivity Another fundamental property of a neighborhood is its connectivity. Informally speaking, a neighborhood is connected if there exists a path from any solution s to an optimal solution s^*. This ensures that the neighborhood is strong enough to reach optimal solutions, although the heuristic and metaheuristic may prevent the algorithm from reaching them in practice.

Definition 10 *A neighborhood $N : \hat{\mathcal{L}_\mathcal{P}} \rightarrow 2^{\hat{\mathcal{L}_\mathcal{P}}}$ is weakly connected if and only if, for each solution s, there exists a path $s \rightarrow^* s^*$ to an optimal solution s^*.*

Definition 11 *A neighborhood $N : \hat{\mathcal{L}_\mathcal{P}} \rightarrow 2^{\hat{\mathcal{L}_\mathcal{P}}}$ is optimally connected if and only if, for each pair of solutions s_1, s_2, there exists a path $s_1 \rightarrow^* s_2$.*

Using (weakly) connected neighborhoods brings several advantages:

- Local search algorithms typically do not need a restarting strategy to reach optimal solutions, since there exist paths leading from each solution to an optimal solution.[1]
- Randomized heuristics, where there is a nonzero probability of accepting a neighbor $k \in N(s)$ for each solution s, may be guaranteed (under certain conditions) to reach a global optimum (in the limit). In other words, connectivity is a requirement for the convergence proofs of metaheuristics such as simulated annealing.

Neighborhood Constraints One of the critical issues in local search is to determine how to combine feasibility and optimality requirements. The simplest option is to maintain feasibility at all times and explore only feasible solutions in the neighborhood search. This is the approach taken by neighborhood N in the graph-partitioning problem presented in Figure 1.4. The neighborhood implicitly maintains the balancing constraint, and it suffices to start with a balanced solution to

[1] A restarting strategy may still be useful for performance results.

maintain feasibility in the local search. In some applications, however, it is preferable to relax a subset of the constraints and explore a larger search space. Consider, for instance, the neighborhood N' for graph partitioning, in which a single vertex is relocated:

$$N'(\langle P_1, P_2\rangle) = \{\langle P_1 \setminus \{a\}, P_2 \cup \{a\}\rangle \mid a \in P_1\} \cup \{\langle P_1 \cup \{b\}, P_2 \setminus \{b\}\rangle \mid b \in P_2\} \tag{1.11}$$

Figure 1.5 illustrates such a move that decreases the cost of the partition from 9 to 6.

The neighbor N' induces the local search to explore infeasible solutions or, in other words, unbalanced partitions. It is thus important to drive the search not only toward high-quality solutions, but also toward feasibility. There are various ways to approach this issue, depending on the actual problem's structure. One possible way is through designing an objective function that combines feasibility and optimality components. For instance, for a neighborhood N' in graph partitioning, the algorithm may use the objective

$$\alpha \cdot \#\{(x, y) \in E \mid x \in P_1 \ \&\ y \in P_2\} + \beta \cdot \left(\#\{x \in E \mid x \in P_1\} - \frac{n}{2}\right)^2 \tag{1.12}$$

for some well-chosen values of α and β and $\#V = n$. The left member of the objective captures the cost of the partition, while the right member penalizes its imbalance.

1.2 Heuristics and Metaheuristics

In Sections 1.3 and 1.4, we briefly introduce some heuristics and metaheuristics used or referred to in the rest of the thesis. The presentation is limited to the basic information necessary for the following chapters. The heuristics and metaheuristics are described as natural instantiations of the framework presented in Section 1.1.

Heuristics typically choose the next neighbor based only on local information or, more precisely, on the current solution and its neighborhood. They typically drive the search toward local minima (in the case of minimization problems). Metaheuristics, on the contrary, collect information on the execution sequence(s) and aim primarily at escaping local minima and driving the search toward global optimality. As a consequence, heuristics are often characterized as memoryless, while metaheuristics typically include some form of memory or learning. Both heuristics and metaheuristics are presented in a unified framework, in terms of the generic local search template. This is simply a generic version of the local search template presented in Figure 1.2.

Figure 1.6 depicts such a generic version parameterized by the objective function f, as well as the functions L and S for specifying legal moves and selecting the next neighbor. The generic implementation also removes the generation of the initial solution so that metaheuristics can apply the template from different initial solutions.

```
1.  function LocalSearch(f, N, L, S, s) {
2.      s* := s;
3.      for k := 1 to MaxTrials do
4.          if satisfiable(s) ∧ f(s) < f(s*) then
5.              s* := s;
6.          s := S(L(N(s), s), s);
7.      return s*;
8.  }
```

Figure 1.6: The Generic Local Search Template

1.3 Heuristics

Heuristics focus on choosing the next neighbor to move to. They can be classified in various ways, including which changes they allow for the objective value and whether they are systematic or randomized. This section reviews some popular deterministic and randomized heuristics of interest to this thesis. They are typically specified by instantiations of the selection function S, which can be combined with some restrictions on the neighborhood (defined by a function L). For instance, improvement heuristics can be formulated in terms of the following implementation of legal moves:

```
1.  function L-Improvement(N, s)
2.      return { n ∈ N | f(n) < f(s) };
```

Of course, some heuristics potentially consider all neighbors, in which case the L implementation is simply the identity function:

```
1.  function L-All(N, s)
2.      return N;
```

1.3.1 Systematic Heuristics

Systematic heuristics perform a (possibly partial) exploration of the neighborhood to determine the next solution.

Best Neighbor The best-neighbor heuristic, presented earlier, consists of choosing the neighbor with the best evaluation:

```
1.  function S-Best(N,s)
2.      N* := { n ∈ N | f(n) = min_{s∈N} f(s) };
3.      return n ∈ N* with probability 1/#N*;
```

where $\#S$ denotes the size of S. Observe that line 3 selects one of the best neighbors randomly to break ties. A best-improvement local search can then be specified as the following instantiation of the generic local search using the appropriate L and S implementations:

```
1.  function BestImprovement(s)
2.      return LocalSearch(f,N,L-Improvement,S-Best);
```

First Neighbor The best-improvement heuristic requires a complete scan of the neighborhood, which may be costly when the neighborhood is large. The first-improvement heuristic simply selects the first move that improves the current solution s. It assumes, without loss of generality, a function $lex(n)$ that specifies the lexicographic order of a neighbor n when scanning the neighborhood.

```
1.  function S-First(N,s)
2.      return n ∈ N minimizing lex(n);
```

A first-improvement local search can thus be specified as the following instantiation of the generic local search:

```
1.  function FirstImprovement(s)
2.      return LocalSearch(f,N,L-Improvement,S-First);
```

1.3.2 Random Walks

Systematic heuristics perform a (possibly partial) exploration of the neighborhood to select the next solution. Random-walk heuristics are quite different: they select an element of the neighborhood randomly and decide whether to accept it as the next solution.

Random Improvement Random improvement, the simplest example of random-walk heuristics, accepts a neighbor if it improves the current solution.

```
1.  function S-RandomImprovement(N,s)
2.      select n ∈ N with probability 1/#N;
3.      if f(n) < f(s) then
4.          return n;
5.      else
6.          return s;
```

Note that line 6 returns the current solution s. This means that, in random-walk heuristics, the current solution is implicitly part of the neighborhood.[2] The randomized nature of random-walk heuristics seems critical in some applications (for instance, for the TTP problem, as shown later in Chapter 3) to reach high-quality solutions. A random-improvement walk is thus specified as the following instantiation of the generic local search:

```
1.  function RandomImprovement(s)
2.      return LocalSearch(f,N,L-All,S-RandomImprovement);
```

[2] Alternatively, a random-walk heuristic may be seen as composed of micro-steps that select solutions in the neighborhood randomly until an appropriate neighbor is found.

The Metropolis Heuristic The Metropolis heuristic is an interesting extension of random improvement, in which some moves degrading the objective value are allowed. Once again, the Metropolis algorithm selects a random neighbor n. If neighbor n does not degrade the current solution, that is, if $f(n) \leq f(s)$, it is accepted as the next solution. If it degrades the objective value, the Metropolis algorithm accepts the move with a small probability

$$exp\left(\frac{-(f(n) - f(s))}{t}\right) \tag{1.13}$$

that depends on the distance between $f(n)$ and $f(s)$ and a parameter t (called the temperature). It rejects n otherwise. More precisely, the Metropolis heuristic is specified by

1. **function** S-METROPOLIS[T](N,s)
2. **select** $n \in N$ with probability $1/\#N$;
3. **if** $f(n) \leq f(s)$ **then**
4. **return** n;
5. **else** with probability $exp(\frac{-(f(n)-f(s))}{t})$
6. **return** n;
7. **else**
8. **return** s;

1.4 Metaheuristics

The heuristics presented in Section 1.3 focus on choosing the next solution from the neighborhood using only local information on the quality of the neighbors. Their goal is to reach high-quality local minima quickly. Metaheuristics have a fundamentally different role: they aim at escaping these local minima and at directing the search toward global optimality. This objective can be approached in many different ways, which explains the great diversity and the wealth of results in this field. Once again, this section is not intended to be comprehensive but rather to review the metaheuristics necessary for the presentation to follow.

1.4.1 Iterated Local Search

Iterated local search is a ubiquitous metaheuristic that iterates a specific local search from different starting points in order to sample various regions of the search space and to avoid returning a low-quality local minimum. This idea can be further refined by generating a new starting point from the local minimum last returned by the local search. For instance, the new starting point may be generated by perturbing the local minimum.

The template for iterated local search is depicted in figure 1.7. The algorithm generates an initial solution and then performs a number of iterations (lines 5–8). Each iteration consists of a local search (line 5) and the generation of a new starting point (line 8) obtained by perturbing the local minimum s or by generating a new initial solution.

```
1.  function IteratedLocalSearch(f, N, L, S) {
2.      s := generateInitialSolution();
3.      s* := s;
4.      for k := 1 to MaxSearches do
5.          s := LocalSearch(f, N, L, S, s);
6.          if f(s) < f(s*) then
7.              s* := s;
8.          s := generateNewSolution(s);
9.      return s*;
10. }
```

Figure 1.7: The Iterated Local Search Template

```
1.  function SimulatedAnnealing(f, N) {
2.      s := generateInitialSolution();
3.      t_1 := initTemperature(s);
4.      s* := s;
5.      for k := 1 to MaxSearches do
6.          s := LocalSearch(f,N,L-All,S-Metropolis[t_k],s);
7.          if f(s) < f(s*) then
8.              s* := s;
9.          t_{k+1} := updateTemperature(s,t_k);
10.     return s*;
11. }
```

Figure 1.8: The Simulated Annealing Template

Observe that there are no restrictions on the embedded local search used in line 5. Hence, iterated local search is naturally composed with the other metaheuristics presented in this section.

1.4.2 Simulated Annealing

Simulated annealing is a popular metaheuristic based on the Metropolis heuristic. As we saw earlier, the Metropolis heuristic accepts a degrading move with probability

$$exp\left(\frac{-(f(n)-f(s))}{t}\right) \tag{1.14}$$

where t is a parameter of the heuristic. Different values of t produce different trade-offs between the quality of the solutions and the execution time. The key idea underlying simulated annealing is to iterate the Metropolis algorithm with a sequence of decreasing temperatures

$$t_0, t_1, \ldots, t_i, \ldots \quad (t_{k+1} \leq t_k). \tag{1.15}$$

The goal is to accept many moves initially in order to sample the search space widely (large values of t_k) and to move progressively toward small values of t_k, thus converging toward random improvement and (we hope) a high-quality local minimum when $t_i \to 0$.

```
1.  function VariableNeighborhoodSearch(f, N, L, S) {
2.      s := generateInitialSolution();
3.      s* := s;
4.      k := 1;
5.      while k ≤ MaxShaking do
6.          s := select n ∈ N_k(s);
7.          s+ := LocalSearch(f,N,L,S,s);
8.          k := k + 1;
9.          if f(s+) < f(s*) then
10.             s := s+;
11.             s* := s;
12.             k := 1;
13.     return s*;
14. }
```

Figure 1.9: The Variable Neighborhood Search Template

The template for simulated annealing in figure 1.8 indicates that the local search algorithm in line 6 is a Metropolis algorithm with temperature t_k. Two critical decisions in simulated annealing are the choice of the initial temperature (line 3) and the cooling schedule (line 9) which specifies how to decrease the temperature. Both of these can be chosen experimentally or can be derived systematically for specific instances [1, 24]. In particular, the initial temperature and the cooling schedule can be derived by performing random walks at different temperatures.

1.4.3 Variable Neighborhood Search

Variable neighborhood search (VNS) is a metaheuristic featuring an interesting way to escape local minima. It works with a collection of neighborhoods $N_1, \ldots, N_i$ to diversify the current solution (local minimum). The intuition is that the neighborhoods $N_1, \ldots, N_i$ are increasing in size, providing more opportunities for significant diversifications over time. Figure 1.9 depicts the template for variable neighborhood search. At all times, VNS maintains a degree of diversification k specifying which neighbor to use when the search reaches a local minimum. A VNS iteration then consists in selecting a solution s in N_k (line 6) and applying a local search on solution s to obtain a solution s^+ (line 7). Whenever s^+ improves the best solution, VNS moves to s^+ and resets the diversification degree to 1 in order to explore the search region around s^+ more extensively (lines 9–12). Otherwise, the diversification degree is incremented (line 8).

One key issue in VNS is the definition of the neighborhoods $N_1, \ldots, N_i$. A traditional technique is to define all neighborhoods in terms of a unique neighborhood N' and its associated functions L' and A'. Neighborhood N_k then specifies the set of solutions that can be reached in k moves using N', L', and A':

$N_0(x) = N'(x)$;
$N_k(x) = S'(L'(N_{k-1}(s), s), s)$;

```
1.  function LocalSearch(f, N, L, S, s_1) {
2.      s* := s_1;
3.      τ := ⟨s⟩;
4.      for k := 1 to MaxTrials do
5.          if satisfiable(s) ∧ f(s_k) < f(s*) then
6.              s* := s_k;
7.          s_{k+1} := S(L(N(s_k), τ), τ);
8.          τ := τ :: s_{k+1};
9.      return s*;
10. }
```

Figure 1.10: The Generic Local Search Template Revisited

1.4.4 Tabu Search

Tabu search is a popular and effective metaheuristic that encompasses a great variety of techniques. This thesis generally follows Simulated Annealing based approaches. However, given that one of its main aspects is showing how one can adapt concepts from Tabu Search into Simulated Annealing, this section reviews some of the concepts necessary for a clearer presentation in subsequent chapters.

To understand the intuition underlying tabu search, it is useful to generalize slightly the generic local search presented earlier. The new generic local search, presented in figure 1.10, consists mainly of maintaining the sequence

$$\tau = \langle s_0, \ldots, s_{k-1}, s_k \rangle \tag{1.16}$$

of solutions explored so far. The sequence τ is initialized with s in line 3 and extended in line 8 by appending solution s_{k+1}. Function L and S are also slightly generalized to accommodate sequences of solutions instead of single solutions. All the implementations presented thus far extend naturally to this framework by working on the last solution in the sequence.

With this minor generalization, the intuition behind tabu search can be expressed and formalized concisely. Indeed, as a first approximation, tabu search can be described as the following strategy:

> Given a sequence $\langle s_0, \ldots, s_{k-1}, s_k \rangle$, select s_{k+1} to be the best neighbor in $N(s_k)$ that has not yet been visited.

As a consequence, tabu search can be viewed (in a first approximation) as the combination of a greedy strategy with a definition of legal moves ensuring that a solution is never visited twice:

```
1.  function TabuSearch(f,N,s)
2.      return LocalSearch(f,N,L-NotTabu,S-Best);
```

where

```
1.  function L-NotTabu(N,τ)
2.      return { n ∈ N  |  n ∉ τ };
```

There are two interesting features to highlight here. First, the definition of legal moves imposes no constraint on the objective value and allows the local search to select moves degrading the quality of the current solution, thus escaping local minima. Second, the greedy nature of the tabu search ensures that the objective function does not degrade too much at any step, since the best neighbor is always chosen.

Short-Term Memory The main difficulty with this tabu-search definition is the need to keep track of all visited solutions: as the computation proceeds, the memory requirements quickly become prohibitive. To remedy this limitation, tabu search uses a short-term memory to prevent the search from returning to recently visited solutions. Such short-term memory may not prevent the local search from revisiting solutions entirely, however, so it is typically combined with other techniques that will be described shortly.

The simplest way to implement a short-term memory is to maintain only a small suffix of the execution sequence, i.e, recently visited solutions. However, even such an implementation may be too costly in space and time, since it requires the search to compare solutions. As a consequence, tabu-search algorithms typically maintain an abstraction $\tilde{\tau}$ of the sequence suffix. The abstraction is often problem-dependent, although many guidelines are available for its specification.

A popular technique is the *transition abstraction* that stores the transitions, not the states, since moves only involve a few variables in general. The tabu neighbors are then defined as those solutions obtained through the inverse of stored transitions. Consider graph partitioning again and a transition

$$s_1 \rightarrow s_2 \tag{1.17}$$

based on neighborhood N. It follows that there exist two vertices a and b such that s_2 is the partition s_1 in which a and b have been swapped:

$$s_2 = swap(s_1, a, b). \tag{1.18}$$

As a consequence, the transition can be abstracted by remembering the pair of vertices (a, b) to avoid swapping a and b again in the near future. More generally, a sequence

$$s_0 \rightarrow \ldots \rightarrow s_k \tag{1.19}$$

can be abstracted by a sequence of pairs

$$\langle (a_1, b_1), \ldots, (a_k, b_k) \rangle \tag{1.20}$$

such that

$$s_{i+1} = swap(s_i, a_i, b_i). \tag{1.21}$$

A solution $n \in N(s_k)$ is then defined as tabu if can be obtained from s_k by swapping a pair (a_i, b_i), that is, if there exists i $(1 \leq i \leq k)$ such that

$$n = swap(s_k, a_i, b_i). \tag{1.22}$$

Intuitively, this means that the local search cannot swap two vertices that have been swapped recently. Moreover, each iteration of the local search then drops an old pair, say (a_1, b_1), and adds a new pair, say (a_{k+1}, b_{k+1}). Note that, in tabu search, the abstract sequence $\tilde{\tau}$ is called a tabu list and may contain the transitions, the inverse transition, or both.

As a consequence, with short-term memory, the legal moves can be specified as follows:

1. **function** L-NotTabu(N,$\tilde{\tau}$)
2. **return** $\{\ n \in N\ \mid\ \neg tabu(n, \tilde{\tau})\ \}$;

where, informally speaking, $tabu(n, \tilde{\tau})$ holds if the neighbor is tabu with respect to the tabu list (abstract sequence) $\tilde{\tau}$. Formally, $tabu(n, \tilde{\tau})$ can be specified as

$$tabu(n, \tilde{\tau}) \equiv \exists \tau \in \gamma(\tilde{\tau}) : n \in \tau. \tag{1.23}$$

where $\gamma(\tilde{\tau})$ is the set of sequences abstracted by $\tilde{\tau}$.

It is important to observe that transition abstractions are at the same time too weak and too strong. On the one hand, transition abstractions are too weak because they cannot prevent the tabu search from revisiting solutions, since only a sequence suffix is abstracted. On the other hand, they are too strong since they forbid moves that should be allowed. Indeed, a transition abstraction $\tilde{\tau}$ represents a set of sequences $\gamma(\tilde{\tau})$ that may be very different from τ but cannot swap the vertices in the tabu list.

Transition abstractions do not abstract any information about the states of a transition $s_1 \rightarrow s_2$. As a result, as mentioned, they may forbid too many transitions to unvisited nodes and prevent the search from reaching high-quality solutions. It is possible to be more discriminating by storing information about the states as well. For instance, in graph partitioning, the tabu search may abstract the states by their objective values[3] and store quadruples (f_1, f_2, a, b) to capture a transition $s_1 \rightarrow s_2$ where

$$s_2 = swap(s_1, a, b)\ \wedge\ f(s_1) = f_1\ \wedge\ f(s_2) = f_2. \tag{1.24}$$

As a consequence, a move $s \rightarrow n$ is tabu if it reverses an earlier move:

$$n = swap(s, b, a)\ \wedge\ f(s) = f_2\ \wedge\ f(n) = f_1. \tag{1.25}$$

Aspiration Since the tabu search stores sequence abstractions and not the sequences themselves, it may forbid transitions $s_1 \rightarrow s_2$, in which s_2 has not been visited before. Some of these transitions may be quite desirable, for instance when s_2 would be the best solution found so far (that is, $f(s_2) < f(s^*)$). To overcome this limitation, tabu search algorithms often feature an aspiration criterion that specifies when the tabu status may be overridden. The simplest and most widely used aspiration criterion overrides the tabu status of those moves that improve the best solution found so far. The resulting legal moves are specified as

1. **function** L-NotTabu-Asp(N,$\tilde{\tau}$)
2. **return** $\{\ n \in N\ \mid\ \neg tabu(n, \tilde{\tau})\ \vee\ f(n) < f(s^*)\ \}$;

[3]The objective value is often called a witness in this context.

```
1.  function IntensifiedLocalSearch(f, N, L, S) {
2.      s := generateInitialSolution();
3.      s* := s;
4.      for k := 1 to MaxSearches do
5.          s := LocalSearch(f, N, L, S, s); {
6.          if f(s) < f(s*) then
7.              s* := s;
8.          s := s*;
9.      return s*;
10. }
```

Figure 1.11: A Simple Template for the Intensification of a Local Search

Long-Term Memory The tabu list abstracts a small suffix of the solution sequence and cannot capture long-term information. As a consequence, it cannot prevent the search from taking long walks, whose solutions have low-quality objective values, or spending too much time in the same region, leaving other parts of the search space unexplored. Many tabu-search algorithms are thus enhanced with additional long-term memory structures to intensify and diversify the search.

Intensification entails storing high-quality solutions during the search and returning to these solutions periodically. It makes it possible to explore extensively the region of the search space in which the best solutions so far have been found. The simplest intensification scheme consists of returning to the best solution found so far. Its template is depicted in figure 1.11; the intensification takes place in line 8. Interestingly, although the template is often used in the context of Tabu Search, it is not specific to tabu search. For example, in Chapter 4, we show an intensification scheme in the framework of Simulated Annealing.

Diversification entails directing the search towards other regions of the search space. Once again, there are many possible ways to achieve such a goal. They include iterative local search to perturb or restart the search, as well as strategic oscillation, which consists of changing the objective function to balance the time spent in the feasible and infeasible regions.

1.5 Cooperative Parallel Search

Given the low speed often exhibited by meta-heuristics such as simulated annealing, especially for larger instances, many researchers have attempted to take advantage of parallelization, in order to speed up or even improve the solution quality of such meta-heuristics. The population-based scheme we are proposing in Chapter 4 falls, to a great extent, into this category of cooperative parallel search schemes.

In this section, we give a brief overview of the various ways in which parallelism has been combined with local search schemes, and, in particular, schemes that are related to the one we propose. We are going to explore these relationships in further detail in the corresponding chapter of the thesis.

There is a great variety of proposed parallel search schemes, a large number of which is based

on simulated annealing. The most straight-forward schemes are attempts to parallelize sequential versions of simulated annealing (e.g., [47]). However, the schemes appearing in the literature cover a much broader spectrum than that.

Onbaşoğlu et. al. 2001 [37] provide an extensive survey of parallel simulated annealing algorithms and compare them experimentally on global optimization problems. They also classify those schemes into application-dependent and application-independent parallelization.

In the first category, the problem instance is divided among several processors, which communicate only to deal with dependencies. For instance, in VLSI design, the processors specialize on different areas of the circuit. See [22] for a detailed account of parallel simulated annealing techniques for VLSI design. In the second category, Onbaşoğlu et. al. further distinguish between asynchronous parallelization with no processor communication, synchronous parallelization with different levels of communication, and highly-coupled synchronization in which neighborhood solutions are generated and evaluated in parallel. In the first two cases, processors work on separate Markov chains while, in the third case, they cooperate on a single Markov chain.

Communication patterns between processors can take the form of simple transmission of cost values, occasional exchange of (possibly partial) solutions, or even intensive exchanges of solutions. Hybrid schemes combining different forms of communication have also been developed (e.g., [27]). There are schemes that cannot be easily classified, such as the parallel simulated annealing in [8], in which the processors work on highly inter-dependent Markov chains by mixing states.

In Chapter 4, we propose a scheme that can be viewed as an application-independent algorithm with synchronous parallelization and periodic exchange of solutions. The scheme proposed in [25] (which only exchanges partial solutions) and the SOEB-F algorithm [37] are probably the closest to ours, although we include some important structural features, that are missing from these two approaches. It is also noteworthy to observe that SOEB-F typically fails to produce sufficiently good solutions [37].

It is also useful to point out that the above classification is not limited to simulated annealing. A cooperative parallel scheme based on tabu search is presented in [5] and is applied to the generalized assignment problem.

We conclude this section with a very short mention of two important types of search algorithms related to our scheme: scatter search and memetic algorithms. In scatter search [29], solutions are not combined but only intensified. Scatter search also features the concept of elite solutions. One could also view our scheme as a degenerated form of memetic algorithms [36], where there is no mutation of solutions: existing solutions are either replaced by the best solution found so far or are "preserved".

1.6 Sport Scheduling

One of the most intriguing features of Sport Scheduling is the big diversity of research interest within the field. Sport Scheduling research ranges from purely theoretical problems in discrete mathematics and combinatorics to large-scale real life applications, involving scheduling big sport leagues in the United Stated and Europe. It also serves as an important showcase of different mathematical and combinatorial optimization techniques, often combined in hybrid approaches to different problems.

A common theme in Sport Scheduling applications is the constant challenge for effective modeling. Real life sport applications impose complicated constraints, often contradictory, and not necessarily of the same significance. It takes a lot of effort and experience to design benchmarks that are, at the same time, meaningful from a researcher's point of view and representative of the salient features of real life problems.

There are many good references in the area. A rich source of theory-oriented information connected to sport scheduling can be found in [57]. A more recent survey on sport scheduling, focusing on round-robin scheduling and also covering representative practical problems, can be found in [42].

In this thesis we focus on two important representative problems in the area of Sport Scheduling. It is interesting to observe that, until recently, local search was not as commonly applied to sport scheduling problems. Moreover, the first local-search based approaches to these two problems did not prove to be as successful as other techniques. As shown in the current work, Local Search can, in fact, be a method of choice for problems in this area.

1.6.1 Break Minimization Problem

Early results on Break Minimization are due to Schreuder [45] who studied the Dutch soccer league. In that paper, Schreuder proved that, for every n, there is a schedule with exactly $n-2$ breaks and that this number is optimal. Schreuder also gave a polynomial time algorithm for constructing such a schedule. This version of the problem is called the *Unconstrained Break Minimization Problem* since the schedule is not fixed. The problem studied in this thesis, in which the schedule is given but not the home/away decisions, is called the *Constrained Break Minimization Problem*.

Régin[43] proposed a constraint programming (CP) approach for Break Minimization. This approach solved the unconstrained version efficiently, but was limited to 20 teams for the constrained version (at least at the time). Trick [48] proposed an integer programming approach which provided solutions to instances with at most 22 teams in times comparable to the CP approach. In relatively recent work, Elf et al., [17] presented a reduction of the constrained version to a cut maximization problem. With the aid of a specialized MAX-CUT solver, they found optimal solutions in reasonable times (e.g., less than 10 minutes) for problems with up to 28 teams.

Elf et. al. also conjectured that the constrained problem is NP-hard, in general, but in P for schedules which have an actual home/away assignment with $n-2$ breaks. This conjecture was (positively) closed in [32] by a transformation to 2SAT. Miyashiro and Matsui [33] also proved that the problem with n breaks are also in P. Moreover, more recently, Miyashiro and Matsui

Miyashiro [34] modeled Break Minimization as a MAX RES CUT problem and applied Goemans and Williamson's approximation algorithm based on semi-definite programming [21]. The resulting solutions are not optimal and the distance with respect to the optimum increases with the size of the instances. In particular, they reported solutions with an additional 3.5 breaks on 24 teams. Among other advanced techniques, one can also mention the Benders decomposition approach to minimizing breaks, proposed by Rasmussen and Trick in [41].

As one can see, these approaches use rather "heavy machinery", involving specialized integer programming solvers and constraint programming tools. Moreover, the results were obtained by experts in the underlying technologies and/or in modeling sport scheduling applications. In contrast, the simulated annealing approach that we proposed in [53] is conceptually simple and easy to implement. Yet, it produces optimal solutions and is significantly faster for large instances.

1.6.2 Traveling Tournament Problem

The Traveling Tournament Problem (TTP) has raised significant interest in recent years since the challenge instances were proposed. The work in [15] described both constraint and integer programming approaches to the TTP which generate high-quality solutions. The combination of Lagrangian relaxation and constraint programming proposed in [6] improved some of the results. Other lower and upper bounds are given in [49], although the details of how they are obtained do not seem to be available in all cases. Note that, similarly to Section 1.6.1 these results were obtained with relatively heavy machinery and state-of-the-art techniques (for example, CP, IP, incorporated in solvers combining different methods).

In 2003, we proposed a simulated algorithm, TTSA, exploring a large neighborhood that produced most of the best-known solutions to the instances [3]. Our neighborhood, or a subset of it, was reused in subsequent work (e.g., [11, 12, 30, 44]) within tabu-search and GRASP approaches or, even, simulated annealing. Recently, Rasmussen and Trick also considered Benders decomposition approaches to the TTP [41]. A more detailed version of the work we proposed in 2003 can be found in [4]. In addition, an enhanced version of the original TTSA algorithm was used in some of our recent work [54], in order to solve more general variants of the TTP.

Unlike research in the direction of improving the best-known upper bounds, research on improving upon the best-known lower bounds for TTP instances has shown much slower progress. Two main reasons for this are: first, the high degree of dependencies between problem variables and, second, the increased complication arising from feasibility patterns involved. Some first lower bounds on TTP instances were proven by Easton et.al. [15], through the computation of what they define as the *Independent Bound*. These lower bounds were further improved by the same authors in [16]. The most success in obtaining lower bounds has been seen for a simpler class of TTP instances, namely the CONST instances, which we describe later in Chapter 3. Lower bounds for this class of instances have been given by Rasmussen and Trick [41], by Fujiwara et.al. [19], and by Urrutia and Ribeiro in [50]. Finally, in a recent paper, Urrutia et.al. [51], utilizing the CONST bounds found above, further improved the best-known lower bound on several of the harder TTP instances.

In both problems (Break Minimization and TTP), our results demonstrate the ability of simulated annealing to successfully attack hard combinatorial optimization problems with much simpler machinery, compared to traditionally employed methods. In the following chapters, we are going to describe our simulated annealing based schemes in more detail starting from simpler schemes, and moving on to more complex ones.

Chapter 2

The Break Minimization Problem

In this chapter, we present a simulated annealing algorithm (BMSA) for Break Minimization, following the presentation of our ICAPS'05 paper [53].

The neighborhood used is very simple and consists of swapping the home/away teams of a particular game, thus maintaining feasibility at all times. Its meta-heuristic fully automates the cooling schedule and the termination criteria by collecting statistics online during the search process. BMSA was applied to the instances proposed in [17]. It finds optimal solutions to all instances regardless of its (random) starting schedules. On the larger instances involving more than 20 teams, it produces significant performance improvements. In particular, it solves instances with 28 teams in less than 20 seconds in average and never takes more than 2 minutes. Moreover, the experimental results indicate that BMSA finds optimal or near-optimal solutions to these large instances within 10 seconds on a 1.66GHz PC. This is an important functionality, since sport scheduling often requires a close interaction between the modeler and the optimization software.

The results are interesting for a number of reasons. First, they show that there exists a local search algorithm for Break Minimization that is both conceptually simple and easy to implement. Second, they indicate that the performance and the quality of the algorithm are excellent, even when CPU time is severely limited. Third, and most intriguingly, this work complements our previous work in [3], in identifying another sport scheduling application, after the TTP, where simulated annealing is a very effective solution technique.

2.1 Problem Description

The sport scheduling application considered in this chapter consists of minimizing breaks in single round-robin tournaments. In a round-robin tournament with n teams (n even), each team plays against every other team over the course of $n-1$ consecutive rounds. Every round consists of $n/2$ games, each team plays exactly once in a round and each game is played in one of the two opponents' home. In other words, a feasible schedule satisfies the following constraints:

- Every team plays exactly once against every other team.
- Every team plays exactly once in every round.
- If team i plays at home against team j in round k, then team j plays against i away, i.e., at i's home.

Feasible schedules are represented by a $n \times (n-1)$ matrix S, such that $S[i,k] = j$ if team i is playing against team j at home in round k, and $S[i,k] = -j$ if team i is playing against team j away in round k. The following matrix

T\R	1	2	3	4	5
1	6	-2	4	3	-5
2	5	1	-3	-6	4
3	-4	5	2	-1	6
4	3	6	-1	-5	-2
5	-2	-3	6	4	1
6	-1	-4	-5	2	-3

is a feasible schedule with 6 teams. The schedule of team i is depicted in row i. For instance, team 3 plays team 4 away, then teams 5 and 2 at home, against team 1 away, and finishes at home against team 6. The columns represent the various rounds.

One way to measure quality for schedules is the number of *breaks*. A break occurs when a team plays two consecutive games at home or two consecutive games away. For instance, in the above example, the games in rounds 2 and 3 of team 3 are both at home, contributing one break. The total number of breaks in the example is 12. More formally, the number of breaks in a schedule is equal to

$$\#\{(i,j) : S[i,j] \cdot S[i,j+1] > 0\,, 1 \leq i \leq n\,, 1 \leq j \leq n-2\} \tag{2.1}$$

and the goal is to minimize the number of breaks.

More precisely, we consider the minimization of breaks for a fixed schedule where only the home/away decisions have been omitted. For instance, the problem may receive, as input, the matrix

T\R	1	2	3	4	5
1	6	2	4	3	5
2	5	1	3	6	4
3	4	5	2	1	6
4	3	6	1	5	2
5	2	3	6	4	1
6	1	4	5	2	3

and must produce an assignment of home/away teams minimizing the number of breaks. More formally, the input is a matrix S and the Break Minimization Problem amounts to finding a feasible schedule S_o satisfying

$$\forall i \in \mathit{Teams}, k \in \mathit{Rounds} : |S_o[i,j]| = S[i,j] \tag{2.2}$$

and minimizing the number of breaks, where $\mathit{Teams} = 1..n$ and $\mathit{Rounds} = 1..n-1$.

2.2 Neighborhood Definition

The neighborhood in the simulated annealing algorithm is remarkably simple: It consists of swapping the home and away teams of a game in a given week. Since a schedule consists of $n(n-1)/2$ matches, the neighborhood size is $O(n^2)$. In other words, the neighborhood consists only of the *SwapHomes* move, which is defined in a similar way with Section 3.1.3 (the only difference being that we now consider single round-robin schedules).

***SwapHomes*(S, T_i, T_j)** The move swaps the home/away roles of teams T_i and T_j. In other words, if team T_i plays at home (respectively away) against team T_j in round r_k, $\mathit{SwapHomes}(S, T_i, T_j)$ is the same schedule as S, except that team T_i now plays away (respectively at home) against team T_j at round r_k. Consider the schedule S:

T\R	1	2	3	4	5
1	6	-2	4	3	-5
2	5	1	-3	-6	4
3	-4	5	2	-1	6
4	3	6	-1	-5	-2
5	-2	-3	6	4	1
6	-1	-4	-5	2	-3

The move $\mathit{SwapHomes}(S, T_3, T_5)$ produces the schedule

T\R	1	2	3	4	5
1	6	-2	4	3	-5
2	5	1	-3	-6	4
3	-4	-5	2	-1	6
4	3	6	-1	-5	-2
5	-2	3	6	4	1
6	-1	-4	-5	2	-3

It is important to note that the neighborhood is connected and that the quality of a move can be evaluated in constant time, since there are at most 2 adjacent teams on each affected line. As a

```
1.    function METROPOLIS(S, T) {
2.        bestCost ← cost(S)
3.        best ← S
4.        c ← 0
5.        while c ≤ phaseLength do
6.            select S′ ∈ neighborhood(S)
7.            Δ ← cost(S′) − cost(S)
8.            if Δ ≤ 0 then
9.                accept ← true
10.           else
11.               accept ← true with probability exp(−Δ/T),
12.                        false otherwise
13.           end if
14.           if accept then
15.               S ← S′
16.               if cost(S) < bestCost then
17.                   bestCost ← cost(S)
18.                   best ← S
19.               end if
20.           c++
21.           end if
22.       end while
23.       return S
24.   }
```

Figure 2.1: The Metropolis Algorithm

result, the simulated annealing algorithm can evaluate a large number of moves over the course of its computations.

Other, more complex, neighborhoods were considered, but they do not compare to the effectiveness of this simple neighborhood when it is combined with simulated annealing. These more complex neighborhoods swap (partial) rounds and (partial) team schedules and are very effective for the TTP problem.

2.3 The BMSA Algorithm

We now present the simulated annealing algorithm used for Break Minimization, by gradually introducing its building blocks.

2.3.1 The Metropolis Heuristic

The core of the algorithm is a Metropolis heuristic which selects schedules randomly from the neighborhood [31]. The algorithm moves to the neighboring schedule if it decreases the number of breaks or with probability $\exp(-\Delta/T)$ where $\Delta = cost(S') - cost(S)$, S is the current schedule, and

S' is the selected neighbor. The algorithm is depicted in Figure 2.1. It receives, as input, a schedule and the parameter T (the temperature). Note the termination condition (line 5) which specifies a maximum number of iterations and lines 11-12 which define when to accept moves increasing the number of breaks. The meaning of parameter *phaseLength* will become clear in the next section.

2.3.2 The Simulated Annealing Meta-Heuristic

To control the temperature T in the Metropolis heuristic, the algorithm uses a simulated annealing meta-heuristic. More precisely, the simulated annealing algorithm iterates the Metropolis algorithm for different values of T. It starts with high temperatures, where many non-improving moves are accepted, and progressively decreases the temperature to focus the search.

The choice of the initial temperature and its updating rule is fully automated in the algorithm and does not require any user intervention. These operations, as well as various termination conditions, are derived from analogies with statistical physics. The rest of this section describes these elements in detail, starting from the analogy with statistical physics in order to provide the necessary intuition. (For an extensive treatment of the analogy, see [55, 56].)

2.3.3 Analogy with Statistical Physics

The analogy consists in viewing the search process (the Metropolis algorithm) as a physical system. Given a temperature T and a starting state S_T, the physical system performs a number of transitions moving from state to state. The sequence of transitions is called a *phase* and the number of transitions is called the *phase length*. It can be shown that, under some assumptions, the physical system will reach an equilibrium. This means that, as the phase length goes to infinity, the probability distribution of the states (i.e., the probability that the system be in a given state) converges to a stationary distribution.

As a consequence, for a given temperature T, it makes sense to talk about the statistical measures of the equilibrium, including its expected cost $\mu_T(C)$ and its variance $\sigma_T^2(C)$. Obviously, since the search algorithm is finite, it does not reach an equilibrium in general, which is why one generally talks about quasi-equilibria (which approximate the "real" equilibria).

2.3.4 Estimating the Statistical Measures

To automate the choice of the temperatures and the termination conditions, the algorithm estimates the values $\mu_T(C)$ and $\sigma_T^2(C)$ for various pairs (T, S_T). The estimation process is simple: It simply consists of performing a relatively large number of transitions. If N denotes the number of transitions and C_k denotes the cost after k transitions, the estimations can be specified as shown in

Equations 2.3 and 2.4. The value N is simply chosen as the phase length of the Metropolis algorithm.

$$\mu_T(C) \simeq \overline{C}_T = \frac{1}{N}\sum_{k=1}^{N} C_k \tag{2.3}$$

$$\sigma_T^2(C) \simeq \overline{\sigma}_T^2 = \frac{1}{N}\sum_{k=1}^{N}(C_k - \overline{C}_T)^2 \tag{2.4}$$

2.3.5 The Initial Temperature

Now that the values $\mu_T(C)$ and $\sigma_T^2(C)$ can be estimated, the temperature T is simply initialized to $\alpha \times \overline{\sigma}_\infty$, i.e., a constant times the estimation of the standard deviation when all moves are accepted. This initialization is, in fact, a variation of the scheme proposed in [24].

2.3.6 Updating the Temperature

When the quasi-equilibrium is reached for a temperature T, simulated annealing decreases T before applying the Metropolis algorithm again. The stationary distribution of the schedules generated during a phase generally depends on the temperature T and, whenever the temperature is decreased, the algorithm will move away from the previous equilibrium and reach a new one. Intuitively, one would expect that significant decreases in the temperature (and hence few phases) require longer phases to reach their quasi-equilibria. Therefore, there is a tradeoff between the temperature decreases and the length of the phases.

As we saw in the experimental results on the TTP, slow cooling schedules, together with short phases, produce high-quality solutions in reasonable time. The algorithm presented here adopts, and automates, the same design decision. Informally speaking, the idea is to strive for a smooth evolution of the equilibria. More precisely, the probabilities $q_k(S)$ of having a schedule with cost S in the stationary cost distribution in phase k should be close to the same probabilities in phase $k+1$ or, in symbols,

$$\forall S: \ \frac{1}{1+\beta} < \frac{q_k(S)}{q_{k+1}(S)} < 1+\beta, k = 1, 2, \ldots, \tag{2.5}$$

for some constant β.

Aarts and van Laarhoven [1], show that, under some reasonable assumptions, Equation 2.5 yields the temperature update rule shown on Equation 2.6:

$$T_{k+1} = T_k \cdot \left(1 + \frac{\ln(1+\beta)\cdot T_k}{3\overline{\sigma}_{T_k}(C)}\right)^{-1} \tag{2.6}$$

where T_k is the temperature of phase k.

2.3.7 Early Phase Termination

A phase typically terminates when it has performed its maximum number of iterations. However, now that estimations of the statistics of the quasi-equilibria are available, it is possible to speed up

the algorithm by detecting that the search process is close to a quasi-equilibrium [24]. Assuming a normal distribution for the costs,[1] the probability, for a chosen δ, of the cost being within the interval

$$[\mu_T(C) - \delta \cdot \sigma_T(C), \mu_T(C) + \delta \cdot \sigma_T(C)] \tag{2.7}$$

is equal to

$$\texttt{erf}(\delta/\sqrt{2})$$

where

$$\texttt{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{2.8}$$

is the error function of the normal distribution. Thus, when the number of the schedules with costs in the above interval divided by the number of transitions performed in the current phase gets close enough to $\texttt{erf}(\delta/\sqrt{2})$ (e.g., within 1%), the phase is terminated prematurely.

2.3.8 Early Termination of the Cooling Process and Reheats

The cooling process itself can also be terminated early. This happens when the temperature or the variance estimation are too low, in which case the algorithm is typically stuck in a local optimum. When this happens and additional computing time is available, the simulated annealing algorithm performs a reheat, i.e., it resets the temperature to three times the temperature of the current best schedule.[2] The algorithm terminates when the computing time available is exhausted or when it has performed a fixed number of reheats without improvement to the best solution.

2.3.9 The Simulated Annealing Algorithm (BMSA)

Figure 2.2 revisits the Metropolis algorithm to remove some of the initializations which are no longer necessary and to incorporate the early termination criterion (line 3). Figure 2.3 depicts the complete simulated annealing scheme (BMSA). The algorithm can be viewed as a hybridization of the schemes proposed in [24, 1], together with some extensions to improve performance. Line 2 initializes the schedule randomly, i.e., it assigns the home/away decisions randomly. Lines 3-6 performs a number of initializations, including the temperature in line 5. Lines 8-24 iterate the core of the algorithm. Each iteration consists of a number of phases (lines 9-21) and a reheat (lines 22-23). Line 22 simply increases the reheat counters, while line 23 resets the temperature to a constant times the temperature of the best solution found so far. The phases are executed for *maxPhases* iterations or until the temperature or the standard deviation are too small (line 21). A phase estimates the mean and variance for the specific temperature (as indicated earlier in Subsection "Estimating the Statistical Measures"), applies the (revised) Metropolis algorithm, and decreases the temperature. If the cost has improved during the Metropolis algorithm, the reheat and phase counters are reset.

[1] Normal distribution assumptions are supported by numerical experiments for large random combinatorial optimization problems; see also Section 4.3 in [55].

[2] This value was chosen based on very limited experimentation.

```
1.   function METROPOLIS-REV(S, T, μ_T, σ_T^2, bestCost, best) {
2.       c ← 0
3.       while c ≤ phaseLength ∧ ¬equilibrium(μ_T, σ_T^2) do
4.           select S' ∈ neighborhood(S)
5.           Δ ← cost(S') − cost(S)
6.           if Δ ≤ 0 then
7.               accept ← true
8.           else
9.               accept ← true with probability exp(−Δ/T),
10.                          false otherwise
11.          end if
12.          if accept then
13.              S ← S'
14.              if cost(S) < bestCost then
15.                  bestCost ← cost(S)
16.                  best ← S
17.              end if
18.          end if
19.          c++
20.      end while
21.      return S
22.  }
```

Figure 2.2: The Metropolis Algorithm Revisited

```
1.  function BMSA() {
2.      S ← a random initial schedule
3.      bestCost ← cost(S)
4.      best ← S
5.      T ← α · σ̄_∞
6.      bestTemp ← T
7.      reheat ← 0
8.      while reheat ≤ maxReheats do
9.          k ← 0
10.         repeat
11.             compute μ̄_T(C) and σ̄²_T(C)
12.             oldBestCost ← bestCost
13.             S ← Metropolis-Rev(S, T, μ̄_T(C), σ̄²_T(C))
14.             if bestCost < oldBestCost then
15.                 reheat ← 0
16.                 k ← 0
17.                 bestTemperature ← T
18.             end if
19.             T ← T · (1 + ln(1+β)·T / (3σ̄_T(C)))^{-1}
20.             k++
21.         until k > maxPhases ⋁ T ≤ ε ⋁ σ̄_T(C) = 0
22.         reheat++
23.         T ← γ · bestTemperature
24.     end while
25.     return S
26. }
```

Figure 2.3: The Simulated Annealing Algorithm (BMSA)

2.4 Experimental Results with BMSA

This section presents the experimental results of the simulated annealing algorithm (BMSA), compares them to the state-of-the-art algorithm presented in [17, 34] (MCMP), and indicates that BMSA clearly dominates earlier approaches.

2.4.1 The Instances

The instances used in the experimental results were provided by the authors of [17]. For some reason, these instances are not exactly the same as those they used in [17], although they are generated in the same fashion. However, the authors also gave us their optimal solutions, as well as their computation times on a double processor PC running Linux with two Intel Pentium 4 CPU at 2.80GHz with 512 KB cache and 1GB RAM. Their code uses CPLEX 7.1 and runs only on one CPU. Note also that these authors ran their code without any parameter tuning, leaving some room

	MCMP			BMSA		
n	Minimum	Average	Maximum	Minimum	Average	Maximum
4	2	2.0	2	2	2.0	2
6	4	4.0	4	4	4.0	4
8	6	7.2	8	6	7.2	8
10	10	12.2	14	10	12.2	14
12	14	17.6	20	14	17.6	20
14	22	24.8	28	22	24.8	28
16	28	31.0	34	28	31.0	34
18	38	41.4	46	38	41.4	46
20	48	52.0	54	48	52.0	54
22	56	61.0	66	56	61.0	66
24	66	73.6	78	66	73.6	78
26	82	85.0	90	82	85.0	90
28*	94	99.2	104	94	99.2	104
28	-	-	-	92	98.00	106
30	-	-	-	110	116.23	124

Table 2.1: Quality of the Schedules: Number of Breaks in MCMP and BMSA

for possible improvements [18].

The instances are generated as follows [17]. For every value of n, an optimal schedule with $n-2$ breaks is computed using Schreuder's algorithm. Then, 10 different instances are created by random permutations of the columns. The resulting schedules typically feature many more breaks in their optimal solutions.

2.4.2 Experimental Setting for BMSA

The experimental results for BMSA were obtained as follows. For each instance, BMSA was applied on 20 different random initial schedules with 100 phases of length

$$20 * |neighborhood| = 20 * n * (n-1)/2, \tag{2.9}$$

maxReheats = 200, $\alpha = 0.5$, $\beta = 0.1$, $\gamma = 3$, $\delta = 1.2$. No special effort has been spent tuning these parameters, which are rather natural. To increase performance, the algorithm also terminates early when BMSA has not improved the best solution for $2t$ seconds, where t is the CPU time BMSA took to find the current best solution. For instance, if BMSA found the current best solution after $t = 15$ CPU seconds, it will terminate in the next $2t = 30$ seconds if no better solution is found. BMSA was run on an AMD Athlon MP 2000+ at 1.66GHz.

2.4.3 Quality of the Schedules

Table 2.1 depicts the quality results. It compares MCMP, a complete search method, and BMSA. A line in the table corresponds to a number of teams n and the results report the minimum, maximum, and average number of breaks for all instances with n teams. Interestingly, for every number of teams

	MCMP			BMSA		
n	Minimum	Average	Maximum	Minimum	Average	Maximum
4	0.0	0.00	0.0	0.0	0.00	0.0
6	0.0	0.00	0.0	0.0	0.00	0.0
8	0.0	0.00	0.0	0.0	0.00	0.1
10	0.0	0.01	0.0	0.0	0.10	0.2
12	0.0	0.01	0.0	0.1	0.29	0.5
14	0.0	0.04	0.1	0.3	0.57	1.3
16	0.0	0.08	0.1	0.7	1.02	3.1
18	0.1	0.43	1.6	1.2	2.36	16.0
20	0.3	0.68	2.1	1.8	3.06	24.8
22	0.4	3.05	18.0	2.6	4.42	25.3
24	0.8	24.00	179.9	3.6	9.52	71.4
26	2.2	53.04	435.6	4.9	13.30	92.2
28*	7.0	465.60	1905.0	6.8	18.76	104.9
28	-	-	-	5.6	22.07	296.9
30	-	-	-	8.2	78.58	684.1

Table 2.2: Performance Comparison: Computation Times in CPU seconds of MCMP and BMSA

n and every instance with n teams, BMSA finds the optimal number of breaks and this regardless of the starting point. In other words, for every number of teams, BMSA finds the optimal solution for all instances and all the starting points. For this reason, we isolate these last rows in the table for clarity. Note that there are two lines for the results for 28 teams. The line 28* reports only results on the five instances that MCMP can solve optimally. Note also that, for 30 teams, the BMSA solutions are 6 breaks below those reported in [34].

2.4.4 Performance of the Algorithm

Table 2.2 depicts the computation times in seconds for both MCMP and BMSA. One has to bear in mind that the results for MCMP are on a 2.8GHz machine, while the results for BMSA are for a 1.66GHz machine (about 40% slower). When $n \leq 20$, MCMP is extremely fast and BMSA does not bring any advantage besides its simplicity.

For larger number of teams, BMSA brings significant benefits and it scales much better than MCMP. For $n = 28$, BMSA is about 25 times faster than MCMP without even accounting for the faster speed of their machine. For up to 28 teams, BMSA is also faster than the approximation algorithm and produces superior solutions. The efficiency of the algorithm seems comparable on 30 teams, although BMSA once again significantly reduces the number of breaks. As a consequence, the performance, and the scaling, of BMSA is quite remarkable, especially in light of the simplicity of the implementation, its generality, and the quality of the results.

	BMSA-R		
n	Minimum	Average	Maximum
4	0.0	0.00	0.0
6	0.0	0.00	0.0
8	0.0	0.00	0.0
10	0.0	0.07	0.1
12	0.1	0.17	0.3
14	0.2	0.33	0.8
16	0.3	0.57	1.1
18	0.7	1.36	17.2
20	1.0	1.86	9.1
22	1.4	2.96	18.0
24	2.0	6.18	42.5
26	2.8	11.41	95.2
28	3.1	16.10	116.5
30	4.2	61.39	719.8

Table 2.3: Performance of BMSA-R: Computation Times in CPU seconds

2.4.5 Restarting

It is also interesting to study the impact of restarting the algorithm from scratch, periodically. Table 2.3 reports the quality and performance results for BMSA-R where *maxReheats = 20* but the re-computation is restarted from scratch after that for a number of times. Note that the results are given for all instances with 28 and 30 teams. On the average, BMSA-R slightly dominates BMSA, although some instances are solved faster with BSMA. The quality of the solutions of BMSA and BMSA-R is the same on these instances.

2.4.6 Quality under Strict Time Constraints

It is often the case in sport scheduling that users like very fast interactions with the scheduler to find and evaluate different schedules quickly (e.g., [35]). It is thus interesting to see how BMSA behaves when the CPU time is limited. Tables 2.4, 2.5, and 2.6 depict the results for large number teams ($n = 24, 26, 28$) under different time constraints, where the CPU time is limited to at most $1, 2, 5, 10, \ldots$ seconds. Interestingly, for 24, 26, and 28 teams, BMSA finds near-optimal results in about 10 seconds. This clearly shows that BMSA outperforms the approximation algorithm in [34], both in quality and efficiency.

$n = 24$	BMSA under Restricted CPU times		
sec	Minimum	Average	Maximum
1	138	160.38	170
2	102	125.31	140
5	66	75.02	82
10	66	74.02	78
20	66	73.80	78
50	66	73.63	78
100	66	73.60	78

Table 2.4: Quality of BMSA with Limited CPU Time: n=24

$n = 26$	BMSA under Restricted CPU times		
sec	Minimum	Average	Maximum
1	192	206.28	216
2	142	169.35	184
5	82	95.54	110
10	82	85.74	92
20	82	85.32	92
50	82	85.08	90
100	82	85.00	90

Table 2.5: Quality of BMSA when Limited CPU Time: n=26

$n = 28$	BMSA under Restricted CPU times		
sec	Minimum	Average	Maximum
1	230	251.85	266
2	198	217.77	230
5	98	131.11	148
10	92	98.84	108
20	92	98.62	106
50	92	98.17	106
100	92	98.05	106
200	92	98.02	106

Table 2.6: Quality of BMSA when Limited CPU Time: n=28

Chapter 3

The Traveling Tournament Problem

In this chapter, we give a detailed account of our simulated annealing approach to the Traveling Tournament Problem, both in the original version of the problem, introduced by Easton et.al. in [49, 15], and in variants of the problem that have been proposed over the years (and can also be found in [49].)

Our first approach to the TTP was presented at the CP'AI'OR'03 workshop [3]. A more complete version of that work can be found in [4]. In CP'AI'OR'06 [54] we used an enhanced version of the scheme presented in [4] to deal with different TTP variants.

Our main simulated annealing algorithm, TTSA, uses a large neighborhood with complex moves, and includes advanced techniques, such as strategic oscillation and reheats to balance the exploration of feasible and infeasible regions and to escape local minima at very low temperatures. Our solutions match the best-known solutions on small instances, and significantly improve over previous approaches on most of the larger instances. Moreover, TTSA is shown to be robust, in terms of worst solution quality over a large number (~ 50) of independent runs.

This chapter provides a more comprehensive coverage of our work in [3, 4, 54], and also explores, in a detailed fashion, issues not previously addressed in our work.

3.1 The Basic TTP Problem

The Traveling Tournament Problem was introduced by Easton, Nemhauser and Trick [49, 15], which contains many interesting discussions on Sport Scheduling. An input consists of n teams (n even) and an $n \times n$ symmetric matrix d, such that d_{ij} represents the distance between the homes of teams T_i and T_j. A solution is a schedule in which each team plays with each other twice, once in each team's home. Such a schedule is called a *double round-robin tournament*. It should be clear that a double round-robin tournament has $2n - 2$ rounds. It turns out that tournaments with $2n - 2$ rounds can

be constructed for every n and we only consider tournaments with this minimal number of rounds. In such tournaments, the number of games per round is always n.

For a given schedule S, the cost of a team is the total distance that it has to travel starting from its home, playing the scheduled games in S, and returning back home. The cost of a solution is defined as the sum of the cost of every team.

The goal is to find a schedule with minimum cost satisfying the following two constraints:

1. **Atmost Constraints:** No more than three consecutive home or away games are allowed for any team.

2. **Norepeat Constraints:** A game of T_i at T_j's home cannot be followed by a game of T_j at T_i's home.

Thus, a double round-robin schedule is feasible if it satisfies the atmost and norepeat constraints and is infeasible otherwise.

In this work, a schedule is represented by a table indicating the opponents of the teams. Each line corresponds to a team and each column corresponds to a round. The opponent of team T_i at round r_k is given by the absolute value of element (i, k). If (i, k) is positive, the game takes place at T_i's home, otherwise at T_i's opponent home. Consider, for instance, the schedule S for 6 teams (and thus 10 rounds).

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

Schedule S specifies that team T_1 has the following schedule. It successively plays against teams T_6 at home, T_2 away, T_4 at home, T_3 at home, T_5 away, T_4 away, T_3 away, T_5 at home, T_2 at home, T_6 away. The travel cost of team T_1 is

$$d_{12} + d_{21} + d_{15} + d_{54} + d_{43} + d_{31} + d_{16} + d_{61}. \tag{3.1}$$

Observe that long stretches of games at home do not contribute to the travel cost but are limited by the atmost constraints. This kind of tension is precisely why this problem is hard to solve in practice.

3.1.1 Overall Design of Local Search

This thesis proposes an advanced simulated annealing algorithm (TTSA) for the TTP. As usual, the algorithm starts from an initial configuration. Its basic step moves from the current configuration c to a configuration in the neighborhood of c. TTSA is based on four main design decisions:

1. Constraints are separated into two groups: hard constraints, which are always satisfied by the configurations, and soft constraints, which may or may not be satisfied. The hard constraints are the round-robin constraints, while the soft constraints are the norepeat and atmost constraints. In other words, each configuration in the search represents a double round-robin tournament, which may or may not violate the norepeat and atmost constraints. Exploring the infeasible region seems to be particularly important for this problem. Obviously, to drive the search toward feasible solutions, TTSA modifies the original objective function to include a penalty term.

2. TTSA is based on a large neighborhood of size $O(n^3)$, where n is the number of teams. In addition, these moves may affect significant portions of the configurations. For instance, they may swap the schedule of two teams, which affects $4(n-2)$ entries in a configuration. In addition, some of these moves can be regarded as a form of ejection chains which is often used in tabu search [28, 40].

3. TTSA dynamically adjusts the objective function to balance the time spent in the feasible and infeasible regions. This adjustment resembles the strategic oscillation idea [20] successfully in tabu search to solve generalized assignment problems [13], although the details differ since simulated annealing is used as the meta-heuristics.

4. TTSA also uses reheats (e.g., [9]) to escape local minima at low temperatures. The "reheats" increase the temperature again and divide the search in several phases.

We next explore some of these aspects in more detail. Since configurations are double round-robin tournaments, they are called schedules in the following.

3.1.2 Initial Solutions

The algorithm to generate random schedules satisfying the hard constraints is depicted in Figure 3.1. The algorithm uses a set Q containing all possible ⟨Team,Week⟩ pairs necessary for defining a complete round-robin schedule. The set Q is initialized in line 2, before calling the recursive procedure GENERATESCHEDULE.

This procedure returns true (and the schedule S) whenever Q is empty (lines 7-9). Otherwise, it selects the pair $\langle t, w\rangle$, which is lexicographically smallest in Q (line 10), and attempts to assign a value to table entry $S[t, w]$, by considering all possible choices in random order (lines 11-12). The set of possible choices contains all teams $t' \neq t$, both at home and away (in other words, all values t' and $-t'$, for $t' \neq t$).

If there is no scheduled game for the selected opponent $|o|$ in week w, then the schedule S is updated and the algorithm is called recursively with the set Q where $\langle t, w\rangle$ and $\langle |o|, w\rangle$ have been removed (line 20). If no value can be assigned to $S[t, w]$, then the procedure returns false (line 25). This procedure is very simple and can be improved considerably; however, it appears sufficient for finding schedules satisfying the hard constraints reasonably fast, for $n \leq 16$.

```
1.   function RandomSchedule() {
2.       Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks}
3.       generateSchedule(Q, S)
4.       return S
5.   }

6.   function generateSchedule(Q, S) {
7.       if Q = ∅ then
8.           return true
9.       end if
10.      select ⟨t, w⟩ ∈ Q such that ∀⟨t′, w′⟩ ∈ Q : ⟨t′, w′⟩ ≥ ⟨t, w⟩
11.      Choices ← {1, −1, ..., t − 1, −(t − 1), t + 1, −(t + 1), ..., n, −n}
12.      forall o ∈ Choices in random order do
13.          if ⟨|o|, w⟩ ∉ Q then
14.              S[t, w] ← o
15.              if o > 0 then
16.                  S[o, w] ← −t
17.              else
18.                  S[−o, w] ← t
19.              end if
20.              if generateSchedule(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
21.                  return true
22.              end if
23.          end if
24.      end forall
25.      return false
26.  }
```

Figure 3.1: Generation of Random Initial Schedules

3.1.3 The Neighborhood

The neighborhood of a schedule S is the set of the (possibly infeasible) schedules which can be obtained by applying one of five types of moves. The first three types of moves have a simple intuitive meaning, while the last two generalize them.

***SwapHomes*(S, T_i, T_j)** This move swaps the home/away roles of teams T_i and T_j. In other words, if team T_i plays home against team T_j at round r_k, and away against T_j's home at round l, *SwapHomes*(S, T_i, T_j) is the same schedule as S, except that now team T_i plays away against team T_j at round r_k, and home against T_j at round r_l. There are $O(n^2)$ such moves. Consider the schedule S:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move *SwapHomes*(S, T_2, T_4) produces the schedule:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	-4	3	6	4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	2	1	5	-2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

SwapRounds(S, r_k, r_l) The move simply swaps rounds r_k and r_l. There are also $O(n^2)$ such moves. Consider the schedule S:

T\R	1	2	**3**	4	**5**	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move $SwapRounds(S, r_3, r_5)$ produces the schedule

T\R	1	2	**3**	4	**5**	6	7	8	9	10
1	6	-2	-5	3	4	-4	-3	5	2	-6
2	5	1	4	-6	-3	3	6	-4	-1	-5
3	-4	5	6	-1	2	-2	1	-6	-5	4
4	3	6	-2	-5	-1	1	5	2	-6	-3
5	-2	-3	1	4	6	-6	-4	-1	3	2
6	-1	-4	-3	2	-5	5	-2	3	4	1

SwapTeams(S, T_i, T_j) This move swaps the schedule of Teams T_i and T_j (except, of course, when they play against each other). There are $O(n^2)$ such moves again. Consider the schedule S:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move $SwapTeams(S, T_2, T_5)$ produces the schedule

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-5	4	3	-2	-4	-3	2	5	-6
2	5	-3	6	4	1	-6	-4	-1	3	-5
3	-4	2	5	-1	6	-5	1	-6	-2	4
4	3	6	-1	-2	-5	1	2	5	-6	-3
5	-2	1	-3	-6	4	3	6	-4	-1	2
6	-1	-4	-2	5	-3	2	-5	3	4	1

Note that, in addition to the changes in lines 2 and 5, the corresponding lines of the opponents of T_i and T_j must be changed as well. As a consequence, there are four values per round (column) that are changed (except when T_i and T_j meet).

It turns out that these three moves are not sufficient for exploring the entire search space and, as a consequence, they lead to suboptimal solutions for large instances. To improve these results, it is important to consider two more general moves. Although these moves do not have the apparent interpretation of the first three, they are similar in structure and they significantly enlarge the neighborhood, resulting to a more connected search space. More precisely, these moves are partial swaps: they swap a subset of the schedule in rounds r_i and r_j or a subset of the schedule for teams T_i and T_j. The benefits from these moves come from the fact that they are not as global as the "macro"-moves *SwapTeams* and *SwapRounds*. As a consequence, they may achieve a better tradeoff between feasibility and optimality by improving feasibility in one part of the schedule, while not breaking feasibility in another one. They are also more "global" than the "micro"-moves *SwapHomes*.

***PartialSwapRounds*(S, T_i, r_k, r_l):** This move considers team T_i and swaps its games at rounds r_k and r_l. Then the rest of the schedule for rounds r_k and r_l is updated (in a deterministic way) to produce a double round-robin tournament. Consider the schedule S

T\R	1	**2**	3	4	5	6	7	8	**9**	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

and the move *PartialSwapRounds*(S, T_2, r_2, r_9). Obviously, swapping the game in rounds r_2 and r_9 would not lead to a round-robin tournament. It is also necessary to swap the games of team 1, 4, and 6 in order to obtain:

T\R	1	**2**	3	4	5	6	7	8	**9**	10
1	6	4	2	3	-5	-4	-3	5	-2	-6
2	5	-6	-1	-5	4	3	6	-4	1	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	-1	-3	-6	-2	1	5	2	6	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	2	-5	4	-3	5	-2	3	-4	1

This move, and the next one, can thus be regarded as a form of ejection chain [28, 40].

Finding which games to swap is not difficult: it suffices to find the connected component which contains the games of T_i in rounds r_k and r_l in the graph where the vertices are the teams and where

an edge contains two teams if they play against each other in rounds r_k and r_l. All the teams in this component must have their games swapped. Note that there are $O(n^3)$ such moves.

PartialSwapTeams(S, T_i, T_j, r_k) This move considers round r_k and swaps the games of teams T_i and T_j. Then, the rest of the schedule for teams T_i and T_j (and their opponents) is updated to produce a double round-robin tournament. Note that, as was the case with *SwapTeams*, four entries are affected for each round considered. There are also $O(n^3)$ such moves. Consider the schedule S

T\R	1	2	3	4	5	6	7	8	**9**	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

The move *PartialSwapTeams*(S, T_2, T_4, r_9) produces the schedule

T\R	1	2	3	4	5	6	7	8	**9**	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

3.1.4 Simulated Annealing

TTSA uses a simulating annealing meta-heuristic to explore the neighborhood graph [26]. TTSA starts from a random initial schedule (which can be obtained through the use of RandomSchedule, for example). TTSA then follows the traditional simulating algorithm scheme. Given a temperature T, the algorithm randomly selects one of the moves in the neighborhood and computes the variation Δ in the objective function produced by the move. If $\Delta < 0$, TTSA applies the move. Otherwise, it applies the move with probability $\exp(-\Delta/T)$.

As typical in simulated annealing, the probability of accepting a non-improving move decreases over time. This behavior is obtained by decreasing the temperature as follows. TTSA uses a variable *counter* which is incremented for each non-improving move and reset to zero when the best solution found so far is improved. When *counter* reaches a particular upper limit, the temperature is updated to $T \cdot \beta$ (where β is a fixed constant smaller than 1) and *counter* is reset to zero.

Figure 3.2 depicts the simulated annealing algorithm in some more detail. The algorithm keeps an implicit representation of the neighborhood as a set of pairs and triplets, since all moves can be

```
1.    function TTSA-BASIC(S, T, w) {
2.        find random schedule S
3.        bestSoFar ← cost(S)
4.        counter ← 0
5.        while phase ≤ maxP do
6.            phase ← 0
7.            counter ← 0
8.            while counter ≤ maxC do
9.                let S' a random schedule in neighborhood(S)
10.               if cost(S') < cost(S) then
11.                   accept ←true
12.               else
13.                   accept ← true with probability exp(−Δ/T),
14.                            false otherwise
15.               end if
16.               if accept then
17.                   S ← S'
18.                   if cost(S') < bestSoFar then
19.                       counter ← 0; phase ← 0
20.                       bestSoFar ← cost(S')
21.                   else
22.                       counter++
23.                   end if
24.               end if
25.           end while
26.           phase++
27.           T ← T · β;
28.       end while
29.   }
```

Figure 3.2: The Basic Simulated Annealing Algorithm

characterized this way. For instance, the *partialSwapTeam*(S, T_i, T_j, r_k) are characterized by triplets of the form $\langle T_i, T_j, r_k \rangle$. Note that a Metropolis algorithm can be obtained by removing line 27 which updates the temperature.

The algorithm, as presented in Figure 3.2, only provides the basic structure of the TTSA algorithm and doesn't show how constraints are handled, or other advanced features, such as reheats or strategic oscillation. In the following paragraphs, after spending some time explaining those features, we are going to give the pseudo-code for the full version of TTSA.

3.1.5 The Objective Function

As mentioned earlier, the configurations in algorithm TTSA are schedules which may or may not satisfy the norepeat and atmost constraints. In addition, the moves are not guaranteed to maintain feasibility even if they start with a feasible schedule. The ability to explore infeasible schedules appears critical for the success of simulated annealing on the TTP.

To drive toward feasible solution, the standard objective function function *cost* is replaced by a more complex objective function which combines travel distances and the number of violations. The new objective function C is defined as follows:

$$C(S) = \begin{cases} cost(S), \text{ if } S \text{ is feasible,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2}, \text{ otherwise} \end{cases} \tag{3.2}$$

where $nbv(S)$ denotes the number of violations of the norepeat and atmost constraints, w is a weight, and $f : N \to N$ is a sub-linear function such that $f(1) = 1$.

It is interesting to give the rationale behind the choice of f. The intuition is that the first violation costs more than subsequent ones, since adding 1 violation to a schedule with 6 existing ones does not make much difference. More precisely, crossing the feasible/infeasible boundary costs w, while v violations only cost $wf(v)$, where $f(v)$ is sub-linear in v. In our experiments, we chose

$$f(v) = 1 + \sqrt{v} \ln v/2 \tag{3.3}$$

This choice ensures that f does not grow too slowly to avoid solutions with too many violations.

Of course, it should be clear that TTSA applies the simulated annealing meta-heuristic, not on the travel distance function *cost*, but on the function C. Also, TTSA must keep track of the best feasible solution found so far.

3.1.6 Strategic Oscillation

TTSA also includes a strategic oscillation strategy which has been often used in tabu search when the local search explores both the feasible and infeasible region (e.g., [20, 13]). The key idea is to vary the weight parameter w during the search. In advanced tabu-search applications (e.g., [13]), the penalty is updated according to the frequencies of feasible and infeasible configurations in the last iterations. Such a strategy is meaningful in that context, but is not particularly appropriate for simulated annealing since very few moves may be selected. TTSA uses a very simple scheme. Each time it generates a new best solution (line 28 of the algorithm), TTSA multiplies w by some constant $\delta > 1$, if the new solution is infeasible, or divide w by some constant $\theta > 1$, if the new solution is feasible.

The rationale here is to keep a balance between the time spent exploring the feasible region and the time spent exploring infeasible schedules. After having spent a long time in the infeasible region, the weight w, and thus the penalty for violations, will become large and it will drive the search toward feasible solutions. Similarly, after having spent a long time in the feasible region, the weight w, and thus the penalty for violations, will become small and it will drive the search toward infeasible solutions. In our experiments, we chose $\delta = \theta$ for simplicity.

3.1.7 Reheats

The last feature of TTSA is the use of reheating, a generalization of the standard simulated annealing cooling schemes that has been proposed by several authors (see, for example, [38]). The basic idea

is that, once simulated annealing reaches very low temperatures, it has difficulties escaping from local minima, because the probability of accepting non-decreasing moves is very low. Reheating is the idea of increasing the temperature again to escape the current local minimum.

TTSA uses a relatively simple reheating method. The idea is to reheat upon completion of the outermost loop by increasing the temperature to a constant γ (typically $\gamma = 2$) times its value when the best solution was found. TTSA now terminates when the number of consecutive reheats, without improving the best solution, reaches a given limit. The algorithm including all these modifications is shown in Figure 3.3.

3.2 Experimental Results on the Basic TTP

This section describes the experimental results on TTSA. It first reports the results for the standard version, which aims at producing the best possible schedules. The impact of the various components is then studied in detail. The next set of experimental results describes how the solution quality evolves over time. The section continues with a discussion on a fast cooling version of TTSA, which improves TTSA's ability to find good solutions quickly. Section 3.2 concludes with the best solutions found using the standard version of TTSA, over the course of this research.

3.2.1 Quality and Performance of TTSA

TTSA was applied to the National League benchmark described in [15, 49] (we did not consider $n = 6$, since TTSA always finds the optimal solution). We experimented with different values for the parameters described previously. The most successful version of the algorithm uses a very slowly cooling system ($\beta \simeq .9999$), a large number of phases (so that the system can reach low temperatures), and long phases. In order to avoid big oscillations in the value of the penalty weight w, the parameters δ and θ were chosen to be close to 1 ($\simeq 1.03$). For each instance set (i.e., for every value of n), in all 50 runs, the parameters had the same initial values.

Table 3.1 describes the quality of the results for 50 runs of TTSA on each of the instances. The first column gives the number of teams, the second column gives the best known solutions at the time of writing our CP-AI-OR'03 paper [3] (January 12, 2003), as shown in [49]. These solutions are obtained using a variety of techniques, including constraint and integer programming, and Lagrangian relaxation. It is not clear, however, how some of these, and newer, results have been obtained. See [49] for more details. The next columns give the best solution found over the 50 runs, the worst solution, the average quality of the solutions, and the standard deviation.

Table 3.2 gives the CPU time in seconds needed by TTSA for producing these results on an AMD Athlon(TM) at 1544 MHz. It gives the time to find the best solution, T for min, the average time, mean(T), and the standard deviation over 50 runs, std(T). The parameter values for obtaining the results are shown on Table 3.3.

```
1.  function TTSA(S, T, w) {
2.      S ← RandomSchedule()
3.      bestFeasible ← ∞; nbf ← ∞
4.      bestInfeasible ← ∞; nbi ← ∞
5.      reheat ← 0; counter ← 0
6.      while reheat ≤ maxR do
7.          phase ← 0
8.          while phase ≤ maxP do
9.              counter ← 0
10.             while counter ≤ maxC do
11.                 let S' a random schedule in neighborhood(S)
12.                 if C(S') < C(S) or
13.                    nbv(S') == 0 and C(S') < bestFeasible or
14.                    nbv(S') > 0 and C(S') < bestInfeasible
15.                 then
16.                    accept ← true
17.                 else
18.                    accept ← true with probability exp(−ΔC/T),
19.                             false otherwise
20.                 end if
21.                 if accept then
22.                     S ← S'
23.                     if nbv(S) == 0 then
24.                         nbf ← min(C(S), bestFeasible)
25.                     else
26.                         nbi ← min(C(S), bestInfeasible)
27.                     end if
28.                     if nbf < bestFeasible or nbi < bestInfeasible then
29.                         reheat ← 0; counter ← 0; phase ← 0
30.                         bestTemperature ← T
31.                         bestFeasible ← nbf
32.                         bestInfeasible ← nbi
33.                         if nbv(S) == 0 then
34.                             w ← w/θ
35.                         else
36.                             w ← w · δ
37.                         end if
34.                     else
35.                         counter++
36.                     end if
37.                 end while
38.             phase++
39.             T ← T · β
40.         end while
41.         reheat++
42.         T ← γ · bestTemperature
43.     end while
44. }
```

Figure 3.3: The Simulated Annealing Algorithm (TTSA)

n	Best (Nov. 2002)	min(D)	max(D)	mean(D)	std(D)
8	39721	39721	39721	39721	0
10	61608	**59583**	**59806**	**59605.96**	53.36
12	118955	**112800**	**114946**	**113853.00**	467.91
14	205894	**190368**	**195456**	**192931.86**	1188.08
16	281660	**267194**	**280925**	**275015.88**	2488.02

Table 3.1: Solution Quality of TTSA on the TTP

n	T for min	mean(T)	std(T)
8	596.6	1639.33	332.38
10	8084.2	40268.62	45890.30
12	28526.0	68505.26	63455.32
14	418358.2	233578.35	179176.59
16	344633.4	192086.55	149711.85

Table 3.2: Computation Times of TTSA on the TTP

n	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
8	400	0.9999	4000	1.04	1.04	5000	7100	10	2
10	400	0.9999	6000	1.04	1.04	5000	7100	10	2
12	600	0.9995	10000	1.03	1.03	4000	1385	50	1.6
14	600	0.9999	20000	1.03	1.03	4000	7100	30	1.8
16	700	0.9999	60000	1.05	1.05	10000	7100	50	2

Table 3.3: Parameter Values for the TTSA Instances

Method	Best	Worst	Mean	Std
TTSA	190,514	196,989	194,560	1,631
TTSA(PS)	191,145	197,383	194,694	1,304
TTSA(NR)	196,561	205,094	200,680	2,152
TTSA(150)	197,781	211,347	203,621	3,157
TTSA(300)	195,627	202,158	198,004	964
TTSA(450)	204,872	215,485	206,862	1,428
TTSA(600)	213,938	218,879	216,412	1,096

Table 3.4: Impact of TTSA Components on Solution Quality (14 Teams)

Method	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
TTSA	1100	0.999	18000	1.03	1.03	3000	710	1000	1.4
TTSA(PS)	1100	0.999	18000	1.03	1.03	3000	710	1000	1.4
TTSA(NR)	1100	0.9999	18000	1.03	1.03	3000000	∞	0	1
TTSA(150)	150	1	18000	1.03	1.03	∞	0	0	1
TTSA(300)	300	1	18000	1.03	1.03	∞	0	0	1
TTSA(450)	450	1	18000	1.03	1.03	∞	0	0	1
TTSA(600)	600	1	18000	1.03	1.03	∞	0	0	1

Table 3.5: Parameter Values for Experiments on the Impact of the Components (14 Teams)

TTSA improved all the best-known solutions (at the time of the experiments) on instances with at least 10 teams. TTSA was the first algorithm to go lower than 60,000 on 10 teams, 190,000 for 14 teams and 270,000 for 16 teams. The improvements ranged between 2% to 5%. The table also shows that the worst solution of TTSA was always smaller than or equal to the best-known solution, indicating the robustness of TTSA.

3.2.2 Impact of the Components

TTSA includes a variety of components and it is interesting to measure how important they are in the performance and quality of the algorithm. Table 3.4 compares various versions of the algorithm on 14 teams with the parameters shown on table 3.5. Each version leaves out some component of TTSA: TTSA(PS) considers partial moves only, TTSA(NR) does not include reheats, and the TTSA(T) versions are not based on simulated annealing but on a Metropolis algorithm with temperature T. All versions were executed at least 35 times, for 100,000 seconds each time. The table reports the minimum, maximum, and mean solution values, as well as the standard deviation. It is of interest to observe that considering only one of the partial moves degrades solution quality. It was apparent early on in our research that both moves bring benefits, since most of our best solutions were obtained when we added *PartialSwapTeams*.

It is interesting to observe that TTSA outperforms all other versions on these experiments.

Time	Method	Best	Worst	Mean	Std
50,000 sec	TTSA	192,040	198,140	195,349	1,311
	TTSA(PS)	193,144	202,435	196,112	1,755
100,000 sec	TTSA	190,514	196,989	194,560	1,631
	TTSA(PS)	191,145	197,383	194,694	1,304
150,000 sec	TTSA	190,514	196,989	194,186	1,550
	TTSA(PS)	191,060	196,665	194,300	1,289

Table 3.6: Impact of Full Moves on the Solution Quality of TTSA (14 Teams)

TTSA(PS) is slightly outperformed by TTSA, although the full moves can be thought as a combination of partial moves. The full moves seem to bring some benefit because of their ability to diversify the schedule more substantially. The use of reheats produce significant benefits. The performance of the algorithm degrades significantly when they are not used, raising the mean from about 194,000 to about 200,000. Similar observations hold for the Metropolis version which are largely dominated in general.

Since the results with and without full moves were rather close, another set of experiments was carried out to understand their effect more precisely. These results are shown in Table 3.6 which evaluates TTSA and TTSA(PS) when the time limit is varied. Interestingly, the results seem to indicate that full swaps are beneficial early in the search and become marginal when long runs are considered.

3.2.3 Solution Quality over Time

As earlier results demonstrate, TTSA is computationally intensive, at least to find very high-quality solutions . It is thus important to study how solution quality evolves over time in TTSA. Figures 3.4 and 3.5 depict the solution values over time for many runs with 12 and 14 teams. The figures depict the superposition of the curves for many runs.

It is interesting to observe the sharp initial decline in the solution values which is followed by a long tail where improvements are very slow. Moreover, at the transition point between the decline and the tail, TTSA typically has improved the previous best solutions. In particular, TTSA takes about 1,000 seconds to beat the previous best results for 12 teams, after which improvements proceed at a much slower rate. The same phenomenon arises for 14 teams.

3.2.4 Fast Cooling

As mentioned earlier, the parameters of TTSA were chosen to find the best possible solutions without much concern about running times. The experimental results indicate that TTSA generally exhibits a sharp initial decline followed by a long tail. Hence it is intriguing to evaluate the potential of TTSA to find “good” solutions quickly by using a fast cooling.

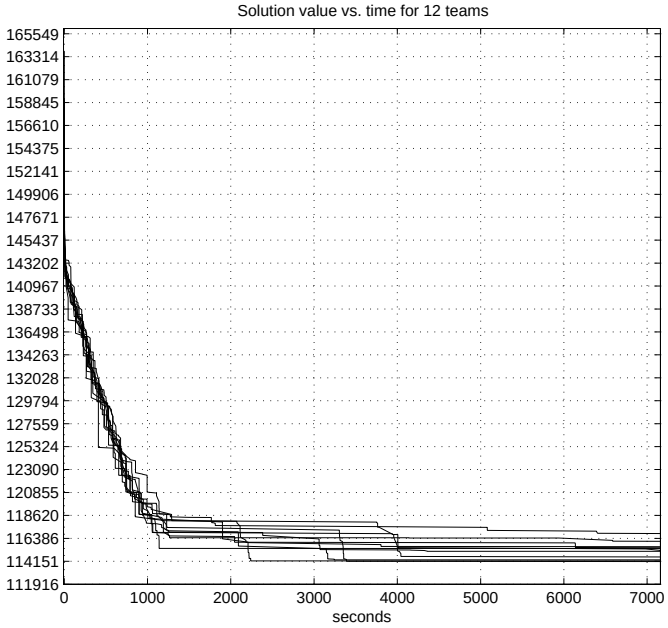


Figure 3.4: Solution Quality over Time for 12 Teams

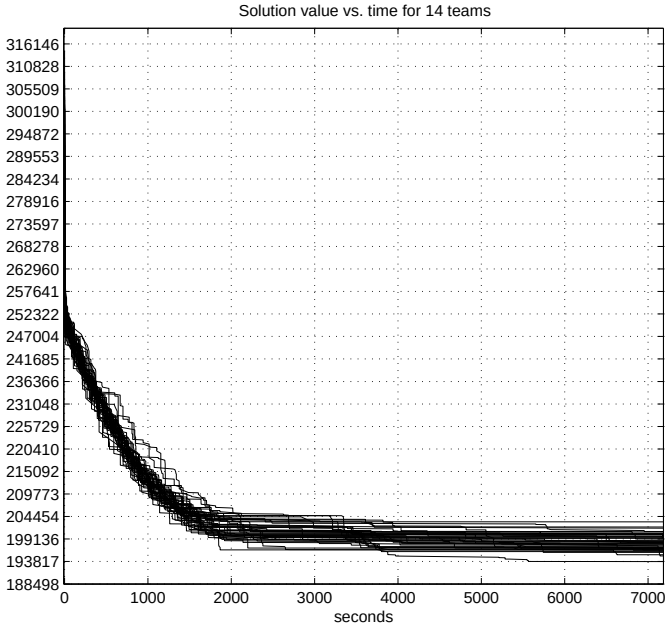


Figure 3.5: Solution Quality over Time for 14 Teams

Method	Best	Worst	Mean	Std
TTSA	282,948	331,014	312,102	7,200
TTSA(FC)	277,626	295,299	286,527	4,125

Table 3.7: Solution Quality of TTSA and TTSA(FC) within 2 Hours on 16 Teams

Method	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
TTSA	700	0.9999	60000	1.05	1.05	10000	7100	50	2
TTSA(FC)	700	0.98	60000	1.05	1.05	5000	70	10000	2

Table 3.8: Parameter Values for Experiments on Fast Cooling (16 Teams)

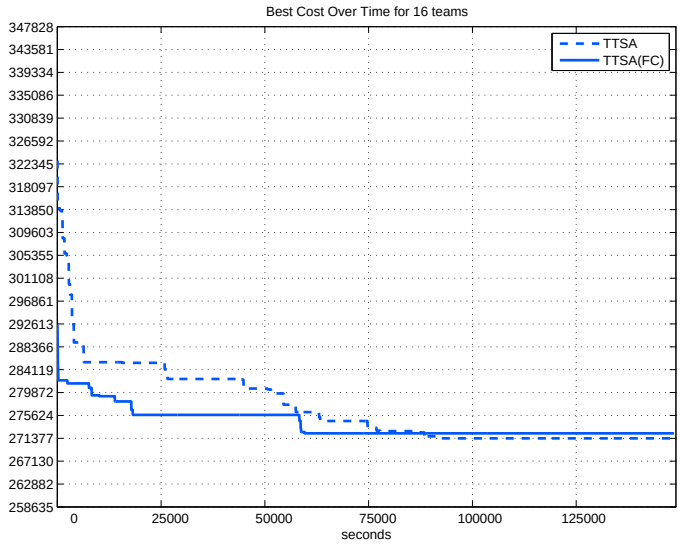


Figure 3.6: Solution Quality over Time for TTSA and TTSA(FC)

Table 3.7 depicts the results of TTSA(FC), a fast cooling version of TTSA, with parameters shown in Table 3.8. TTSA(FC) uses a cooling factor (beta) of .98, phases of length (maxC) 5000 and a number of non-improving phases before reheating (maxP) of 70. Table 3.7 compares TTSA and TTSA(FC) on the 16 teams instance for a running time of 2 hours. The results clearly show the benefits of a fast cooling schedule for obtaining high-quality solutions quickly. Within two hours, the best run of TTSA(FC) outperforms the previous best solution for 16 teams, while its average solution is less than 2% above this solution.

Figure 3.6 depicts the solution quality over time, for the best runs of TTSA and TTSA(FC). Observe the sharper initial decline of TTSA(FC), which significantly outperforms TTSA for short runs. Of course, over time, TTSA(FC) is slightly dominated by TTSA. These results seem to indicate the versatility of TTSA and its potential to find high-quality solutions reasonably fast.

n	Nov 2002	Jun 2003	Apr 2004	May 2004	Sep 2005	Apr 2006	May 2006
8	39721 (E)	39721 (E)	39721 (E)	39721 (E)	39721 (E)	39721 (E)	39721 (E)
10	61608 (Z)	**59583**	**59583**	**59583**	59436 (L)	59436 (L)	59436 (L)
12	118955 (C)	**112800**	112298 (L)	**111248**	**111248**	**111248**	**111248**
14	205894 (C)	**190368**	190056 (L)	**189766**	**189766**	189759 (D)	**189156**
16	281660 (S)	**267194**	**267194**	**267194**	**267194**	**267194**	**267194**

Table 3.9: Timeline of Best-Known Solutions: TTSA solutions are shown in bold face

n	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
12	158	1	10000	1.03	1.03	∞	0	0	1
14	193	1	18000	1.03	1.03	∞	0	0	1

Table 3.10: TTSA Parameter Values used in our May 2004 solutions ($n = 12, 14$)

3.2.5 Best Solutions Since the Beginning of this Research

Table 3.9 reports the evolution of the best-known solutions on the standard NLB instances, since we started this research in November 2002. The table summarizes the history of updates to the best-known upper bounds. Each column corresponds to a time point where an improvement took place on one or more instances, and shows the best-known upper bounds at that time. TTSA solutions are shown in bold face. Other solutions are followed by a letter in parenthesis referring to the corresponding authors, according to the following:

- E: Easton, Nemhauser, and Trick [15]
- Z: Lim, Rodrigues, and X. Zhang [30]
- C: Cardemil and Durán [7]
- S: Shen and H. Zhang [46]
- L: Langford [49]
- D: Dorrepaal and Chackman [49]

Our May 2004 solutions for $n = 12$ and $n = 14$, were produced by running TTSA at a fixed temperature, as shown in Table 3.10, which shows the parameters for each instance. The fixed temperature in each case was determined in the following way. For every n, we ran 50 experiments starting from different random points, all with the same parameters used in obtaining our previous best solution for that n. Then, we computed the average over the 50 experiments of the temperature at which the best solution was found for every experiment. This average gave us the starting temperature used in obtaining these results.

For $n = 16$, although we did not improve the best-known solution using this approach, the results were very close to the value of the best-known solution. In particular, after running 50 experiments, we got a minimum cost of 268137 and a maximum cost of 272376. This is very interesting, considering

the following: Until June 2005, when Di Gaspero and Schaerf [11] gave a solution of value 270063, the best-known solution not produced with TTSA was still worse than that maximum cost (the closest to it was by Langford, who gave a solution of value 272902 in January 2004).

Note that our 189156 solution for NLB14 was produced using a more enhanced version of TTSA (although still following the same framework), which will be presented in Section 3.4. Also, note that, using the population-based simulation annealing framework presented in Chapter 4, we have recently been able to produce further improvements to the best-known upper bounds for instances NLB12, NLB14 and NLB16.

3.3 Comparison with Break Minimization

At this point, it might be interesting to make some remarks on the neighborhood structure used in the TTP and the Break Minimization Problem. Observe that, contrary to simpler problems such as Break Minimization [53], we found it critical to explore the infeasible region in the TTP and to consider moves that significantly alter the schedules. In our experiments with the TTP, neighborhoods consisting of simpler moves or considering only feasible schedules tend to produce local minima of low quality. Of course, this does not necessarily mean that such neighborhoods do not exist; however, it appears to be very difficult to design an efficient neighborhood, without including the infeasible region. This difference between the TTP and the Break Minimization Problem stems from the fact that Break Minimization assumes a fixed schedule: only the home/away patterns must be determined. In contrast, the objective function in the TTP must not only determine the home/away patterns; it must also determine the schedule of each team to minimize the total travel distance. This additional difficulty, together the tension with the feasibility constraints it produces, is what makes the TTP particularly challenging.

3.4 Variants of the TTP

The Traveling Tournament Problem, as originally introduced in [15], was first studied only for benchmarks based on the distances between cities of the 16 teams of National League Baseball. The solution space consisted of all double round-robin schedules and did not include mirroring. In recent years, increased attention to the problem has led to the study of a variety of variants. On one hand, the problem was studied under the additional constraint that the schedule be a *mirrored* double round-robin tournament. On the other hand, alternative sets of distances between cities were considered. Furthermore, the problem has been studied for larger number of teams (up to 24). This section describes the TTP in its different variants.

On the original TTP instances, our original simulated algorithm has remained most effective in producing best-known solutions, but it had not been applied to the variants proposed subsequently. However, these variants may fundamentally alter the combinatorial structure of the problem. For instance, with constant distances, the order of the games in a sequence of away games is irrelevant,

which is obviously not the case with the original distances. Similarly, mirroring requires that the second half of the schedule be similar to the first half but with the home-away patterns of the games reversed. Mirroring imposes severe feasibility constraints on the schedule, making it harder to find feasible tournaments and to remain in the feasible region. As a result, it was not clear at all that the algorithm would scale and perform effectively on the entire spectrum of TTP instances.

Our recent paper in CP-AI-OR 2006 [54], originated as an attempt to determine the effectiveness and limitations of our algorithm across all TTP instances. From a practical standpoint, the paper's main contribution was to show that the original algorithm can be enhanced to be effective across all distance metrics and mirroring. More precisely, the enhanced algorithm matched or improved most of the best-known solutions for all variants and it also produced numerous new best solutions for many of the variants. From a technical standpoint, the research led to new insights into the nature of the TTP and to the following contributions:

- It showed that the algorithm can smoothly handle mirroring constraints as soft constraints by including new neighborhood moves that preserve the mirroring structure of the candidate tournament;
- It showed that the algorithm can successfully accommodate all distance metrics by including new neighborhood moves that preserve the distance structure of the candidate tournament;
- It showed how to refine the original strategic oscillation scheme to the instances where feasibility constraints are much stronger.

In what follows, we are going to see these variants in more detail, and describe the enhancements introduced to the original TTSA algorithm, in order to deal with the additional cases. The presentation will assume that the original TTSA is the starting point, and will only describe the points in which the enhanced version differs from the original one. For example, in the case of the neighborhood, the previously defined moves are still used, and the enhancements consist of simply adding new moves to the neighborhood.

3.4.1 Definition of TTP Variants

Mirroring In some sports leagues (e.g., in most European soccer leagues), it is common practice to adopt a two-stage mirrored schedule. If n is the number of participating teams, each stage has $n-1$ rounds and the $n-1$ rounds of the second stage are simply a copy of the the first-stage rounds with a swap of home/away patterns of each game. In terms of the table representation, a schedule is *mirrored* if

$$\forall i, j: \ 1 \leq j \leq n-1: \ S[i, j+n-1] = -S[i, j] \tag{3.4}$$

Distance Metrics All the distance metrics studied in this section can be found on web page [49]. They are defined as follows:

- [NLBn:] For $n = 4, 6, \ldots, 16$, NLBn is the set of distances between the subset of the first n teams of National League Baseball.
- [Circular:] The n teams are labeled with numbers 0 through $n-1$ and are placed on a circle in this order. Then, for any two teams, the distance is given by the length of the shortest arc connecting the teams. The formula for the distance between any i and j is given by the formula:
$$d_{ij} = \min\{i - j, j - i + n\} \tag{3.5}$$
- [Constant:] The n teams are at unit distance to one another, i.e., $d_{ij} = 1$, for all i, j.

For each distance metric, we are interested in solutions to both the mirrored and the non-mirrored version of the TTP.

3.4.2 Handling Mirroring

At this point, we review the enhancements of the algorithm to find mirrored tournaments.

Mirroring Constraints

Mirroring constraints, like atmost and norepeat constraints, are considered soft constraints in the algorithm, since restricting the neighborhood graph to only mirrored schedules was not found effective. In other words, the neighborhood graph consists of nodes representing both mirrored and non-mirrored schedules and it is the role of the objective function to drive the search toward mirrored schedules. More precisely, the algorithm associates a soft mirroring constraint with each entry $S[i, j]$ $(1 \leq i \leq n \;\&\; 1 \leq j < n - 1)$ and the constraint holds when

$$S[i, j] = -S[i, j + n - 1] \tag{3.6}$$

It is thus possible to include the violations of these constraints in the objective function as was the case for the atmost and norepeat constraints.

Unfortunately, this simple modeling is not directly effective given the large number of mirroring constraints that can be violated in candidate schedules. As a result, the algorithm appropriately weighs the mirroring constraints and uses a modified version of the function $nbv(S)$ appearing in the objective function $C(S)$, as defined in Equation 3.2. $nbv(S)$ is rewritten as the sum

$$nbv(S) = nbv_a(S) + nbv_r(S) + \frac{nbv_m(S)}{\mu} \tag{3.7}$$

where nbv_a, nbv_r, and nbv_m represent the violations of the atmost, norepeat, and mirroring constraints respectively, while μ is a weighting factor that depends on the size of the instance. The number of violations of the mirroring constraints is simply defined as

$$nbv_m(S) = |\{(i, j) : S[i, j] \neq -S[i, j + n - 1] \;\&\; 1 \leq i \leq n \;\&\; 1 \leq j \leq n - 1\}| \tag{3.8}$$

Mirrored Moves

Once the algorithm reaches a (possibly infeasible) mirrored schedule, it is beneficial to let the search explored neighboring *mirrored* schedules with fewer violations of the remaining constraints or smaller distances. The simulated-annealing algorithm, with its present moves, has a low probability of exploring such moves. It is thus important to design mirrored versions of the moves that affect the mirroring constraints: the *SwapRounds*, *PartialSwapRounds*, and *PartialSwapTeams* moves. The basic idea behind the aggregate moves is to apply the original moves to both parts of the schedule simultaneously. For instance, the new move $SwapRoundsMirrored(S, r_k, r_l)$ for $1 \leq r_k < r_l \leq n-1$ is the aggregate

$$(SwapRounds(S, r_k, r_l), SwapRounds(S, T_i, r_k + n - 1, r_l + n - 1)).$$

Mirroring constraints are invariant with respect to mirrored moves: in particular, if the schedule is mirrored, it remains so. As a result, the algorithm is able to preserve the structure of the schedule with respect to mirroring constraints, while performing transformations that affect the remaining constraints or the distances.

3.4.3 Handling Different Distance Metrics

This section presents two additional composite moves that affect the distances in interesting ways. Once again, the novel moves aggregate sequences of existing moves that have a low probability of taking place in the original algorithm. Hence, they also preserve some significant structure in the schedule, while performing some interesting transformations.

The key idea underlying the novel moves is to reverse subsequences of away moves. Recall that travel only occurs for successive pairs of away games and for successive pairs of (home,away) and (away,home) games. Thus, by reversing a subsequence of away games, we preserve a significant part of the distance structure, while modifying it in a way that is difficult to achieve by sequences of original moves. In particular, the distances in the reversed subsequence, as well as the distances in the sequence of the other teams that must also be reversed to maintain a double round-robin tournament, remain the same. It is only at the beginning and at the end of the subsequences that distances are changing. In fact, moves similar in spirit are also used in car sequencing [10] but they are simpler since they do not have to account for the round-robin constraints and the distance structure.

The algorithm thus considers moves of the form $ReverseAwayRun(S, T_i, r_k, m)$ where team T_i plays an away sequence from round r_k to round r_{k+m}. The effect of the move is similar to the sequence of $p = (m+1)/2$ moves

$PartialSwapRounds(S, T_i, r_k, r_{k+m})$
$PartialSwapRounds(S, T_i, r_{k+1}, r_{k+m-1})$
...
$PartialSwapRounds(S, T_i, r_{k+p-1}, r_{k+m+1-p})$

For instance, consider the schedule S

T\R	1	2	3	4	5	**6**	7	8	9	10
1	6	2	4	3	-5	-4	-3	-6	-2	5
2	5	-1	-3	-6	4	3	6	-5	1	-4
3	-4	5	2	-1	6	-2	1	4	-5	-6
4	3	6	-1	-5	-2	1	5	-3	-6	2
5	-2	-3	6	4	1	-6	-4	2	3	-1
6	-1	-4	-5	2	-3	5	-2	1	4	3

The move $ReverseAwayRun(S, T_1, r_6, 4)$ produces the schedule S':

T\R	1	2	3	4	5	**6**	7	8	9	10
1	6	2	4	3	-5	-2	-6	-3	-4	5
2	5	-1	-3	-6	4	1	-5	6	3	-4
3	-4	5	2	-1	6	-5	4	1	-2	-6
4	3	6	-1	-5	-2	-6	-3	5	1	2
5	-2	-3	6	4	1	3	2	-4	-6	-1
6	-1	-4	-5	2	-3	4	1	-2	5	3

When $d_{52} + d_{41} < d_{54} + d_{21}$, the move improves the distance with respect to team T_1, without affecting the atmost violations of team T_1 and the distance structure inside the subsequence. There are several points worth highlighting here. First, reversing entire sequences of away games for a team T_i does not change the total distance T_i has to travel and should not be considered. Second, the value m is never very large in the TTP instances, since the atmost constraints drive the search toward small subsequences of away games. Finally, the algorithm must include a mirrored version *ReverseAwayRunMirrored* of the moves since they affect the mirroring constraints.

In both variants, the novel neighborhood moves are in fact sequences of existing moves. As such, they do not improve the connectivity of the neighborhood for the TTP. Their significance comes from the fact that, in the original algorithm, these sequences have a low probability of taking place, although they capture fundamental aspects of the problem structure.

3.4.4 Algorithmic Refinements

Finally, we discuss some additional refinements to the original TTSA algorithm, that were necessary for dealing with the new instance sets. We first describe a small change in the definition of the objective function C using by TTSA, that allows for more flexibility, especially as the number of teams, n, grows. In particular, the sub-linear function $f(v)$ is now more generally defined as

$$f(v) = 1 + (\sqrt{v} \ln v)/\lambda \tag{3.9}$$

(In the original TTSA algorithm, we always used $\lambda = 2$). In our experiments, we chose λ equal to 2 on small instances (up to $n = 16$) and 1 on larger ones. The reason was that larger instances

tended to generate an increased number of violations, and choosing a smaller λ was observed to better control this increase.

Strategic Oscillation

The mirroring constraints make it harder to find feasible tournaments and the search may spend considerable time in the infeasible region, before finding its first feasible solution. As a result, even small values for μ and λ, the strategic oscillation scheme will overly inflate the violation weight w, leading the search to stagnation. To alleviate this pathological case, the algorithm now takes a two-step approach. In a first phase, which lasts until the first feasible tournament is found, no oscillation takes place. In the second phase, the strategic oscillation scheme is activated as before and balances the time spent in the infeasible and feasible regions. Note also the synergy between this scheme and the new neighborhood moves. By including mirrored moves, the algorithm is able to better balance the time it spends in the feasible and infeasible regions in presence of mirroring constraints.

Initial Schedules

The simple backtrack search used in [3] to find initial schedules does not scale well when the number of teams increases, which is the case in the constant and circular variants. As a result, like in [44], the algorithm now uses a randomized version of the hill-climbing algorithm for generating 1-factorizations [14]. The initial schedules generated by this randomized algorithm are more diversified than perturbations of schedules obtained by the polygon algorithm. The algorithm in [14] works for single round-robin schedules but a double round-robin schedule can be obtained by a simple mirroring.

3.5 Experimental Results on TTP Variants

The enhanced version of the TTSA algorithm was run on all the mirrored and non-mirrored instances given on Michael Trick's webpage [49], with the exception of the NFL instances posted in December 2005, shortly before our CP-AI-OR'06 paper [54] was completed. For each instance, 20 experiments were carried out from randomly chosen schedules on an AMD Athlon 64 at 2Ghz. The results are reported in two tables for each variant. The first table reports the best, mean, and worst solutions found by the algorithm, as well as the standard deviation and the best-known solution value at the time of writing. The second table reports the time to reach the best solution, the mean time of each experiment, and the standard deviation. Bold face indicates improved results. It is also important to mention that many authors (e.g., [11, 12, 30, 44]) now use the neighborhood we originally proposed in [3] which makes it much more difficult to improve the results (since, in a sense, we are also competing with ourselves). Our 2006 implementation is also slightly more incremental than the 2003 one, but this is not seen as a major factor in these results in contrast to the new moves proposed in Section 3.4.

n	Old Best	min(D)	max(D)	mean(D)	std(D)
8	41928	41928	43025	42037.65	291.98
10	63832	63832	64409	63860.85	125.75
12	120665	**119608**	120774	**120121.55**	417.07
14	208086	**199363**	210599	**202400.50**	2883.39
16	279618	**279077**	297173	284036.95	4770.61

n	T for min	mean(T)	std(T)
8	0.1	1555.55	1880.94
10	477.2	8511.29	17132.49
12	15428.1	49373.31	32834.88
14	34152.3	70898.90	48551.27
16	55640.8	47922.16	36948.40

Table 3.11: Solution Quality and Solution Times for NLB Distances with Mirroring

3.5.1 Mirrored Instances

Tables 3.11, 3.12, and 3.13 report the results for mirrored instances, which are particularly impressive. The algorithm matches or improves all best-known solutions (but one). It produces 8 new best solutions and the improvements essentially occur for larger instances. This was a surprising result for us, since we thought that mirroring instances would be significantly more challenging for the algorithm. Some of the improvements may also be quite large and reach more than 4%.

3.5.2 Non-Mirrored Instances

Tables 3.14, 3.15, and 3.16 report the results for the non-mirrored instances. On the NLBn and constant distance metrics, the algorithm is once again impressive, it matches or improves all the best-known solutions, and improves 6 instances. It is interesting to note that, even though we did not extensively experiment with the non-mirrored NLBn instances, we were able to further improve upon the results of the standard TTSA algorithm in two cases (for $n = 10$ and $n = 14$). Once again, the higher gains are obtained on the larger instances.

The results on the circular instances are somewhat disappointing. The algorithm cannot match the best-known results on the larger instances, although it is often very close to the best-known solutions. This could be due to the fact that the algorithm only uses mirrored starting schedules, which may bias the search. In fact, the best solutions found by our algorithm for 16 and 20 teams are mirrored schedules. These instances may need to be investigated more carefully, to determine whether this failure is related to the instances' structure or simply to bad parameter tuning.

However, as we will show in Chapter 4, incorporating our algorithm into a more advanced framework for Simulated Annealing can lead to big improvements in solution quality, improving the best-known results for all $n \geq 12$.

n	Old Best	min(D)	max(D)	mean(D)	std(D)
8	80	80	80	80	0
10	130	130	130	130	0
12	192	192	192	192	0
14	253	253	253	253	0
16	342	342	342	342	0
18	432	432	432	432	0
20	524	**522**	**522**	**522**	0
22	650	650	650	650	0
24	768	768	768	768	0

n	T for min	mean(T)	std(T)
8	0.1	0.06	0
10	0.1	0.10	0
12	0.3	0.56	0.38
14	6.0	154.26	147.95
16	2.7	3.29	1.53
18	8.1	24.60	19.20
20	1106.3	12556.20	10347.58
22	24.3	45.42	22.90
24	813.3	1791.77	983.47

Table 3.12: Solution Quality and Solution Times for Constant Distances with Mirroring

n	Old Best	min(D)	max(D)	mean(D)	std(D)
8	140	140	140	140	0
10	272	272	276	273.60	1.01
12	456	**432**	**444**	**434.90**	3.12
14	714	**696**	726	**708.90**	7.05
16	978	**968**	1072	1001.60	28.55
18	1306	1352	1364	1357.80	3.40
20	1882	**1852**	2198	2017.60	60.64

n	T for min	mean(T)	std(T)
8	0.2	74.18	55.13
10	28160.0	12527.18	12208.25
12	93.1	4658.58	3560.27
14	53053.5	23549.14	16311.15
16	38982.7	23360.81	14451.53
18	178997.5	106139.77	57175.01
20	59097.9	43137.13	22515.46

Table 3.13: Solution Quality and Solution Times for Circular Distances with Mirroring

n	Old Best	min(D)	max(D)	mean(D)	std(D)
8	39721	39721	39721	39721	0
10	59436	59436	59583	59561.63	48.33
12	111483	**111248**	116018	112663.32	738.55
14	189759	**189156**	195742	193187.85	1432.99
16	270063	**267194**	282005	273552.64	3461.49

n	T for min	mean(T)	std(T)
8	1169.0	1639.33	332.38
10	2079.6	27818.24	64873.91
12	202756.2	150328.30	92385.48
14	90861.4	77587.86	40346.49
16	344633.4	476191.65	389371.71

Table 3.14: Solution Quality and Solution Times for NLB Distances without Mirroring

n	Old Best	min(D)	max(D)	mean(D)	std(D)
8	80	80	80	80	0
10	124	124	124	124	0
12	181	181	181	181	0
14	252	252	252	252	0
16	327	327	329	328	0.31
18	418	**417**	418	**417.65**	0.47
20	521	**520**	522	**520.90**	0.53
22	626	626	629	628.80	0.77
24	757	**749**	**753**	**750.60**	0.96

n	T for min	mean(T)	std(T)
8	0.2	0.14	0.14
10	4.6	3.96	2.43
12	128.7	1126.85	1480.45
14	26.1	95.32	59.42
16	82884.1	16042.20	22332.36
18	10362.8	7091.27	6614.78
20	7781.7	22850.72	25094.76
22	57487.2	25082.01	21426.11
24	95516.2	29727.86	24904.04

Table 3.15: Solution Quality and Solution Times for Constant Distances without Mirroring

n	Old Best	min(D)	max(D)	mean(D)	std(D)
8	132	132	132	132.00	0
10	242	242	256	252.70	3.24
12	408	420	432	427.13	3.43
14	654	662	690	679.70	5.14
16	928	968	1072	1001.60	28.55
18	1306	1352	1364	1357.80	3.40
20	1842	1852	2198	2017.60	60.64

n	T for min	mean(T)	std(T)
8	3.2	589.23	590.74
10	19261.6	14491.27	7937.21
12	151459.1	96717.13	52788.38
14	127942.1	88381.18	68978.05
16	38982.7	23360.81	14451.53
18	178997.5	106139.77	57175.01
20	59097.9	43137.13	22515.46

Table 3.16: Solution Quality and Solution Times for Circular Distances without Mirroring

3.6 Understanding the Neighborhood Structure

In order to get a better understanding of the neighborhood and, in particular, how different subsets of the neighborhood affect the efficiency of the search algorithm, we performed a series of experiments with varying neighborhood subsets.

We ran two kinds of experiments: We first compared the whole neighborhood (including mirrored moves) with the subset of the neighborhood that includes no mirrored moves. We then compared the whole neighborhood with its subset consisting only of mirrored moves (in which case we had to start from a mirrored initial schedule).

The test bed for our experiments were the mirrored NLBn and CIRCn instances, for $n = 14$ and $n = 16$. On all the experiments, the maximum running time was set to 40,000 seconds for $n = 14$ and 60,000 for $n = 16$. The main reason for this choice was that these instance size are big enough to be representative of the whole set of instances, and small enough to allow for relatively extensive experimentation. One caveat, before trying to draw any definite conclusions, is the inherent dependence of Simulated Annealing on good tuning. Although we did our best in determining a good tuning, these results can only serve as good indications and not as conclusive proofs. With this in mind, we proceed with a detailed account of our findings in the following subsections.

3.6.1 Using No Mirrored Moves

The goal in these experiments was to explore whether it is beneficial to include mirrored moves, when dealing with mirrored instances. Our methodology was as follows. For every instance set, in the first phase, we executed 20 runs using the whole neighborhood starting from different random schedules. In the second phase, for every instance set we executed three sets of 20 runs using no mirror moves, each time starting from the same 20 starting points we used in the first phase. In every set of runs we used exactly the same parameters as in the first phase, only varying the Markov chain length of the Simulated Annealing (given the smaller neighborhood size). For each instance set, our conclusions are based on the best of the three second-phase experiments.

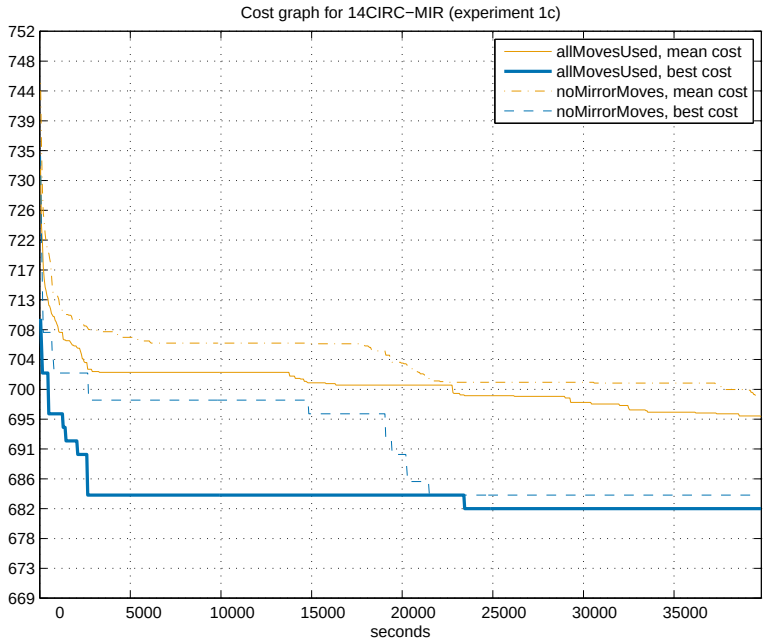
Figures 3.7 and 3.8, compare the cost evolution over time for each instance set, when using each neighborhood subset. As one can deduce from the graphs, including mirrored moves in the neighborhood leads to better efficiency in terms of mean cost over time. Moreover, in two cases (CIRC14 and NLB16), using no mirrored moves does not allow for matching the best solution found using the whole set of moves (at least in the given time limitations). This is a clear indication that using mirrored moves really enhances the algorithm's search capability.

3.6.2 Using Only Mirrored Moves

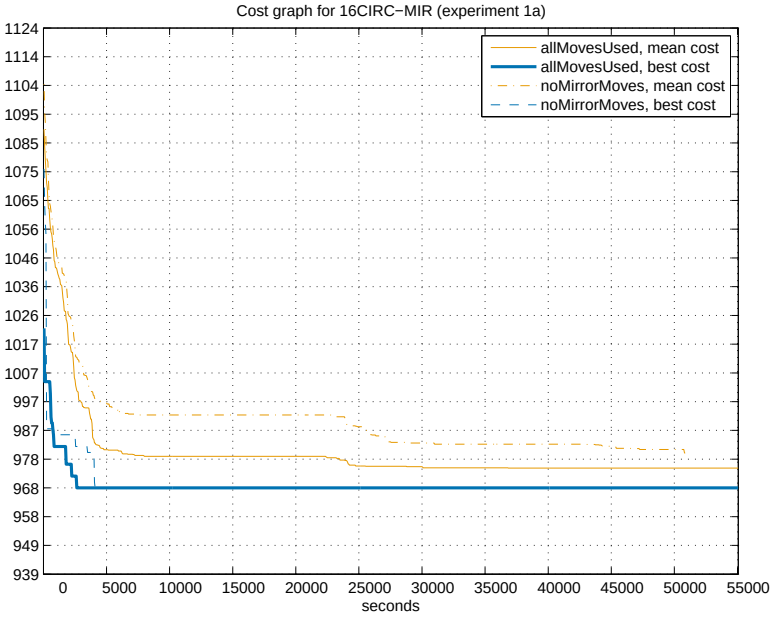
We use the same methodology as in Section 3.6.1, while only changing the subset of the neighborhood used: instead of the subset consisting only of non-mirrored moves, we use the subset consisting *only* of mirrored moves. The corresponding graphs are depicted in Figures 3.9 and 3.10. As seen in the figures, the picture is not as clear, in this case: In particular, although, in terms of mean cost, using only mirrored moves seems to be more efficient, in terms of best cost, there are cases in which using only mirrored moves leads to worse solutions within the set time limits.

The general pattern seems to be the following: using only mirrored moves leads to increased efficiency in the beginning of the search; however, using the whole set of moves seems to have better performance, in the long run. This indicates a direction for future work: for instance, one could design more elaborate search schemes, that start with the restricted neighborhood of only mirrored moves, and gradually enhance it by including increasing subsets of the full neighborhood (possibly in a framework of Variable Neighborhood Search (VNS) [23]

Also, we note that, as a positive side-effect of the above sets of experiments, we were able to further improve on two of the mirrored instances: for the mirrored NLB16 we found a solution of value 278,305, and for the mirrored CIRC14 a solution of value 672.

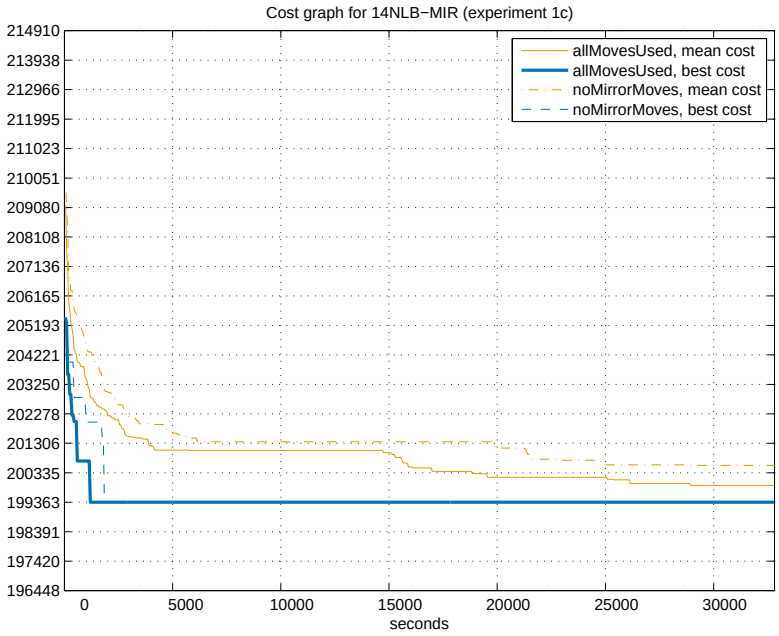


(a) CIRC14-MIR

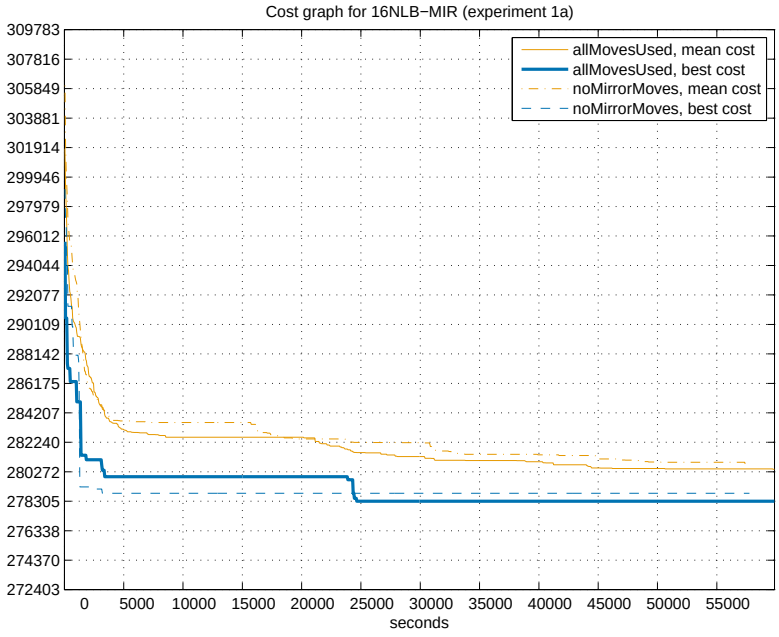


(b) CIRC16-MIR

Figure 3.7: Comparison of Using No Mirrored Moves for Circular Instances

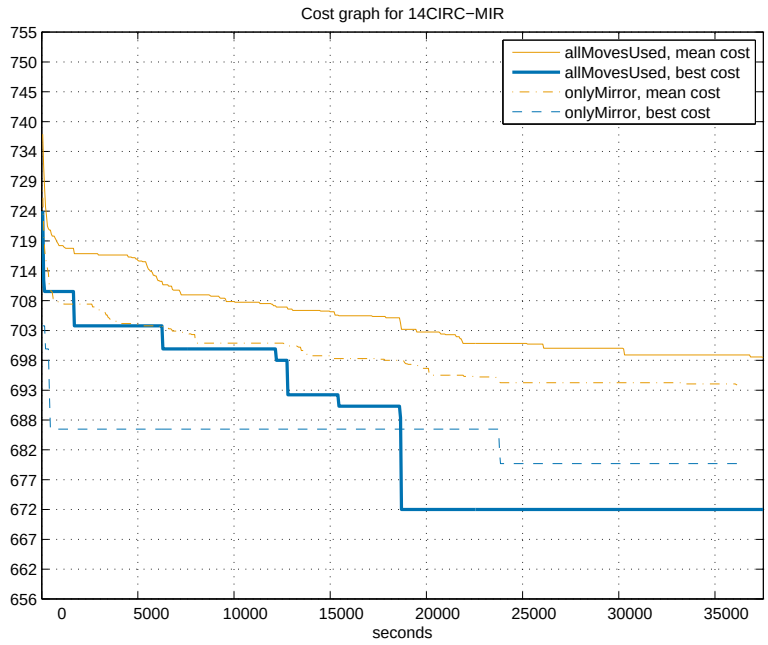


(a) NLB14-MIR

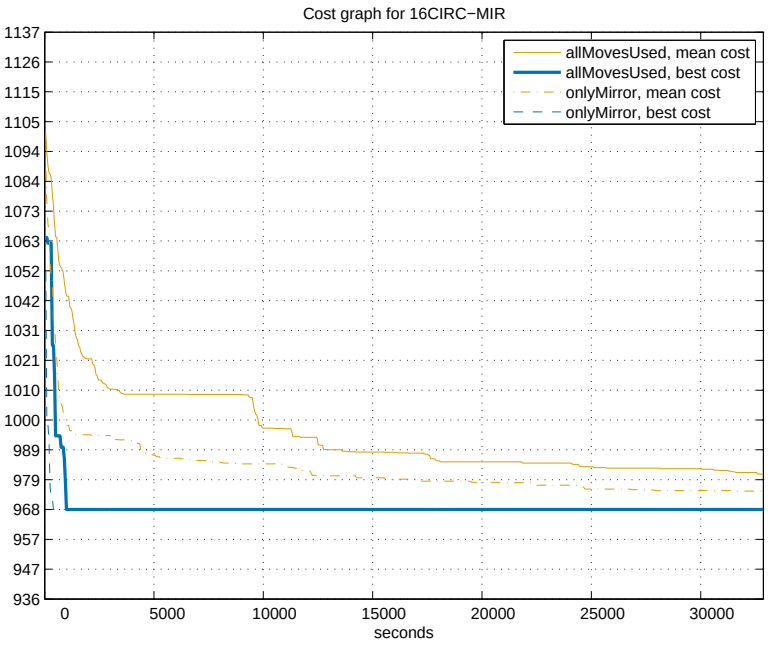


(b) NLB16-MIR

Figure 3.8: Comparison of Using No Mirrored Moves for NLB Instances

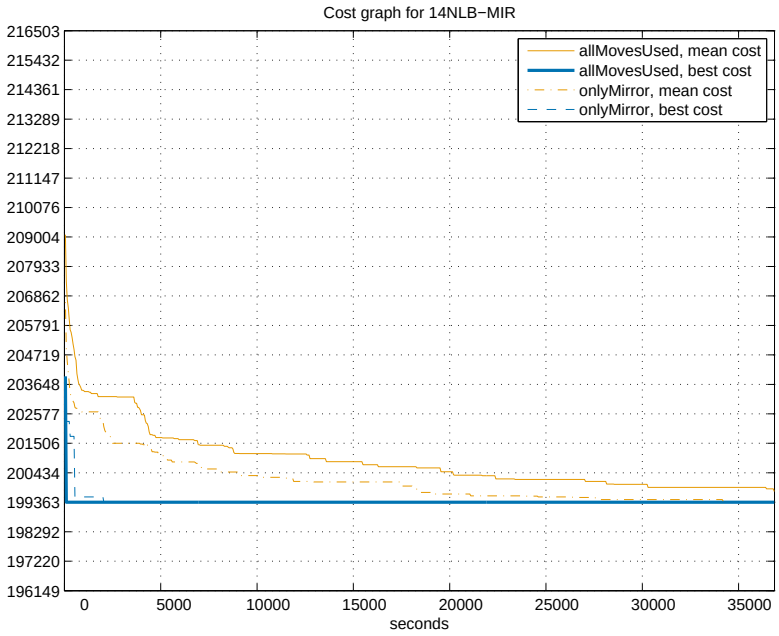


(a) CIRC14-MIR

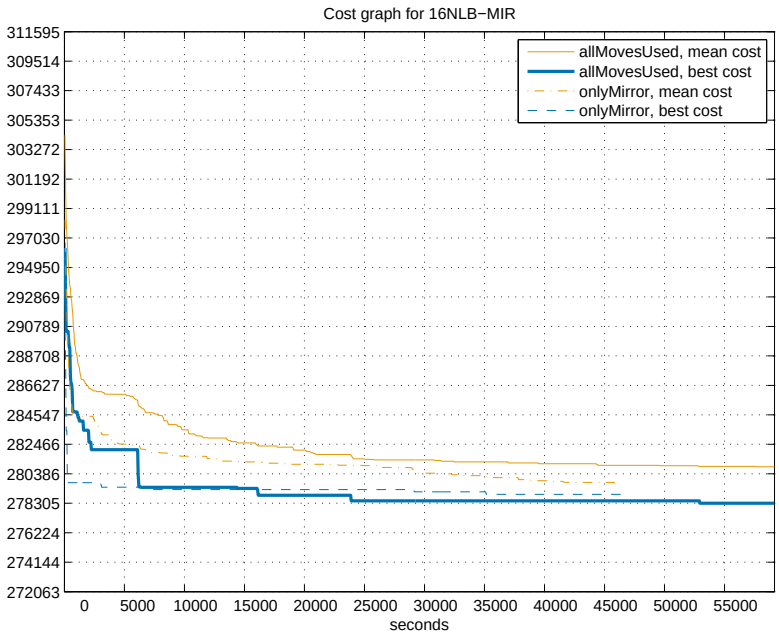


(b) CIRC16-MIR

Figure 3.9: Comparison of Using Only Mirrored Moves for Circular Instances



(a) NLB14-MIR



(b) NLB16-MIR

Figure 3.10: Comparison of Using Only Mirrored Moves for NLB Instances

3.7 Time Considerations

In the course of our work on the TTP (including variants), our main focus was on acquiring solutions of as high quality as possible, and not so much on efficiency. For this reason, the question remained as to whether we could get the same solution quality faster. That has been pointed out in some related work. For example, Di Gaspero and Schaerf [11, 12] embedded a subset of our neighborhood in a tabu-search algorithm and obtained high-quality solutions much faster (although their best solutions still do not match our best found solutions).

However, we have recently been able to resolve this question to a great degree. Looking back into our TTSA code more carefully, we realized that there was substantial room for improvement in its incremental structure. After spending some time doing a new, more incremental, implementation of the moves, we were, in fact, able to obtain the same results shown in Section 3.5 in times faster by a factor of 5 to 6. This indicates that the aforementioned question may not pertain as much to our search scheme as to implementation issues.

3.8 Lower Bounds

Although the focus of this work is on obtaining high-quality upper bounds for the different instances, it is of high interest to give a short summary of the results appearing in the references regarding lower bounds.

Unlike research on improving the best-known upper bounds, research on improving the best-known lower bounds for TTP instances has shown much slower progress, since the problem was introduced. Two main reasons for this are: first, the high degree of dependencies between individual team schedules and, second, the increased complication arising from the feasibility patterns involved.

Some first lower bounds on TTP instances were proven by Easton et.al. [15], through the computation of what they define as the *Independent Bound*. These lower bounds were further improved by the same authors in [16].

The most success in obtaining lower bounds has been seen for the simplest class of TTP instances, namely the CONST instances described earlier in Section 3.4. Rasmussen and Trick [41] gave lower bounds for most of the CONST instances mentioned on the website [49]. Fujiwara et.al. [19], were able to prove improved lower bounds for some of the larger CONST instances. For mirrored instances, many of the best-known lower bounds have been proven by Urrutia and Ribeiro in [50]. An interesting aspect of CONST instances is the connection with the problem of minimizing or maximizing breaks under given constraints, which further illustrates the significance of the problem we described in Chapter 2.

One can point out that, in most CONST instances, the above mentioned lower bounds match the best-known upper bounds, indicating that these instances are relatively easier. However, this does not undermine the significance of the lower bounds: for instance, in a recent paper, Urrutia et.al. [51], utilizing the CONST bounds found above, further improved the best-known lower bound on several of the harder TTP instances.

Instance	Best other	LB	min	%G
nlb8	39721	39479	39721	0
nlb10	59436	57500	59436	0
nlb12	111483	107494	**111248**	5.9
nlb14	189759	182797	**189156**	8.7
nlb16	270063	249477	**267194**	13.9
circ8	132	128	132	0
circ10	242	228	242	0
circ12	408	384	420	-50.0
circ14	654	590	666	-18.8
circ16	928	846	968	-48.8
circ18	1306	1188	1352	-39.0
circ20	1842	1600	1852	-4.1
const8	80	80	80	0
const10	124	124	124	0
const12	181	181	181	0
const14	252	252	252	0
const16	327	327	327	0
const18	418	414	**417**	25.0
const20	521	520	**520**	100
const22	626	626	626	0
const24	757	744	**749**	61.5

Table 3.17: Summary of TTSA Results for Non-Mirrored Instances Relative to LB

Tables 3.17 and 3.18 summarize the results found in this chapter, setting them in the context of the corresponding best-known upper bounds. The tables describe the previous best solution (Best other) (found by other researchers), the best-known lower bound (LB), the best solution found using TTSA (min) and the reduction of the optimality gap (best - LB) in percentage (%G). Many of the conclusions described earlier, regarding TTSA performance, are more vividly illustrated by looking at the optimality gap reduction (%G).

For example, for non-mirrored instances, we have seen that TTSA performs well only on NLB and CONST instances, while having poor performance for CIRC instances. This is clearly illustrated in the last column of Table 3.17. As we can see, TTSA solutions usually have more than 40% worse optimality gap, compared to solutions computed using other methods.

On the other hand, as mentioned in earlier sections of this chapter, TTSA is highly successful on mirrored instances, and this is also reconfirmed by Table 3.18, where it is shown that TTSA often improves the optimality gap by more than 30%.

Instance	Best other	LB	min	%G
nlb8-mir	41928	41928	41928	0
nlb10-mir	63832	58277	63832	0
nlb12-mir	120655	110519	**119608**	10.3
nlb14-mir	208086	182996	**199363**	34.8
nlb16-mir	279618	253957	**278305**	5.1
circ8-mir	140	140	140	0
circ10-mir	272	240	272	0
circ12-mir	456	384	**432**	33.3
circ14-mir	714	590	**672**	33.9
circ16-mir	978	876	**968**	9.8
circ18-mir	1306	1188	1352	-39.0
circ20-mir	1882	1600	**1852**	10.6
const8-mir	80	80	80	0
const10-mir	130	130	130	0
const12-mir	192	192	192	0
const14-mir	253	253	253	0
const16-mir	342	342	342	0
const18-mir	432	432	432	0
const20-mir	524	520	**522**	50.0
const22-mir	650	650	650	0
const24-mir	768	768	768	0

Table 3.18: Summary of TTSA Results for Mirrored Instances Relative to LB

Chapter 4

Population-Based Simulated Annealing

Looking back at Table 3.9, shown in Chapter 3, or spending some time on the result sections of the web page found in [49], an interesting observation is that progress in improving the beat-known upper bounds on TTP instances seems to have slowed down in recent years. In fact, some of the early instances have not been improved for several years.

This chapter reconsiders the Traveling Tournament Problem by proposing a population-based simulated annealing algorithm, PBSA, with both intensification and diversification components, in an a successful attempt to reverse the situation. The chapter leverages the simulated annealing algorithm TTSA presented in Chapter 3, which is used as a black-box.

The core of the algorithm is organized as a series of waves, each wave consisting of a collection of simulated annealing runs. At the end of each wave, a (macro-)intensification takes place: a majority of the simulated annealing runs are restarted from the best found solution. Diversification is achieved through the concept of *elite runs*, a generalization of the concept of elite solutions. More precisely, at the end of each wave, the simulated annealing runs that produced the k-best solutions so far continue their execution opportunistically from their current states. This core procedure is terminated when a number of successive waves fail to produce an improvement and is then restarted at a lower temperature.

A parallel implementation of PBSA on a cluster of workstations exhibits remarkable results. It produces new best solutions on all TTP instances considered, sometimes reducing the optimality gap by about 60%, including the larger NLB instances which had not been improved for several years, the circular instances, and the NFL instances for up to 26 teams. Although TTSA is not the most appropriate algorithm for circular instances, the population-based algorithm has improved all best solutions for 12 teams or more on these instances. The improvements are often significant, reducing the optimality gap by almost 40% on some circular instances. In addition, the parallel implementation also obtained these results in relatively short times, compared to the version of

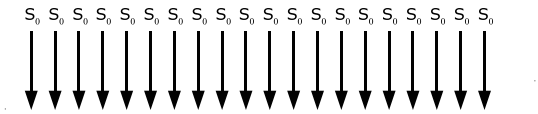
TTSA used in [54].

The broader contributions of the current chapter are twofold. First, it demonstrates the potential complementarity between the macro-intensification and macro-diversification typically found in tabu-search and population-based algorithms and the micro-intensification (temperature decrease) and micro-diversification (threshold acceptance of degrading move) featured in simulated annealing. Second, it indicates the potential benefit of population-based approaches for simulated annealing, in contrast to recent negative results in [37].

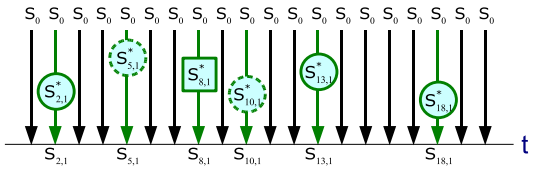
4.1 The PBSA Algorithm

The core of the population-based simulated annealing receives a configuration S (e.g., a schedule) and a temperature T. It executes a series of waves, each of which consists of n executions of the underlying simulated annealing algorithm (in this case, TTSA). The first wave simply executes `SA(S,T)` N times (where N is the size of the population). Subsequent waves consider both opportunistic and intensified executions. The simulated annealing runs that produced the k-best solutions so far continue their executions: hopefully, they will produce new improvements and they provide the macro-diversification of the algorithm. The $N - k$ remaining runs are restarted from the best solution found so far and the temperature T.

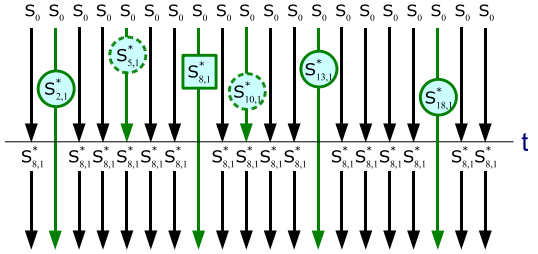
Figure 4.1 illustrates the core of the algorithm for a population of size $N = 20$ and $k = 4$. Figure 4.1(a) shows that all executions start from the same configuration and Figure 4.1(b) depicts the behavior during wave 1. The best solution obtained is $S^*_{8,1}$ (solid square in figure). Several other executions also produce solutions ($S^*_{2,1}$, $S^*_{5,1}$, $S^*_{10,1}$, $S^*_{13,1}$, $S^*_{18,1}$) that improve upon their starting points (circles in figure). The best 3 of these (solid circle), together with the best solution found so far, define the elite runs used for diversification (i.e., $S^*_{2,1}$, $S^*_{8,1}$, $S^*_{13,1}$, $S^*_{18,1}$). Figure 4.1(c) depicts the start of the second wave. It highlights that the elite runs continue their execution from their current state, while the remaining 16 executions restart from the best solution $S^*_{8,1}$ and the initial temperature. Figure 4.1(d) shows the executions of the first two waves. The second wave found a new best solution $S^*_{13,2}$ (produced by one of the elite runs), while several executions improve upon their starting points. Execution 18, which had produced elite solution $S^*_{18,1}$ after the first wave, further improves upon its best solution, producing solution $S^*_{18,2}$. However, it is not among the best 4 solutions. These are now $S^*_{7,2}$, $S^*_{8,1}$, $S^*_{13,2}$, and $S^*_{15,2}$, and the simulated annealing executions that produced them are now the set of elite runs. Figure 4.1(e) depicts the launch of the third wave. Observe that the two elite runs (those that produced $S^*_{8,1}$ and $S^*_{13,2}$) will now execute for the third successive wave, while two new ones have emerged. This core procedure terminates after a number of stable waves, i.e., successive waves that have not improved the best solution. It is embedded in an outermost loop that progressively decreases the temperature T.



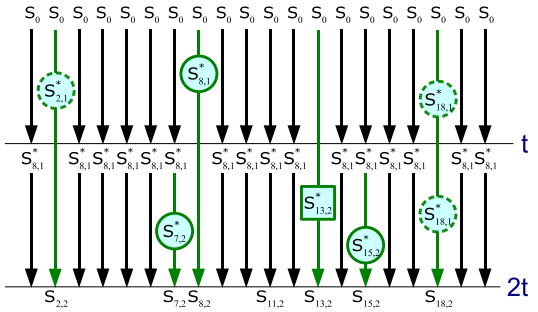
(a) Launch of first wave



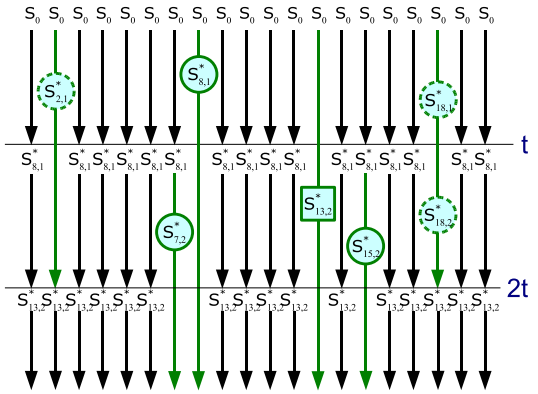
(b) After first wave



(c) Launch of second wave



(d) End of second wave



(e) Launch of third wave

Figure 4.1: Illustrating PBSA with $k = 4$

The overall algorithm is depicted in Figures 4.2 and 4.3. Figure 4.2 describes the core procedure PBSA-P for a population $\mathcal{P}$ of size $N = |\mathcal{P}|$. For each member p of the population, the algorithm maintains its current starting configuration S_p and temperature T_p, as well as the value f_p of the best solution p has generated. These variables are initialized in lines 2–6. The algorithm also maintains the best overall solution S^* and the number, *stable*, of successive waves without improvement to S^*. Lines 8–23 are concerned with the execution of a wave. For each $p \in \mathcal{P}$, PBSA-P applies the simulated annealing algorithm for t units of time on configuration S_p with starting temperature S_p. The simulated annealing execution returns the best solution S_p^* of this run and the final configuration S_p^+ and temperature T_p^+ (line 8). If the run improves its starting solution, i.e., $f(S_p^*) < f(S_p)$, PBSA-P updates variable f_p (line 11). If these runs have not improved the best solutions, the next wave continues each of the runs from their current state (see the instructions in lines 12–13 that implement this behavior). Otherwise, the runs that produced the k-best solutions (the elite runs) continue their executions, while the remaining $N - k$ runs (the set $\mathcal{R}$ in line 19) are restarted from their current best solution S^* and the initial temperature T (line 20–22). Figure 4.3 shows that PBSA-P is embedded in a loop which progressively decreases the temperature (lines 3–6). The overall algorithm PBSA also starts from a solution produced by simulated annealing or, possibly, by any other algorithm.

4.2 Experimental Results with PBSA

The algorithm was implemented in parallel, to execute each run in a wave concurrently. The experiments were carried out on a cluster of 60 Intel-based, dual-core, dual-processor Dell Poweredge 1855 blade servers. Each server has 8GB of memory and a 300G local disk. Scheduling on the cluster is performed via the Sun Grid Engine, version 6. The tested instances are the non-mirrored NLBn, CIRCn and NFLn instances described in [49].

The experiments use a population of size $N = 80$ and the number k of elite runs is in the range [10,30]. The time duration t of each wave is in the range of [60,150] seconds depending on the size of the instances. PBSA-P terminates after a maximum number of successive non-improving waves chosen in the range of [5,10]. PBSA is run for 10 phases with $\beta = 0.96$. We report two types of results starting from either low or high-quality TTSA solutions. All results reported are averaged over 10 runs.

4.2.1 PBSA from High-Quality Solutions

The experimental results are summarized in Tables 4.1, 4.2, and 4.3, which report both on solution quality and execution times. With respect to solution quality, the tables describe the previous best solution (best) (not found by PBSA), the best lower bound (LB), the minimum (min) and average (mean) travel distances found by PBSA, and the improvement in the optimality gap (best - LB) in percentage (%G). The results on execution times report the times (in seconds) taken for the best run (Time(Best)), the average times (mean(T)), and the standard deviation (std(T)).

```
1.  function PBSA-P(S, T) {
2.      forall p ∈ P do
3.          S_p = S;
4.          f_p = f(S);
5.          T_p = T;
6.      S* = S;
7.      stable = 0;
8.      while stable < maxStableWaves do
9.          forall p ∈ P do
10.             ⟨S*_p, S+_p, T+_p⟩ = SA_t(S_p, T_p);
11.             if f(S*_p) < f(S_p) then f_p = f(S*_p);
12.             S_p = S+_p;
13.             T_p = T+_p;
14.         b = arg min_{p∈P} f_p;
15.         if f(S*) > f(S*_b) then
16.             S* = S*_b;
17.             stable = 0;
18.             f^k = k- min_{p∈P} f_p;
19.             R = {p ∈ P : f_p > f^k};
20.             forall p ∈ R do
21.                 S_p = S*;
22.                 T_p = T;
23.         else stable = stable + 1;
24.     return S*;
25. }
```

Figure 4.2: PBSA-P: A Phase of PBSA

```
1.  function PBSA(S) {
2.      T ← T_0;
3.      S ← SA_t(S,T);
4.      for phase = 1 to maxPhases do
5.          S ←PBSA-P(S, T);
6.          T ← T · β;
7.      end for
8.      return S;
9.  }
```

Figure 4.3: The Population-Based Simulated Annealing Algorithm PBSA

n	Best	LB	min	mean	%G
14	189156	182797	**188728**	**188728.0**	6.7
16	267194	249477	**262343**	**264516.4**	27.3

n	Time(Best)	mean(T)	std(T)
14	360	264.0	139.94
16	600	468.0	220.94

Table 4.1: Quality and Times in Seconds of PBSA for NLB Distances

n	Best	LB	min	mean	%G
12	408	384	**406**	414.5	8.3
14	654	590	**632**	**645.2**	34.3
16	928	846	**916**	**917.8**	14.6
18	1306	1188	**1294**	1307.0	10.1
20	1842	1600	**1732**	**1754.4**	45.4

n	Time(Best)	mean(T)	std(T)
12	1440	787.5	706.39
14	1080	402.0	287.81
16	180	342.0	193.58
18	4680	3380.0	1950.86
20	10270	8437.0	1917.18

Table 4.2: Quality and Times in Seconds of PBSA for Circular Distances

n	Best	LB	min	mean	%G
16	235930	223800	**231483**	**232998.4**	36.7
18	296638	272834	**285089**	**286302.9**	48.5
20	346324	316721	**332041**	**332894.5**	48.2
22	412812	378813	**402534**	**404379.7**	30.2
24	467135	-	**463657**	**465568.7**	-
26	551033	-	**536792**	**538528.0**	-

n	Time(Best)	mean(T)	std(T)
16	2220	1356.0	998.31
18	3120	2412.0	1811.52
20	6750	4419.0	1349.06
22	8100	4365.0	2484.79
24	5490	4113.0	2074.70
26	6480	3024.0	1927.42

Table 4.3: Quality and Times in Seconds of PBSA for NFL Distances

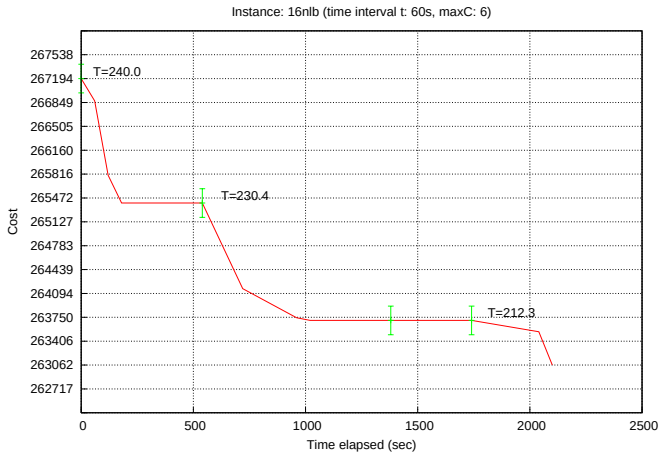


Figure 4.4: Evolution of the Objective on NLB-16

As far as solution quality is concerned, PBSA improves on all best-known solutions for the circular instances with 12 teams or more. It also improves on the NLB instances with 14 and 16 teams. These NLB instances had not been improved for several years despite new algorithmic developments and approaches. It also improves the NFL instances for 16 to 26 teams (larger instances were not considered for lack of time). The improvement in the optimality gap is often substantial. For NLB-16, CIRC-20, and NFL-20, the improvements are respectively about 27%, 45%, and 48%.

As far as solution times are concerned, PBSA typically finds its best solutions in times significantly shorter than TTSA. On the NLB instances, PBSA found its new best solutions within 10 minutes, although these instances had not been improved for a long time. Typically, the new best solutions are found within an hour for problems with less than 20 teams and in less than two hours otherwise. These results are quite interesting as they exploit modern architectures to find the best solutions in competitive times, the elapsed times being significantly shorter than TTSA.

It is also interesting to look at PBSA's running behavior. Figure 4.4 depicts the evolution of its objective function over time for NLB-16 and is representative of the typical behavior of the algorithm. The execution exhibits alternating sequences of fast-improving and stagnant periods. Intuitively, repeated intensifications and diversifications allow the search to escape from very "difficult" local minima and to start improving again rapidly.

4.2.2 PBSA from Scratch

Tables 4.4, 4.5, and 4.6 describe the performance of PBSA when the TTSA is only run for a short amount of time to produce a starting point. These results are particularly interesting. PBSA improves the best-known solutions for the NLB instances for 12, 14, and 16 teams, for the circular instances 12 and 14, and for NFL instances 16, 18, 20, 22, and 26.

For the NLB, the improvement for 14 teams is the same as when PBSA starts from a high-quality solution, while the improvement for 12 and 16 teams is even better, producing new best solutions.

n	Best	LB	min	mean	%G
12	111248	107494	**110729**	112064.0	13.8
14	189156	182797	**188728**	190704.6	6.7
16	267194	249477	**261687**	**265482.1**	31.0

n	Time(Best)	mean(T)	std(T)
12	2370	1501.5	816.73
14	3045	2491.5	1067.94
16	18150	12858.0	3190.31

Table 4.4: Quality and Times in Seconds of PBSA from Scratch for NLB Distances

n	Best	LB	min	mean	%G
12	408	384	**404**	418.2	16.6
14	654	590	**640**	654.8	21.8
16	928	846	958	971.8	-36.5
18	1306	1188	1350	1371.6	-37.2
20	1842	1600	1856	1874.0	-5.7

n	Time(Best)	mean(T)	std(T)
12	2200	1102.0	560.31
14	1720	1396.0	457.10
16	7260	4962.0	1743.11
18	6660	5994.0	4070.67
20	12930	9587.1	2364.97

Table 4.5: Quality and Times in Seconds of PBSA from Scratch for Circular Distances

n	Best	LB	min	mean	%G
16	235930	223800	**233419**	**234847.9**	20.7
18	296638	272834	**282258**	**285947.6**	60.4
20	346324	316721	**333429**	**337280.3**	43.5
22	412812	378813	**406201**	**411967.5**	19.4
24	467135	-	471536	476446.6	-
26	551033	-	**545170**	553175.5	-

n	Time(Best)	mean(T)	std(T)
16	14010	14325.0	1626.11
18	19320	17097.0	2164.13
20	19680	18771.0	2053.67
22	23730	17691.0	4199.72
24	22110	18645.0	2150.32
26	18600	24621.0	3448.87

Table 4.6: Quality and Times in Seconds of PBSA from Scratch for NFL Distances

The optimality gap is reduced by about 14%, 7%, and 31% respectively. For the circular instances, the improvement is better than when PBSA starts from a high-quality solution, for 12 teams, but not as good for more teams. In fact, starting from scratch improves upon the best-known solution only for $n = 14$, and is not very competitive for a larger number of teams. Finally, for the NFL instances, it is interesting to note that, in the case of $n = 18$, PBSA produces a better solution starting from scratch rather than from a high-quality solution.

In addition, it is important to note that, in all cases, the elapsed times are more significant compared to starting from a given high-quality solution, but are still lower than the times for TTSA.

4.2.3 TTSA versus PBSA

Figure 4.5 depicts the behavior of TTSA over a long time period (three days) and compares it with PBSA. In this experiment, 80 independent TTSA processes run concurrently with no information exchange and the figure shows the evolution of the best found solution over time. The results show that TTSA achieves only marginal improvement after the first few hours. However, in about five hours, PBSA achieves a substantial improvement over the best solution found by TTSA in the three days.

4.2.4 The Effect of Macro-Diversification

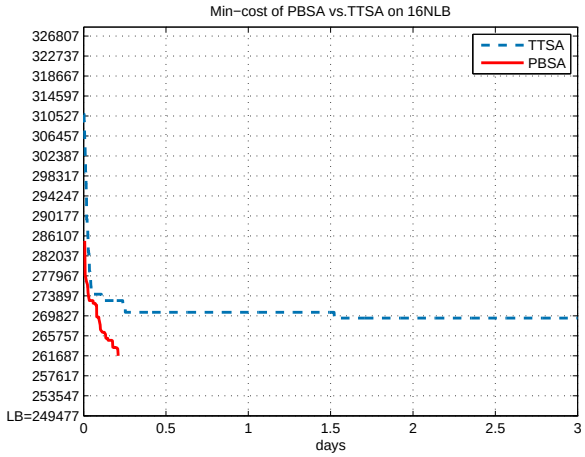
In order to assess the extent to which macro-diversification affects solution quality, we ran PBSA (for 10 iterations) on NLB-16 starting from scratch, varying the number of elite runs k. Note that $k = 0$ corresponds to no macro-diversification. The table in Figure 4.6(a) depicts the minimum and the mean cost, and the gap reduction. The results seem to indicate a nice complementarity between macro-intensification and macro-diversification. The same data is illustrated in Figure 4.6(b), in which the trend is even more obvious.

4.3 Connections with Related Work

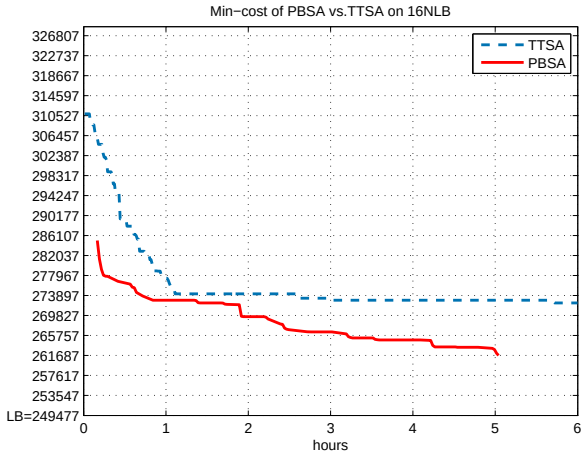
At this point, it would be of interest to explore in a bit more detail the relationships between PBSA and the cooperative parallel search, scatter search, and memetic algorithms mentioned in Chapter 1

4.3.1 Cooperative Parallel Search

Population-based simulated annealing can be viewed as a cooperative parallel search. There is a great variety of proposed schemes of this kind, a large number of which is based on simulated annealing. The most straight-forward schemes are attempts to parallelize sequential versions of simulated annealing (e.g., [47]). However, the schemes appearing in the literature range over a much broader spectrum.



(a) Min-Cost Graph for TTSA and PBSA on NLB-16

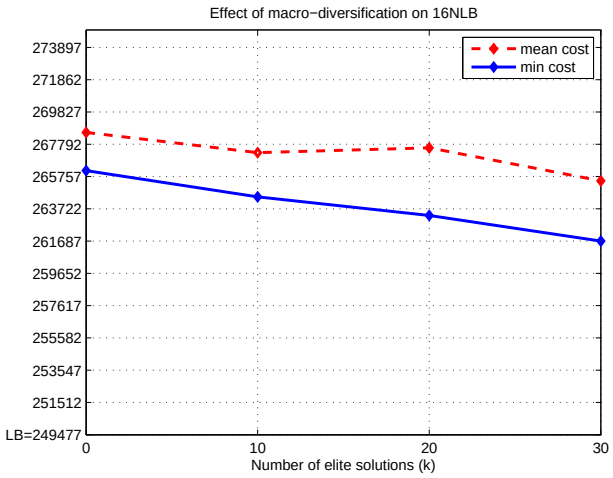


(b) Min-Cost Graph for TTSA and PBSA on NLB-16 (Zoomed)

Figure 4.5: Comparison of Min-Cost Evolution for TTSA and PBSA on NLB-16

k	Best	LB	min	mean	%G
0	267194	249477	**266130**	268538.6	6.0
10	267194	249477	**264472**	267261.0	15.3
20	267194	249477	**263304**	267563.1	21.9
30	267194	249477	**261687**	**265482.1**	31.0

(a) Tabular Form



(b) Graph Representation

Figure 4.6: The Effect of Macro-Diversification (NLB-16)

Onbaşoğlu et. al. 2001 [37] provide an extensive survey of parallel simulated annealing algorithms and compare them experimentally on global optimization problems. They also classify those schemes into application-dependent and application-independent parallelization.

In the first category, the problem instance is divided among several processors, which communicate only to deal with dependencies. For instance, in VLSI design, the processors specialize on different areas of the circuit. See [22] for a detailed account of parallel simulated annealing techniques for VLSI design. In the second category, Onbaşoğlu et. al. further distinguish between asynchronous parallelization with no processor communication, synchronous parallelization with different levels of communication, and highly-coupled synchronization in which neighborhood solutions are generated and evaluated in parallel. In the first two cases, processors work on separate Markov chains while, in the third case, they cooperate on a single Markov chain.

Communication patterns between processors can take the form of simple transmission of cost values, occasional exchange of (possibly partial) solutions, or even intensive exchanges of solutions. Hybrid schemes combining different forms of communication have also been developed (e.g., [27]). There are schemes that cannot be easily classified, such as the parallel simulated annealing in [8], in which the processors work on highly inter-dependent Markov chains by mixing states.

PBSA can thus be viewed as an application-independent algorithm with synchronous parallelization and periodic exchange of solutions. The scheme proposed in [25] (which only exchanges partial solutions) and the SOEB-F algorithm [37] are probably the closest to PBSA; however, they do not use diversification and elite runs. An interesting observation is also that SOEB-F typically fails to produce sufficiently good solutions [37].

It is also useful to point out that the above classification is not limited to simulated annealing. A cooperative parallel scheme based on tabu search is presented in [5] and is applied to the generalized assignment problem. In this context, we can point out that PBSA could be lifted into a generic algorithm providing macro-intensification and macro-diversification for any meta-heuristic. Whether such a generic algorithm would be useful in other contexts remains to be seen, however.

4.3.2 Memetic Algorithms and Scatter Search

PBSA can be seen as a degenerated form of scatter search [29] where solutions are not combined but only intensified. Moreover, the concept of elite solutions is replaced by the concept of elite runs which maintains the state of the local search procedures. PBSA can also be viewed as a degenerated form of memetic algorithms [36], where there is no mutation of solutions: existing solutions are either replaced by the best solution found so far or are "preserved". Once again, PBSA does more than preserving the solution: it also maintains the state of the underlying local search through elite runs. Obviously, an interesting research direction would be to study how to enhance PBSA into an authentic scatter search and memetic algorithm. The diversification so-obtained may further improve the results.

4.4 Summary of All Results

We conclude this chapter with an updated version of the tables summarizing the best upper bounds for mirrored and non-mirrored instances, taking into consideration both TTSA and PBSA results. Note that, unlike similar tables presented earlier in Chapter 4, gap reduction percentages are now computed by comparing the best upper bound produced by any of our approaches (TTSA or PBSA) to the best-known upper bound published by other researchers.

Table 4.7 shows results for non-mirrored instances, while Table 4.8 shows the corresponding results for mirrored instances. For the non-mirrored case, we are able to produce matching or improving upper bounds on all cases considered, with a significant gap reduction, which often reaches above 60%.

Because of time constraints, we have very few results using PBSA on mirrored instances: we only have results for the NFL16-MIR and NFL24-MIR instances, and also for the BRAZ24-MIR instance (posted on [49]), which uses cities in the 2003 Brazilian soccer championship, with 24 teams. The most recent improvement to this last instance was posted two years ago. On all three instances, PBSA was able to improve upon the best-known upper bound, which indicates that it is worth devoting some future work into fully assessing the effectiveness of PBSA on mirrored instances. Since we did not apply PBSA on other mirrored instances, Table 4.8 is almost identical to the corresponding Table 3.18 presented in Chapter 3; we include it here for the sake of completeness. Once again, the high effectiveness of simulated annealing based techniques is evident, since there is only a single case (CIRC18-MIR), in which we do not match the best-known upper bound. In all other cases, we find matching or improving solutions, often reducing the gap from the lower bound by more than 30%.

Instance	Best other	LB	min	%G
nlb8	39721	39479	39721	0
nlb10	59436	57500	59436	0
nlb12	111483	107494	**110729**	18.9
nlb14	189759	182797	**188728**	14.8
nlb16	270063	249477	**261687**	40.7
circ8	132	128	132	0
circ10	242	228	242	0
circ12	408	384	**404**	16.7
circ14	654	590	**632**	34.4
circ16	928	846	**916**	14.6
circ18	1306	1188	**1294**	10.2
circ20	1842	1600	**1732**	45.5
const8	80	80	80	0
const10	124	124	124	0
const12	181	181	181	0
const14	252	252	252	0
const16	327	327	327	0
const18	418	414	**417**	25.0
const20	521	520	**520**	100
const22	626	626	626	0
const24	757	744	**749**	61.5
nfl16	235930	223800	**231483**	36.7
nfl18	296638	272834	**282258**	60.4
nfl20	346324	316721	**332041**	48.2
nfl22	412812	378813	**402534**	30.2
nfl24	467135	-	**463657**	-
nfl26	551033	-	**536792**	-

Table 4.7: Summary of All Results for Non-Mirrored Instances Relative to LB

Instance	Best other	LB	min	%G
nlb8-mir	41928	41928	41928	0
nlb10-mir	63832	58277	63832	0
nlb12-mir	120655	110519	**119608**	10.3
nlb14-mir	208086	182996	**199363**	34.8
nlb16-mir	279618	253957	**278305**	5.1
circ8-mir	140	140	140	0
circ10-mir	272	240	272	0
circ12-mir	456	384	**432**	33.3
circ14-mir	714	590	**672**	33.9
circ16-mir	978	876	**968**	9.8
circ18-mir	1306	1188	1352	-39.0
circ20-mir	1882	1600	**1852**	10.6
const8-mir	80	80	80	0
const10-mir	130	130	130	0
const12-mir	192	192	192	0
const14-mir	253	253	253	0
const16-mir	342	342	342	0
const18-mir	432	432	432	0
const20-mir	524	520	**522**	50.0
const22-mir	650	650	650	0
const24-mir	768	768	768	0
nfl16-mir	251289	228251	**248818**	10.7
nfl24-mir	467135	-	**465863**	-
braz24-mir	503158	-	**500756**	-

Table 4.8: Summary of All Results for Mirrored Instances Relative to LB

Conclusion

Sport league scheduling has received considerable attention in recent years, since these applications involve significant revenues for television networks and generate challenging combinatorial optimization problems. In the following paragraphs, we first summarize the most important conclusions drawn from previous chapters of the dissertation; we then indicate possible extensions and future research directions related to this thesis; we conclude with some general remarks.

As a conceptual contribution, this thesis shows that, contrary to common belief, Local Search is, in fact, an effective method for tackling sport scheduling problems. This is demonstrated by our approach to two very important problems in Sport Scheduling: the Break Minimization problem and the TTP.

For the Break Minimization Problem, we propose a simulated annealing scheme, BMSA, based on a simple connected neighborhood and a systematic scheme for cooling the temperature and deciding termination. The resulting algorithm is conceptually simple and easy to implement; yet, it always finds optimal solutions on the instances used in evaluating the state-of-the-art algorithm of [17], regardless of its starting points. More importantly, BMSA exhibits excellent performance and significantly outperforms earlier approaches on instances with more than 20 teams.

In the case of the Traveling Tournament Problem proposed in [49, 15], our simulated annealing algorithm, TTSA, is able (with suitable enhancements) to match or significantly improve the best-known solutions for most instances of the TTP, both in its original form, and in a number of variants, including different distance metrics and mirroring constraints. The gains are higher for larger instances. The key to these results is the design of a sophisticated neighborhood that is very well adapted to the problem's particular structure. Moreover, further enhancing TTSA by embedding it within the PBSA population-based scheme leads to even more impressive results.

Our PBSA results shed some light on the complementarity between the micro-intensification and micro-diversification inherent to simulated annealing and the macro-intensification and macro-diversification, typically found in other meta-heuristics or frameworks. They also illustrate another technical contribution of this work: the successful adaptation of ideas of tabu search into simulated annealing. In addition, an important feature of the PBSA scheme is that it is general enough to be applied to other problems or instances, or even use search methods other than TTSA.

A technical novelty that came out of studying variants of the TTP problem was the introduction of novel neighborhood moves that capture sequences of earlier moves. As such, these novel moves do not improve the connectivity of the neighborhood for the TTP. Their significance comes from the fact that, in the original algorithm, these sequences have a low probability, although they capture fundamental aspects of the mirroring or distance structure. An interesting direction of future research would be to explore to which extent this "aggregation" technique for introducing new moves capturing the special structure of a problem can lead to improved solutions. From a theoretical aspect, it would also be interesting to determine if the original TTSA neighborhood is actually connected.

An obvious extension would be to assess the general applicability of the PBSA scheme, through its application to different problems. A potentially more exciting branch of future research would be to expand our understanding of how to combine TTP solutions to produce scatter search and memetic algorithms for the TTP. Alternatively, one could study how to enhance PBSA into an authentic scatter search and memetic algorithm. These techniques will have the benefit of producing structural diversification, which may be fundamental in improving the TTP results even further.

From the overall discussion in this work, it follows that simulated annealing appears to be the method of choice for finding high-quality solutions to sport scheduling problems, especially for large-scale problems. Its efficiency can be significantly boosted through its incorporation into broader population-based schemes.

From a practical standpoint, this is extremely significant, especially given the fast-increasing commercial interest in sport scheduling applications and the corresponding increase in complexity and problem sizes that this entails.

This aside, it is very crucial, from a research-oriented perspective, to develop highly efficient local search schemes that, combined with other established advanced techniques, will broaden our understanding in the general field of combinatorial optimization.

Bibliography

[1] E. Aarts and P. van Laarhoven. Statistical Cooling: A General Approach to Combinatorial Optimization Problems. *Philips Journal of Research*, **40**(4):193–226, 1985.

[2] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Ltd, England, 1997.

[3] A. Anagnostopoulos, L. Michel, P. Van Hentenryck and Y. Vergados. A Simulated Annealing Approach to the Traveling Tournament Problem. In the *Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'03)*, Montréal, Canada, May 2003.

[4] A. Anagnostopoulos, L. Michel, P. Van Hentenryck and Y. Vergados. A Simulated Annealing Approach to the Traveling Tournament Problem. *Journal of Scheduling*, **9**(2):177–193, April 2006.

[5] Y. Asahiro, M. Ishibashi and M. Yamashita. Independent and Cooperative Parallel Search Methods for the Generalized Assignment Problem. *Optimization Methods and Software*, **18**(2):129–141, April 2003.

[6] T. Benoist, F. Laburthe and B. Rottembourg. Lagrange Relaxation and Constraint Programming Collaborative Schemes for Travelling Tournament Problems. In the *Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'01)*, Wye College (Imperial College), Ashford, Kent, UK, April 2001.

[7] A. Cardemil and G. Durán. Un algoritmo tabú sarch para el traveling tournament problem. (In Spanish). In *Revista Ingeniería de Sistemas*, **18**(1):95–115, June 2004.

[8] K. W. Chu, Y. Deng and J. Reinitz. Parallel simulated annealing by mixing of states. *Journal of Computational Physics*, **148**(2):646–662, January 1999.

[9] D. T. Connolly. General Purpose Simulated Annealing. *Journal of the Operational Research Society*, **43**(5):495–505, May 1992.

[10] A. Davenport and E. Tsang. Solving Constraint Satisfaction Sequencing Problems by Iterative Repair. In *Proceedings of the First International Conference on the Practical Applications of Constraint Technologies and Logic Programming (PACLP'99)*, pp. 345–357, London, UK, April 1999.

[11] L. Di Gaspero and A. Schaerf. A Tabu Search Approach to the Traveling Tournament Problem. In *Proceedings of RCRA 2005, Associazione Italiana per l'Intelligenza Artificiale (AI*IA)*, pp. 23–27, Ferrara, Italy, June 2005.

[12] L. Di Gaspero and A. Schaerf. A Composite-Neighborhood Tabu Search Approach to the Traveling Tournament Problem. *Journal of Heuristics*, **13**(2):189–207, April 2007.

[13] J. A. Díaz and E. Fernández. A Tabu Search Heuristic for the Generalized Assignment Problem. *European Journal of Operational Research*, **132**(1):22–38, July 2001.

[14] J. H. Dinitz and D. R. Stinson. A Hill-Climbing Algorithm for the Construction of One-Factorizations and Room Squares. *SIAM Journal on Algebraic and Discrete Methods*, **8**(3):430–438, July 1987.

[15] K. Easton, G. Nemhauser and M. Trick. The Traveling Tournament Problem Description and Benchmarks. In T. Walsh, editor, *Proceedings of the 7th International Conference on the Principles and Practice of Constraint Programming (CP'01)*, pp. 580–584, Paphos, Cyprus, 2001. LNCS **2239**, Springer-Verlag, 2001.

[16] K. Easton, G. Nemhauser and M. Trick. Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV, 4th International Conference (PATAT'02), Selected Revised Papers*, pp. 100–112, Gent, Belgium, 2002. LNCS **2740**, Springer-Verlag, 2003.

[17] M. Elf, M. Jünger and G. Rinaldi. Minimizing Breaks by Maximizing Cuts. *Operations Research Letters*, **31**(3):343–349, May 2003.

[18] M. Elf, M. Jünger and G. Rinaldi. Personal communication, 2004.

[19] N. Fujiwara, S. Imahori, T. Matsui and R. Miyashiro. Constructive Algorithms for the Constant Distance Traveling Tournament Problem. In E. Burke and H. Rudova, editors, *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)*, pp. 402–405, Masaryk University, Brno, Czech Republic, August 2006.

[20] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[21] M. Goemans and D. Williamson Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems using Semidefinite Programming. *Journal of the ACM*, **42**(6):1115–1145, November 1995.

[22] D. R. Greening. Parallel Simulated Annealing Techniques. *Physica D: Non-linear Phenomena*, **42**(1-3):293–306, June 1990.

[23] P. Hansen and N. Mladenovic. An Introduction to Variable Neighborhood Search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pp. 433–458. Kluwer Academic Publishers, 1998.

[24] M. Huang, F. Romeo and A. Sangiovanni-Vincentelli. An Efficient General Cooling Schedule for Simulated Annealing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 381–384, Santa Clara, CA, USA, November 1986.

[25] D. Janaki Ram, T. H. Sreenivas and K. Ganapathy Subramaniam. Parallel Simulated Annealing Algorithms. *Journal of Parallel and Distributed Computing*, **37**(2):207–212, September 1996.

[26] S. Kirkpatrick, C. D. Gelatt, Jr. and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, **220**(4598):671–680, May 1983.

[27] G. Kliewer and S. Tschöke. A General Parallel Simulated Annealing Library and its Application in Airline Industry. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 55–62, Cancun, Mexico, May 2000.

[28] M. Laguna, J. P. Kelly, J. L. Gonzalez-Velarde and F. Glover. Tabu Search for the Multilevel Generalized Assignment Problems. *European Journal of Operational Research*, **82**(1):176–189, April 1995.

[29] M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, Boston, USA, 2003.

[30] A. Lim, B. Rodrigues and X. Zhang. A Simulated Annealing and Hill-Climbing Algorithm for the Traveling Tournament Problem. *European Journal of Operational Research*, **174**(3):1459–1478, November 2006.

[31] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller. Equation of State Calculation by Fast Computer Machines. *Journal of Chemical Physics*, **21**(6):1087–1092, June 1953.

[32] R. Miyashiro and T. Matsui. Round-Robin Tournaments with a Small Number of Breaks. *Technical Report, Mathematical Engineering Technical Reports, METR 2003-29*, Department of Mathematical Informatics, Graduate School of Information Science and Technology, the University of Tokyo, 2003.

[33] R. Miyashiro and T. Matsui. A Polynomial-Time Algorithm to Find an Equitable Home-Away Assignment. *Operations Research Letters*, **33**(3):235–241, May 2005.

[34] R. Miyashiro and T. Matsui. Semidefinite Programming Based Approaches to the Break Minimization Problem. *Computers and Operations Research*, **33**(7):1975–1982. Elsevier Science Ltd., Oxford, UK, July 2006.

[35] G. Nemhauser and M. Trick. Scheduling a Major College Basketball Conference. *Operations Research* **46**(1):1–8, January 1998.

[36] M. G. Norman and P. Moscato. A Competitive and Cooperative Approach to Complex Combinatorial Search. In *Proceedings of the 20th Informatics and Operations Research Meeting*, pp. 3.15–3.29, Buenos Aires, Argentina, August 1991.

[37] E. Onbaşoğlu and L. Özdamar. Parallel Simulated Annealing Algorithms in Global Optimization. *Journal of Global Optimization*, **19**(1):27–50, January 2001.

[38] I. H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. *Annals of Operations Research (Special issue on Tabu search)*, **41**(1–4):421–451, 1993.

[39] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.

[40] E. Pesch and F. Glover. TSP Ejection Chains. *Discrete Applied Mathematics*, **76**(1–3):165–181, June 1997.

[41] R. Rasmussen and M. Trick. A Benders Approach to the Constrained Minimum Break Problem. *European Journal of Operational Research*, **177**(1):198–213, February 2007.

[42] R. Rasmussen and M. Trick. Round Robin Scheduling - A Survey. *European Journal of Operational Research* (Accepted for publication), 2007.

[43] J. C Régin. Minimization of the Number of Breaks in Sports Scheduling Problems Using Constraint Programming. In E. C. Freuder and R. J. Wallace, editors, *Constraint Programming and Large Scale Discrete Optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **57**:115–130. American Mathematical Society Publications, 2001.

[44] C. C. Ribeiro and S. Urrutia. Heuristics for the Mirrored Traveling Tournament Problem. *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT'04)*, pp. 323–342, Pittsburgh, PA, USA, August 2004.

[45] J. A. M. Schreuder. Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues. *Discrete Applied Mathematics*, **35**(3):301–312, March 1992.

[46] H. Shen and H. Zhang. Greedy Big Steps as a Meta-Heuristic for Combinatorial Search. Available online at `http://goedel.cs.uiowa.edu/AR-group/readings/aaai_ttp.pdf`, 2004.

[47] A. Sohn. Parallel n-ary Speculative Computation of Simulated Annealing. *IEEE Transactions on Parallel and Distributed Systems*, **6**(10):997–1005, October 1995.

[48] M. A. Trick. A Schedule-Then-Break Approach to Sports Timetabling. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III, Third International Conference (PATAT'00), Selected Papers*, pp. 242–253, Konstanz, Germany, August 2000. LNCS **2079**, Springer-Verlag, 2001.

[49] M. Trick. `http://mat.gsia.cmu.edu/TOURN/` on-line reference. 2002-2007, last visited in May 2007.

[50] S. Urrutia and C. C. Ribeiro. Maximizing Breaks and Bounding Solutions to the Mirrored Traveling Tournament Problem. *Discrete Applied Mathematics*, **154**(13):1932–1938, August 2006.

[51] S. Urrutia, C. C. Ribeiro and R. Melo. A New Lower Bound to the Traveling Tournament Problem. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)*, pp. 15–18, Honolulu, HI, USA, April 2007.

[52] P. Van Hentenryck and L. Michel *Constraint-Based Local Search*. The MIT Press, 2005.

[53] P. Van Hentenryck and Y. Vergados. Minimizing Breaks in Sport Scheduling with Local Search. In S. Biundo, K. L. Myers and K. Rajan, editors, *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pp. 22–29, Monterey, CA, USA, June 2005.

[54] P. Van Hentenryck and Y. Vergados. Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealling. In J. C. Beck and B. M. Smith, editors, *Proceedings of the Third International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'06)*, pp. 228–243, Cork, Ireland, May 2006. LNCS **3990**, Springer-Verlag, 2006.

[55] P. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordrecht, The Netherlands, 1987.

[56] P. van Laarhoven. *Theoretical and Computational Aspects of Simulated Annealing*. Stichting Mathematisch Centrum, Amsterdam, The Netherlands, 1998.

[57] W. D. Wallis. *One-factorizations.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.