Jari Kyngäs

# Solving Challenging Real-World Scheduling Problems

Turku Centre *for* Computer Science

TUCS Dissertations
No 140, November 2011

# Solving Challenging Real-World Scheduling Problems

by

Jari Kyngäs

*To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Pub3 on 11th November 2011, at 12 noon.*

University of Turku

Department of Information Technology

Turku 2011

## Supervisors

Professor Olli Nevalainen
Department of Information Technology
University of Turku
Turku, Finland

Research Director Kimmo Nurmi
Satakunta University of Applied Sciences
Pori, Finland

## Reviewers

Professor Pasi Fränti
University of Eastern Finland
Joensuu, Finland

Professor Miklos Kresz
University of Szeged
Szeged, Hungary

## Opponent

Professor Jyrki Nummenmaa
School of Information Sciences
Tampere, Finland

ISBN 978-952-12-2633-5 (printed)
ISBN 978-952-12-2634-2 (electronic)
ISSN 1239-1883

Uniprint, Suomen Yliopistopaino Oy

## List of publications


[P1.] K. Nurmi and J. Kyngäs, ”A Framework for School Timetabling Problem” in Proc of the 3rd Multidisciplinary Int. Scheduling Conf.: Theory and Applications (MISTA), Paris, France, 2007, pp. 386-393.

[P2.] G. Post, S. Ahmadi, S. Daskalaki, J. H. Kingston, J. Kyngäs, K. Nurmi, D. Ranson and H. Ruizenaar, “An XML Format for Benchmarks in High School Timetabling”, Annals of Operations Research, Springer, USA, 2010.

[P3.] J. Kyngäs and K. Nurmi, “Scheduling the Finnish Major Ice Hockey League”, in Proc of the IEEE Symposium on Computational Intelligence in Scheduling, Nashville, USA, 2009, pp. 84-89.

[P4.] J. Kyngäs and K. Nurmi, “Scheduling the Finnish 1st Division Ice Hockey League”, in Proc of the 22nd Florida Artificial Intelligence Research Society Conference, Florida, USA, 2009, pp. 195-200.

[P5.] K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngäs, J. Marenco, C.C. Ribeiro, F. Spieksma, S. Urrutia and R. Wolf-Yadlin, “A Framework for Scheduling Professional Sports Leagues”, in Ao, Sio-Iong (ed.): IAENG Transactions on Engineering Technologies Volume 5, Springer, USA, 2010, pp. 14-28.

[P6.] E.I. Ásgeirsson, J. Kyngäs, K. Nurmi and M. Stølevik, “A Framework for Implementation-Oriented Staff Scheduling”, in Proc of the 5th Multidisciplinary Int. Scheduling Conf.: Theory and Applications (MISTA), Phoenix, USA, 2011. (submitted for publication)

[P7.] K. Nurmi, J. Kyngäs and G.Post, “Staff Scheduling for Bus Transit Companies”, Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists, Hong Kong, 2011. (in press)

**Abstract:** This work contains a series of studies on the optimization of three real-world scheduling problems, school timetabling, sports scheduling and staff scheduling. These challenging problems are solved to customer satisfaction using the proposed PEAST algorithm. The customer satisfaction refers to the fact that implementations of the algorithm are in industry use.

The PEAST algorithm is a product of long-term research and development. The first version of it was introduced in 1998. This thesis is a result of a five-year development of the algorithm. One of the most valuable characteristics of the algorithm has proven to be the ability to solve a wide range of scheduling problems. It is likely that it can be tuned to tackle also a range of other combinatorial problems.

The algorithm uses features from numerous different metaheuristics which is the main reason for its success. In addition, the implementation of the algorithm is fast enough for real-world use.

**Keywords:** Efficient algorithms, Metaheuristics, Scheduling, School timetabling, Sports scheduling, Staff scheduling.

# Content

# 1. Introduction to Scheduling Problems

The focus of this thesis is to present general frameworks for *school timetabling*, *sports scheduling* and *staff scheduling* problems, and to show that these problems can be solved to customer satisfaction using intelligent heuristic algorithms. The thesis is a collection of seven publications. In the first two publications we analyze the algorithm presented in [Nurmi, 1998] and discuss our participation in making an XML format for benchmarks in high school timetabling. The next two publications deal with scheduling the Finnish major and 1st division ice hockey leagues. A framework for scheduling professional sports leagues is given in the third sports scheduling article. The last two publications propose an implementation-oriented staff scheduling framework and describe the solution of problem instance combinatorial optimization problem from a Finnish bus transportation company.

This section briefly discusses the difference of hard and easy problems. A literature summary of the problems of the present thesis is also given. In Section 2 an introduction to metaheuristics is given. Our solution method is discussed as an example of these metaheuristics. Sections 3, 4 and 5 describe the optimization problems of school timetabling, sports scheduling and staff scheduling respectively. Mathematical models, implementation-oriented models, practical results and cooperation partners are discussed in these sections. Finally, Section 6 summarizes the results of the research.

School timetabling, sports scheduling and staff scheduling problems are hard combinatorial problems and they belong to a class called *NP-hard (non-deterministic polynomial-time hard) problems* [Garey and Johnson, 1979]. Other examples of this problem class include traveling salesman problem, subset sum, set partitioning, multiprocessor scheduling, graph coloring, vertex cover and integer linear programming. Under the best knowledge of recent days, to solve a NP-hard problem optimally, some kind of enumerative method has to be implemented, that is, a list of nearly all possible solutions has to be listed and the one with the best value will finally be selected. The time required to solve an NP-hard problem depends exponentially on the size of the problem. However, no-one has been able to prove that no polynomial time algorithm for NP-hard problems exists.

Intelligent heuristic algorithms need to be designed and implemented to be able to solve real-world instances of NP-hard problems. These heuristics find solutions in a reasonable time, but it is unlikely that the solutions are optimal, though probably of good quality. When developing a heuristic method one still has to be concerned about the running time. Even though an algorithm works in polynomial time it could still prove to be too slow for real-life applications. However, it is perfectly reasonable to run an algorithm fortnight if the solution has to be generated only once a year and if the customer is satisfied with the result. The customer is interested in the best possible value of the objective function, not in how fast this is reached. Therefore, in real-world scheduling the “customer satisfaction” is the most important optimality criterion.

Scheduling problems have attracted researchers since the mid-1950s, when the first commercial computers where used to solve project scheduling problems. Since that time, the evolution of scheduling has closely tracked the development of computers. Usable practical applications began to show up about 15 years ago. Initially the solution methods were not able to solve any difficult problems − usually solutions were found for some toy-problems with little or no connection to real-world problems. This thesis is not about how to solve challenging problems in theory. On the contrary, it presents how to solve these problems in practice.

Real-world scheduling has gained much attention among the researchers in the last decade. One important reason for this is that computers have evolved to such a state that they are able to calculate complicated and time-consuming mathematics in a reasonable amount of time. Another reason is that the requirements and constraints of real-world problems have become more complicated which makes it impossible to produce solutions manually. Public institutions and private companies around the world have become more aware of the possibilities for decision support technologies, and they no longer want to create the schedules manually. One further significant benefit of automating scheduling processes is the considerable amount of time saved by the administrative staff involved.

Several solution approaches for many different scheduling problems have been introduced. A great deal of research has concentrated on examination timetabling, university timetabling and course timetabling [Schaerf, 1999; McCollum, 2006]. Timetabling research has achieved promising and practicable results. Sports scheduling and staff scheduling have also been extensively studied. Excellent overviews on sports scheduling can be found in [Easton et al., 2004] and [Rasmussen and Trick, 2008] and on staff scheduling in [Burke et al., 2004] and [Ernst et al., 2004]. Furthermore, vehicle scheduling [Toth and Vigo, 2001] and production line scheduling are state-of-the-art scheduling problems. The production line scheduling derives from job-shop scheduling [Blazewicz et al., 1996; Jain and Meera, 1999]. The job-shop problem itself is a generalization of the famous traveling salesman problem, see e.g. [Applegate et al., 2007].

PATAT (The International Conference for the Practice and Theory of Automated Timetabling) is perhaps the most important scheduling conference concentrating on university timetabling, sports scheduling and staff scheduling. The publications of the conference give a good idea how the academic interest towards these scheduling problems has evolved in recent years. Another important scheduling conference is MISTA (Multidisciplinary International Scheduling Conference). MISTA concentrates on a wide variety of scheduling problems, for example on production scheduling. These conferences are held both in Europe and in the USA. Table 1 shows a classification of the scheduling publications presented in the PATAT and MISTA conferences. The “Transit scheduling class” includes vehicle, bus, railway and airport scheduling. The Other scheduling class includes e.g. production line scheduling, processor and memory scheduling and project and resource scheduling.

While the main focus in the PATAT 06 conference was on university, school and exam timetabling, the interest in PATAT 10 has moved to sports and staff scheduling (see Table 1). Our research focused on school timetabling in 2006-2007, sports scheduling in 2007-2009 and staff scheduling in 2009-2010. In this respect the advancement of our research has followed the academic main stream. Our current research addresses to transit scheduling and production scheduling. At the time of writing, MISTA 2011 conference announced to arrange a special session on passenger transport and logistics systems.

Table 1. Classification of the scheduling publications presented in PATAT and MISTA conferences from years 2004 to 2009.

| | PATAT 04 | PATAT 06 | PATAT 08 | PATAT 10 | MISTA 05 | MISTA 09 |
|---|---|---|---|---|---|---|
| University timetabling | 26 | 29 | 29 | 22 | 4 | 10 |
| Sports scheduling | 5 | 8 | 8 | 11 | 0 | 3 |
| Staff scheduling | 12 | 8 | 12 | 14 | 5 | 7 |
| Transit scheduling | 2 | 3 | 1 | 3 | 7 | 5 |
| Other scheduling | 10 | 5 | 6 | 0 | 77 | 55 |
| General methodology | 8 | 4 | 3 | 6 | 8 | 9 |
| Total sum | 63 | 57 | 59 | 56 | 101 | 89 |

## 2. Metaheuristics

Intelligent heuristic algorithms need to be designed and implemented to be able to solve real-world instances of hard problems. Perhaps the most prominent heuristic approaches used to be *neural networks (NN)* [Kohonen, 1984; Hecht-Nielsen, 1990; Rumelhart and McClelland, 1986] and *genetic algorithms (GA)* [Goldberg 1989]. These methods use ideas from their biological counterparts and are related in particular to artificial learning. Neural networks are loosely modeled on the operation principle of the human brain and learn by themselves using a set of training patterns. This learning can then be applied to classification, prediction or control tasks. The application areas of NNs have been rapidly widening including such interesting applications as credit card fraud detection, handwriting recognition and financial forecasting. Neural networks have also been quite successful in solving different types of combinatorial problems.

The basic idea of genetic algorithms is to use reproduction, crossover and mutation operators to produce good solutions. Although the idea of evolutionary computing is fascinating, its practical use is for the most part yet to come. There are, however, many examples of practical applications and future potential: criminal face-printing, network hardware development and oil exploration. Genetic algorithms can also be used to solve combinatorial problems, and this area has been steadily increasing.

The most successful metaheuristics in solving challenging scheduling problems do not solely use neural networks or genetic algorithms. They most likely use mixed local search and population-based methods. Osman and Laporte [1996] defined *metaheuristic* as follows: “A metaheuristic is an iterative generation process which guides a subordinate heuristic by combining intelligent concepts for exploring and exploiting the search space. Furthermore, learning strategies are used to structure information in order to efficiently find near-optimal solutions”.

Blum and Roli [2003] classify metaheuristics based on five criteria:

1) *Nature-inspired algorithms* are those that are inspired by some phenomenon in the nature. Genetic algorithms simulate natural selection, particle swarm optimization mimics the behavior of swarms, and so on. Nowadays many algorithms are some kind of *hybrids* – the algorithm may be nature-inspired but it can include parts that are not nature-inspired (for example tabu search).
2) *Population based algorithms* work with many simultaneous solution candidates. This may lead to high-quality solutions but the computing time increases very quickly.
3) *Dynamic objective function* means that the algorithm modifies the weights of the multiobjective function during the search. This can be of great help when the importance and difficulty of the objectives are not known or cannot even be guessed.
4) *One vs. various neighborhood structures*. Using various neighborhood structures usually makes the search space larger. An algorithm can search one neighborhood for some time and swap into another if good solutions are not found.
5) *Memory usage* means that an algorithm saves the search history while running. This helps the evaluation of search locations – if the search location has been visited earlier the algorithm can use that information.

Metaheuristic methods guide the search process towards near-optimal solutions. The idea is to efficiently explore the search space with a combination of simple local search procedures and/or complex learning processes. The methods are usually non-deterministic and they may include

mechanisms to avoid getting trapped in confined areas of the search space (i.e. local minima). Popular metaheuristic methods include ant colony optimization [Colorni et al., 1992], cooperative local search [Preux and Talbi, 1999; Landa Silva, 2003], ejection chains [Glover, 1992], genetic algorithms, hill-climbing, hyper-heuristics [Burke et al., 2003], idwalk [Neveu et al., 2004], iterated local search, memetic algorithms [Moscato, 1989], particle swarm optimization [Eberhart and Kennedy, 1995], simulated annealing [Kirkpatrick et al., 1983], tabu search [Glover et al., 1985] and variable neighborhood search [Hansen et al., 2000]. A classification of these metaheuristics is shown in Figure 1.

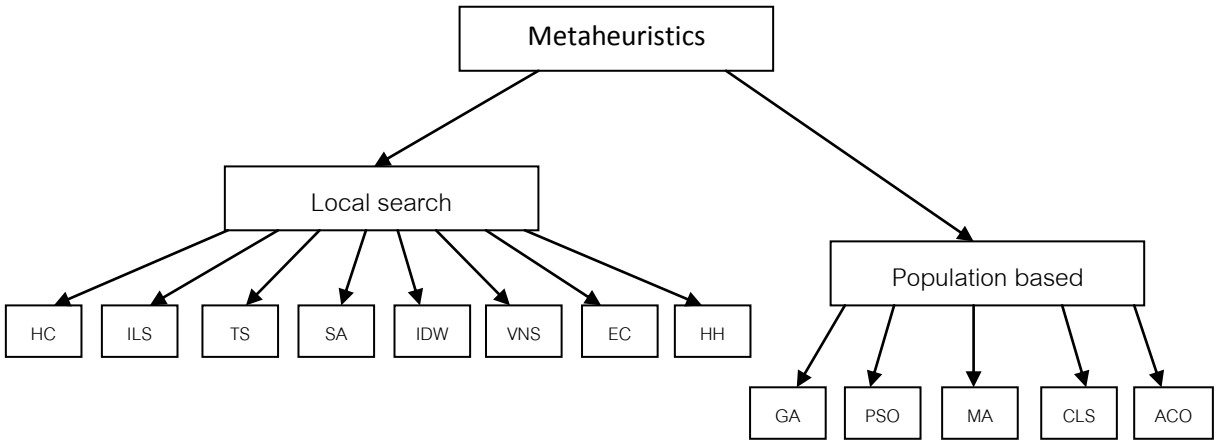


Figure 1. A classification of metaheuristic methods. For abbreviations, see text.

A local search method is defined by

1) A neighborhood structure, which is a mechanism to obtain a new set of neighbor solutions by applying a small perturbation to a given solution.
2) A method of moving from one solution to another.
3) Parameters of the method.

*Hill-Climbing (HC)* is a basic local search method. The idea of the method is to accept every better move. This enables the algorithm to “climb up the hill” until a local optimum is reached. HC starts with a random solution, and iteratively makes small changes to the solution to generate a neighborhood solution. If the neighborhood solution is better than or equal to the current solution, the neighborhood solution substitutes the current solution. When the current solution can no longer be improved, hill-climbing terminates. Almost all other local search methods are based on hill-climbing to some extent.

*Iterated Local Search (ILS)* is a sophisticated version of hill-climbing with random restarts. Hill-climbing with random restarts first finds a local optimum. Then it restarts the iteration from a random point and finds another local optimum, and so on. Finally, it returns the best optimum found. ILS is an extension to this procedure. It tries to stochastically hill-climb in the space of these local optimums. The idea is that it is more probable to find a global optimum near the local optimums than to restart from some random point.

*Tabu Search (TS)* is one of the most cited and used metaheuristics. The main idea is to prevent a new move from some location back to a location which has been visited before. TS is basically a

hill-climbing method, but it employs further strategies to avoid terminating at a local optimum, which often happens in hill-climbing. TS modify the neighborhood structure using a tabu list which is a short-term memory containing the solutions that have been visited in the recent past. TS exclude solutions in the tabu list from the new neighborhood.

Hill-Climbing methods accept only improving or equally good moves. Another approach is to accept also deteriorating moves. These moves help to escape from the local optimum.

In *Simulated Annealing (SA)* this acceptance is decreased over time so that it finally diminishes to zero. In a basic simulated annealing version a move is accepted with a probability $\exp(-\Delta f/T)$, where $\Delta f$ gives the increase in the cost function and $T$ is a control parameter which approaches zero.

*IDWalk (IDW)* performs *N* neighborhood moves and returns the best solution found. If this solution is better than or equal to the current solution, it is substituted for the current solution. If no such solution has been found, a previously rejected solution is selected to substitute the current solution. The rejected solution is selected from a list of rejected solutions according to some selection mechanism.

*Ejection Chains (EC)* are methods that generate a sequence of interrelated hill-climbing moves to create a more complex compound move. The compound moves allow a variable number of solution components to be modified within any single iteration of a local search. These modifications cause some solution components to be ejected from the solution space. Neighboring solutions obtained by an ejection chain process are created by a succession of embedded neighborhoods that lead to intermediate trial solutions at each level of the chain.

In *Variable Neighborhood Search (VNS)* more than one neighborhood structure is considered. After finishing the exploration with respect to one neighborhood, the search diversifies to another neighborhood. The neighborhood is changed during the search if no better solutions have been found in the current neighborhood. In this way one hopes to escape from local optima.

The efficiency of metaheuristics depends on the quality of the neighborhood structure. To overcome this problem, *Hyper Heuristics (HH)* use a set of simple and fast low-level heuristics. While Variable Neighborhood Search changes the neighborhood operator, HH changes the heuristic operator. The decision for selection of the heuristic during the search is based on problem independent measures, such as the change in the quality of solutions when the selected heuristic is used.

The main difficulty for a local search algorithm is

1) to explore promising areas in the search space to a sufficient extent, while at the same time
2) to avoid staying stuck in these areas too long and
3) to escape from these local optima in some systematic way.

Population-based methods use a population of solutions in each iteration. The outcome of an iteration is also a population of solutions. As already mentioned, the most commonly used population-based method is a genetic algorithm. Genetic algorithms are good at exploring the search space but find it difficult to zoom-in to find better-than-good solutions.

*Memetic Algorithm (MA)* is an extension of GA incorporating a local search for each solution in between generations. The main purpose is to introduce problem-dependent knowledge as a way of speeding-up the search process. However, the loss of diversity is specifically problematic in MA as the local search tends to focus in a few good solutions. MA is also known as hybridized or hybrid genetic algorithm.

*Particle Swarm Optimization (PSO)* is a population-based method which maintains a population of solutions called particles and moves these particles around in the search-space. Each particle changes its position and velocity based on the individual particle's best found solution and the global best found solution. That is, while flying in the search-space, every particle updates its velocity and position based on its own best experience and that of the entire population.

In a *Cooperative Local Search (CLS)* scheme, each individual carries out its own local search operator. When an individual gets stuck it asks for the cooperation of the population in order to find something to do to get unstuck and to continue the search from another position in the solution space. The results achieved by each individual may be different at different times and this encourages diversity within the population.

*Ant Colony Optimization (ACO)* is an approach which mimics ants searching for food. The ants are able to quickly find the shortest path from their nest to food sources. They accomplish this by leaving a substance called pheromone after them. When the followers choose directions to go to, they pick the paths with stronger pheromone concentration.

The described methods have been found successful in various scheduling and timetabling problems. Even though each of these methods works by itself, most of them best in cooperation with other methods.

## 3. The Scheduling Algorithm

This section describes the algorithm developed for tackling various scheduling problems. The design of the new algorithm was quite a long process starting from determining the power of the original algorithm and ending with the analysis of the various shuffling operators.

### The Original Algorithm, h-HCGA

The proposed algorithm, which will be used to solve school timetabling, sports scheduling and staff scheduling problems, has features from many metaheuristics. The foundation of the algorithm was established in [Nurmi, 1998]. Nurmi developed a new hybrid hill-climbing genetic algorithm (h-HCGA) that outperformed a standard GA and questioned some components of the GA that were normally considered necessary parts of genetic algorithms. The basic hill-climbing method was boosted by adding a tabu list and a simulated annealing refinement. A combination of HC and GA techniques was used to solve timetabling problems. Figure 2 shows a pseudo-code of the original h-HCGA algorithm. It should be noted that the new improved algorithm does not use the running time limit – the algorithm runs for a given number of generations. The improved algorithm does not use marriage selection [Muhlenbein, 1989] either, because it was found ineffective (see [P1]). Marriage selection is a process where we randomly pick two schedules $S_1$ and $S_2$ and choose the better one of them.

```
Set the running time limit t and the population size n
Generate initial random population of schedules
Set better_found = 0
WHILE elapsed-time < t
    REPEAT n times
        Select a schedule S by using marriage selection
        Apply GHCM to S to get a new schedule S'
        Calculate the change Δ in the objective function
        IF Δ ≤ 0 THEN
            Replace S with S'
            IF Δ < 0 THEN
                better_found = better_found + 1
            ENDIF
        ENDIF
    ENDREPEAT
    IF better_found > n THEN
        Replace the worst schedule with the best schedule
        Set better_found = 0
    ENDIF
    Update the dynamic weights of the hard constraints (ADAGEN)
    Calculate the new temperature for simulated annealing
ENDWHILE
Choose the best schedule from the population
```

Figure 2. A pseudo-code of the h-HCGA timetabling algorithm [Nurmi, 1998].

The algorithm uses a local search operator called GHCM (greedy hill-climbing mutation), see Figure 3. The operator moves an object, $o_1$, from its old position $p_1$ to a new position $p_2$ and then moves another object, $o_2$, from position $p_2$ to a new position $p_3$ and so on, ending up with a sequence of moves. The initial object selection is random. The new position $p_2$ is selected by considering all possible positions and selecting the one that minimizes the objective function. Then, the new object $o_2$ is again selected by considering all the objects in the position $p_2$ and picking the one that minimizes the object function. Next, a new position $p_3$ is selected, and so on. The sequence of moves stops if the last move causes an increase in the objective function value and if the value is larger than that of the previous non-improving move (see Figure 4). Then, a new sequence of moves is started. The initial solution is created randomly.

```
Randomly select one game g
Set best_cost = ∞, cumulative_cost = 0 and move_number = 1
WHILE true
    Select a new round r for the game among all the possible rounds so
        that the move g to r is not in the tabu list and the cost of moving g to r is minimized
    Update tabu list with the move g to q where q is g's old round
    cumulative_cost = cumulative_cost + cost (g to r)
    IF cumulative_cost ≤ best_cost THEN
        best_cost = cumulative_cost
        Save the sequence of moves
    ENDIF
    IF cost (g to r) > 0 AND the cost function value is above the cost function value in the
        previous non-improving move OR move_number = 10 THEN
        EXIT WHILE
    ENDIF
    Remove a game u from round r among all the possible games so that u and g
        have clashes and the cost function is minimized
    Set g = u
    Set move_number = move_number + 1
ENDWHILE
Save the sequence of moves
```

Figure 3. A pseudo-code of the GHCM operator [Nurmi, 1998].

An example of the utilization of the GHCM operator comes from scheduling the Finnish major ice hockey league. The objects, *o,* that the operator move, are the games to be scheduled. The positions, *p,* of the games are the rounds in which they are scheduled to be played. The operator starts with a random game in a random round. The game is moved to the round where it minimizes the object function. From that round another game is chosen to be moved. Again, the game that minimizes the object function is chosen. These moves are repeated until no further improvement in the object function is expected.

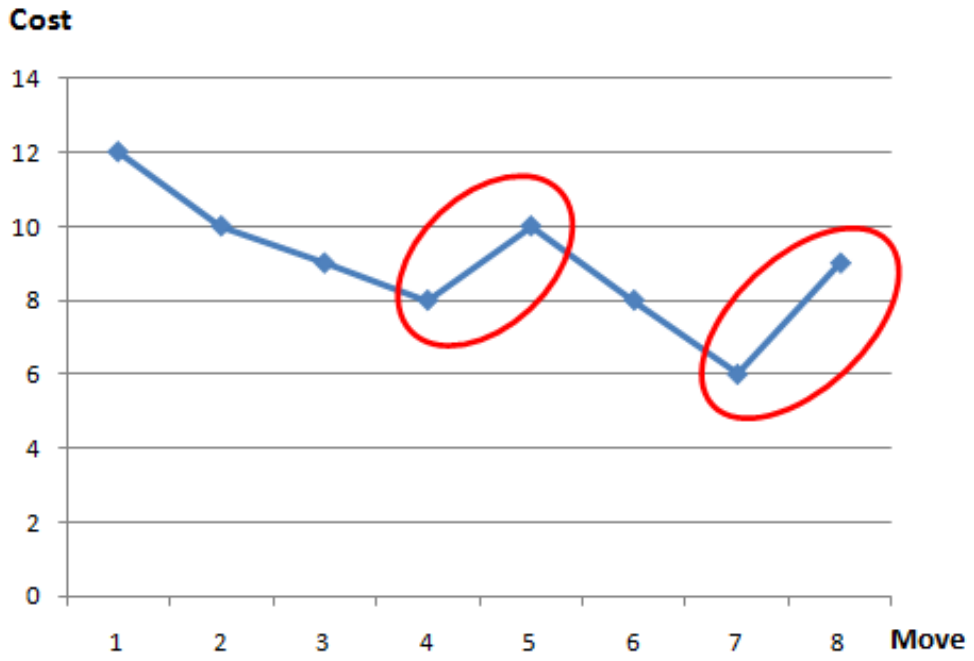


Figure 4: The latest move (step 7 to step 8) causes the GHCM to stop because the increase in the cost function value is bigger than in the previous non-improving move (step 4 to step 5).

The reproduction operation of the h-HCGA algorithm is, to a certain extent, based on the steady-state reproduction [Syswerda, 1989]: the new schedule replaces the old one if it has a better or equal objective function value. Furthermore, the worst schedule is replaced with the best one when *n* better schedules have been found, where *n* is the size of the population.

The algorithm uses an adaptive penalty method (ADAGEN) for multi-objective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints based on the weights assigned to the soft constraints [Nurmi, 1998]. The soft constraints are assigned constant weights according to their significance.

The hard constraint's lower bound is the same as the biggest soft constraint weight. The upper bound is calculated as (see [Nurmi, 1998]):

$$1 + (m + 1)\rho_{max} - \sum_{i=1}^{m} \rho_i \qquad (2.1)$$

where

$m$ is the number of soft constraints,
$\rho_{max}$ is the biggest weight assigned to the soft constraints,
$\rho_i$ are the weights assigned to the soft constraints.

All hard constraint weights are initially set to their lower bound. The weights are updated by -1, 0 or 1 in each generation according to their effect on the cost function between the generations (see [Nurmi, 1998] for a detailed description).

## The Improved Scheduling Algorithm, PEAST

An improved version of the h-HCGA algorithm is proposed in this thesis. As a first step we determined the best configuration of the h-HCGA algorithm, see [P1]. The best values of the nine control parameters were found by using both brute force and statistical analyses. These parameters included the reproduction/selection method (as in genetic algorithms), the maximum length of the move sequence (as in ejection chain method), the population size, the tabu list size and the update frequency of the hard constraint weights. The research showed that the chosen parameter values were very good for the introduced real-world and artificial school timetabling problems.

Next we made two changes to the h-HCGA algorithm, see [P4]. These changes aided the search procedure to escape from local optima as well as better explore the fitness landscape. As a result of these modifications the algorithm is able to solve a very challenging real-world sports scheduling problem instances, as discussed in [P3] and [P4].

The h-HCGA algorithm uses a simulated annealing refinement with a standard exponential cooling scheme. Test runs showed that a good strategy is to stop the cooling at some predefined temperature. Therefore, after a certain number of iterations $m$ we let the algorithm accept an increase in the cost function with some constant probability $p$. The results are surprisingly good by choosing $m$ equal to the maximum number of iterations with no improvement to the cost function and $p$ equal to 0.0015. The new annealing schedule produced superior solutions compared to the well-known annealing schedules.

The most successful modification to the h-HCGA algorithm concerned shuffling the current solution. As explained earlier, a hyper-heuristic is a mechanism that chooses a heuristic from a set of simple heuristics, applies it to the current solution, then chooses another heuristic and applies it, and continues this iterative cycle until the termination criterion is satisfied. The same idea is used in the modification algorithm, but the other way around. For this we introduced a number of simple heuristics that could normally be used to improve the current solution but, instead, we used them to shuffle the current solution - that is, worse solution candidates were allowed to replace better ones in the current population. Five shuffling operations were used and one random shuffling operation was selected in every $m$/20th iteration of the algorithm. The shuffling produced clearly better solutions than without shuffling, see [P4].

We tried to further improve the algorithm in [Nurmi and Kyngäs, 2009]. The GHCM operator normally selects the best move sequence. The operator was changed to accept the first better or equal move sequence and then not to continue further. In this way the greedy character of the moves is decreased in some degree. The idea for another potential improvement originates from the observations of various other researchers. Almost all variable neighborhood search methods and hyper-heuristics use the 2-Swap operator - that is, two components of the current solution are exchanged. The idea has been further extended to make a 3-Swap. The h-HCGA algorithm was therefore changed to perform like a hyper-heuristic by augmenting the GHCM operator with 2-swap and 3-swap operators. The results were clear in that selecting the first better or equal move sequence rather than continuing and selecting the best one did not improve the solution. Including the 2-Swap and 3-Swap operators did not work either, in fact the solutions deteriorated on average. A possible reason for this might be that the GHCM operator alone is able to capture the benefits of the swapping. This observation is beneficial also for the reason that swapping takes

some time to complete. It still remains an open question of the present thesis why the swapping makes the results somewhat worse.

It is very difficult to classify the improved algorithm as one of the earlier mentioned metaheuristics. The basic hill-climbing step is extended to generate a sequence of moves in one step, leading from one solution to another as is done in the ejection chain method. The algorithm avoids staying stuck (i.e. the objective function value does not improve for some predefined number of generations) in the same solutions using tabu search and the refined simulated annealing method. The algorithm belongs to population-based methods in that it uses a population of solutions. This enables it to explore promising areas in the search space to a sufficient extent. A cooperative local search scheme is used in a sense that when an individual gets stuck in local minima it asks for the cooperation of the population in order to continue the search from another position. To escape from local optima the algorithm shuffles the current solution mimicing a hyper-heuristic mechanism. We named the improved algorithm PEAST (as Population, Ejection, Annealing, Shuffling and Tabu).

The outline of the PEAST algorithm is given in Figure 5. The algorithm is able to solve very challenging scheduling problems. The round-robin schedules of the Finnish major ice hockey leagues as well as the days-off and shift schedules of certain Finnish transportation companies can be efficiently generated using the improved algorithm.

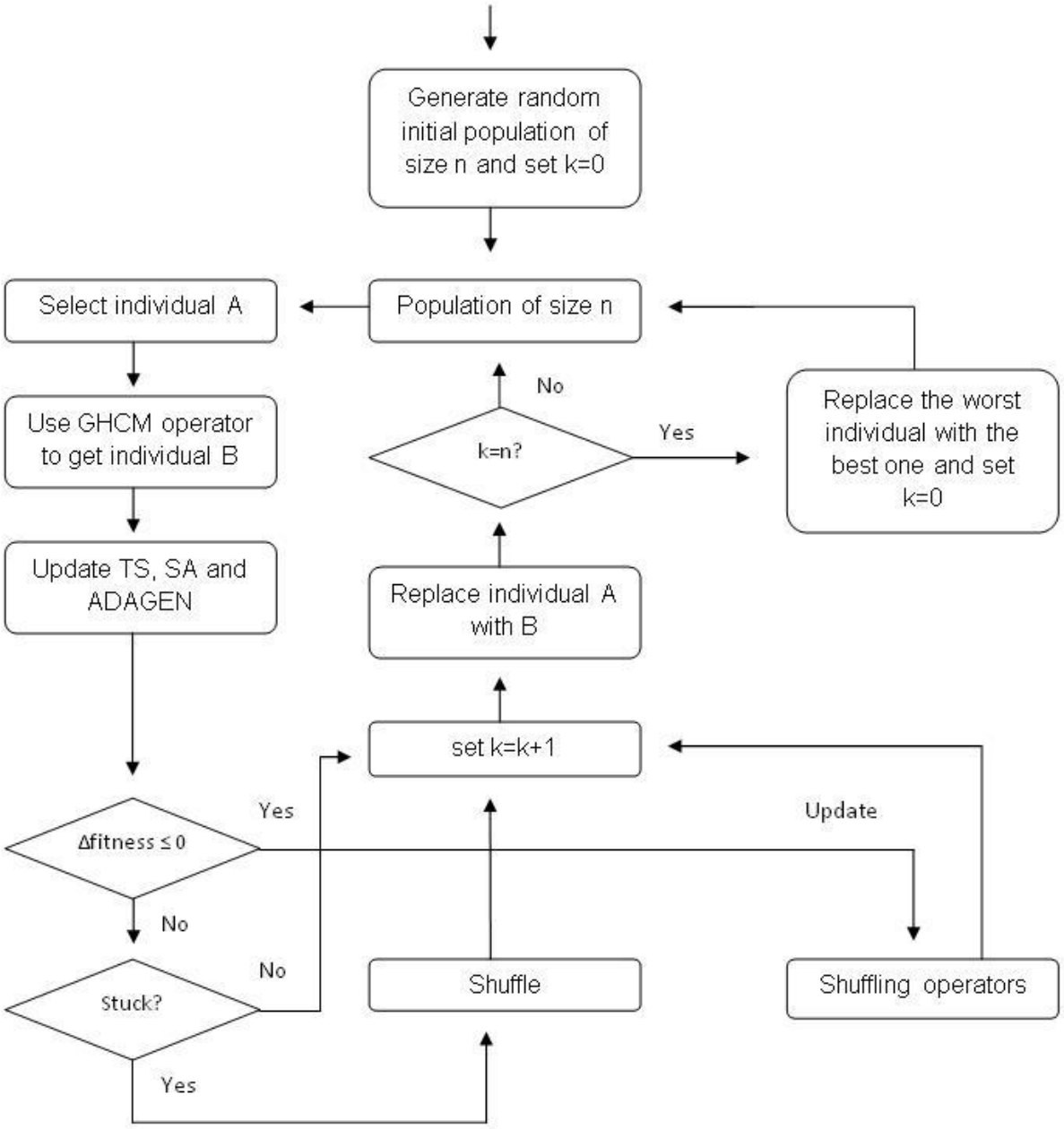


Figure 5. The outline of the PEAST algorithm.

We have recently started a profound inspection of the different heuristic features of the algorithm and especially the parameters used by these features. It has been noticed that to achieve the best solutions the algorithm must include shuffling and a tabu list. Preliminary test runs have shown that simulated annealing might be replaced by a faster, but very similar mechanism. The update of the ADAGEN method weights is also under development. In addition, we are currently studying the behavior of different parameter values and their combination to find out how to best apply them in solving real-world scheduling problems. This study has to be completed because the new PEAST algorithm has additional parameters compared to the h-HCGA algorithm, see [P1]. The findings will be reported in an upcoming publication (2011).

## Notes on the implementation of the algorithm

The first thing to be considered, when implementing the PEAST algorithm, was the choice of the programming language. The most important thing was that it should be very easy to install on the customers' computers. Installing any kind of service routines can lead to a chain reaction: one service routine needs another service routine, which in turn needs another and so on. It was also decided to not use any external software or software packages because their use would lead to license problems and to a situation where customers would have to buy one or more additional programs and/or licenses. The programming language had to generate fast object code and be object oriented. A high priority was also given to built-in data structures.

As a result of the above considerations the obvious choice was Java, but there were some suspicions of its speed. Practical tests with several programming languages revealed that Java is the choice. Java might not be the fastest language in any subcategory of possible speed measurements (CPU, memory, and so on) but it was fast enough for the implementation of the PEAST algorithm. Java is very easy to install and requires no service routines. It is supplied with a very good API-documentation and efficiently coded built-in data structures. It is fully object-based and it can be run without any modifications on (almost) any platform.

The implementation of PEAST utilizes almost every built-in data structure of Java. Furthermore, different data structures were combined in order to get the implementation as fast as possible. Sometimes it has been necessary to use ”not-so-pretty” structures to reach this goal. By not-so-pretty it is meant structures that make the code look and feel complicated. One example of such a case is a class holding job types, days-off types and the constraints relating to them. It is based on two (hard and soft constraints) HashMaps which map the types to the user defined names. These HashMaps are backed by two EnumMaps that contain the constraints (e.g. a late shift cannot be followed by an early shift). These EnumMaps contain an EnumSet because one shift type can have many constraints. The code does not look elegant but it is very fast. Normally one would never use such a complicated structure but with this structure we gained about 40% savings in the running time, compared to the traditional mapping done with pure Strings.

While making optimizations like above one should be well aware that usually the most important thing in programming is that the code is nicely and understandably written and that proper data structures are used in their proper places. Thus, in most cases, a small performance gain in running time is not worth the cluttering of the code. Though, in time consuming programs almost every decrease in running time is vital, no matter how small it is. In conclusion, the code of PEAST is very fast but maybe not the most beautiful one to read.

## 4. School Timetabling

In the next three sections (3, 4 and 5) we will describe school timetabling, sports scheduling and staff scheduling problems in detail. For each problem the mathematical model, NP-completeness or NP-hardness are illustrated. An implementation-oriented framework for each problem is given. In addition, the results of solving real-world problems are summarized. Finally, the cooperation with other researchers and business customers is discussed briefly.

School timetabling can be divided into at least three subcategories: *exam timetabling* [Carter, 1986], *course timetabling* (or university timetabling) [Tripathy, 1992] and *class timetabling* [de Werra, 1985]. In this thesis we concentrate on the class timetabling problem which occurs in Finnish schools and universities. The problem is to schedule teachers to classes and vice versa. The problem can be modified to include classrooms.

In a simplified class timetabling problem one has to assign teacher-class pairs to periods. Assume that there are $P$ periods, $C$ classes and $T$ teachers and the task is to assign teachers and classes to the periods in a way that no teacher or class is scheduled to the same period more than once. Let

$s_{ij}$ = 1, if teacher $t_j$ has been preassigned class $c_i$, otherwise 0
$x_{ijk}$ = 1, if class $c_i$ and teacher $t_j$ are assigned to period $p_k$, otherwise 0

and assume that each teacher-class pair occurs only once. Now the mathematical formulation of the basic class timetabling problem is as follows:

Determine the values of variables $x_{ijk}$ for $i$, $j$ and $k$, (3.1)

subject to the constraints

$$\begin{aligned}
&\textstyle\sum_{k\in P} x_{ijk} = s_{ij}, && i \in C, j \in T \\
&\textstyle\sum_{i\in G} x_{ijk} = 0 \text{ or } 1, && j \in T, k \in P \\
&\textstyle\sum_{j\in T} x_{ijk} = 0 \text{ or } 1, && i \in C, k \in P \\
&x_{ijk} = 0 \text{ or } 1, && i \in C, j \in T, k \in P
\end{aligned}$$

This formulation of the problem is exactly the same as the formulation of the well-known graph coloring problem [de Werra, 1985]. The problem belongs to the group of NP-complete problems and therefore the class timetabling problem is also an NP-complete problem.

In real-world school timetabling the scheduling must be completed in such a way that the solution is feasible and mostly acceptable to both school staff and teachers. A feasible solution satisfies the following hard constraints in the basic model (see [P1]):

1) No teacher, class or room is scheduled to the same period more than once.
2) No class is scheduled to the same period, if they have common students.
3) The given teachers are scheduled to the same period.
4) The given teachers are scheduled for predetermined periods.
5) For each class the daily minimum and maximum number of periods is respected.
6) A class $c_1$ is scheduled to earlier in the week than class $c_2$, if $c_1$ must precede $c_2$.
7) A lecture cannot be scheduled to periods where the teacher, the class or the room is unavailable.

A solution is most acceptable if it satisfies the following soft constraints:

1) The timetable of each class should have as few idle (leap) periods as possible.
2) The school day of some given classes should not start in the first period.
3) The school day of some given classes should end as early as possible.
4) The timetable of some given teachers should have as few idle periods as possible.
5) For some teachers the preferred daily minimum and maximum number of periods is given.
6) Some teachers prefer not to be scheduled in certain time periods.
7) Some teachers should be scheduled only on a limited number of days.
8) The different lessons of a subject should be on different days.

Timetabling has been done manually until the last decade when computers have been used to aid the task. However, intelligent algorithms and computer programs were not used in Finland in the year 2006 when we started to study the problem.

Algorithms for solving timetabling problems usually use a number of parameters to guide the process of learning/convergence. The values of these parameters play an important role in the overall performance of the algorithm − some combinations of parameter values turn often out to be better than others. A central question is how to find the best combination of the parameters.

In [P1] three artificial and three real-world school timetabling problem instances were solved. As described in Section 2, the main focus of the study was to find the best values of the parameters of the h-HCGA algorithm. The algorithm was controlled with seven parameters and each parameter could have 2-4 different values. The combination of the parameter values lead to a total of 972 configurations. So, in a real-world situation one should perform 972 test runs just to get every configuration tested once. Because we wanted to do an inclusive test and there was a strict time limit, we choose only one data set for our analysis. The most difficult real-world problem instance was then used in the tests.

For an inclusive test all the parameter combinations were considered. Real-world problems vary a great deal in difficulty and because there was only one problem instance, we decided to test one supporting method, F-Race [Birattari et al., 2002]. The method starts with all the possible parameter combinations and drops some of the combinations during the run. The dropping of the configurations is based on the Friedman test [Conover, 1999]. The results clearly implied that the Friedman test was suitable for choosing the parameter values. We were able to find parameter values that were very good for the introduced real-world and artificial school timetabling problems.

We solved a secondary school instance, a high school instance and a college instance. Even though we were able to find excellent results, the school administrators were not so keen on the optimized timetables. One reason for this might be that the schools do not have any funds to allocate to this kind of projects; even though the authorities could actually save money in the long run.

As a further study we created a conversion scheme for turning a curriculum-based timetabling problem into a school timetabling problem [Nurmi and Kyngäs, 2008]. The curriculum-based timetabling problem consists of the weekly scheduling of the lectures for several university courses within a given number of rooms and time periods. Curriculum-based timetabling differs from the school timetabling in that students enroll on courses and there is no student group concept. The motivation for the conversion was to show that a school timetabling algorithm can be used to solve

curriculum-based timetabling problems. As far as we know the publication was the first one on such a conversion scheme. The converted problem instances were solved using the h-HCGA school timetabling algorithm described in [P1]. The algorithm found a feasible solution within the given time limit for 12 of the 14 problem instances used in the 2nd International Timetabling Competition [McCollum and McMullan, 2010].

We also joined an international research group of seven researchers to create an XML format for benchmarks in high school timetabling, see [P2]. Together we described the high school timetabling problems in several countries in order to state a common set of constraints and objectives. The main goal was to provide exchangeable benchmarks for this problem. To achieve this we proposed a standard data format suitable for different countries and educational systems, defined by an XML schema. Later, four new researchers joined the group and we presented the progress on the benchmarking project for high school timetabling in [Post et al., 2010]. The timetabling archive currently includes 15 problem instances from seven countries and an evaluator capable of checking the syntax of instances and evaluating the solutions.

## 5. Sports Scheduling

Sports scheduling gained increased academic interest when the traveling tournament problem was introduced [Easton et al., 2001]. Before that time sports scheduling was mostly of theoretical nature. In the traveling tournament problem the most important goal is to minimize the total distance traveled by the teams. Another important goal is to avoid long home stands and away trips i.e. make the home and away matches vary sufficiently. The goal, minimizing the number of breaks, was of theoretical interest in the 1980s and 1990s (see e.g. Schreuder, 1980). Nowadays in practice, it is usually the most important one to minimize, regardless of other possible goals, excluding e.g. NHL and KHL ice hockey leagues because in these leagues the distances between the teams' home venues are remarkably lengthy.

Consider a single round robin tournament where every team plays against every other team exactly once. Every round has to be compact which means that every team must play on every round. Thus, if there are *n* teams, there must be exactly n-1 rounds. If n is odd a dummy team will be added. Each game consists of an ordered pair of teams (i, j), where team *i* is the home team and team *j* is the away team.

Let *T* be the set of teams, *P* the set of rounds and $c_{ijp}$ the cost of team *i* playing at home against team *j* at round *p.* Now the mathematical formulation of a simple sports scheduling problem is as follows:

$$\min \sum_{i \epsilon T} \sum_{j \epsilon T \setminus \{i\}} \sum_{p \epsilon P} c_{ijp} x_{ijp} \tag{4.1}$$

subject to the constraints

$$\sum_{p \epsilon P} (x_{ijp} + x_{jip}) = 1, \qquad \forall i, j \, \epsilon \, T, \; i < j$$

$$\sum_{j \epsilon T \setminus \{i\}} (x_{ijp} + x_{jip}) = 1, \qquad \forall i \, \epsilon \, T, \, p \, \epsilon \, P$$

$$x_{ijp} \, \epsilon \, \{0,1\}, \qquad \forall i, j \, \epsilon \, T, i \neq j, p \, \epsilon \, P$$

The first constraint guarantees that each team plays on exactly one round. The second constraint guarantees that each team plays exactly once per round. The binary variable $x_{ijp}$ is equal to 1 if and only if team *i* plays at home against team *j* at round *p*.

This problem is called the minimum cost single round robin tournament and it has been proven to be NP-hard [Easton 2002], [Briskorn et al., 2006]. Therefore the general sports scheduling problem is also an NP-hard problem.

In real-world sports scheduling, the games should be scheduled in rounds in such a way that the solution is feasible and mostly acceptable to both the league authorities and the teams. Some important requirements a sports league uses for its feasible schedule include (see [P3]):

1) Every team plays exactly once in every round (if a compact schedule is required).
2) A team cannot play at home on a certain day (e.g. a venue is unavailable).
3) Two teams cannot play at home on the same day (e.g. they share a venue).
4) A game must be pre-assigned to a certain round.

The quality of the final schedule is related to the correct optimization criteria and assignments given by the league. The league prefers to optimize many goals at the same time. Some important requests for acceptable schedules include:

1) A team cannot have more than two consecutive home games.
2) There must be at least *k* rounds before two teams meet again.
3) A team wishes to play most of its home games on certain weekdays.
4) Two teams do not want to play at home on the same day (e.g. they are located in the same region).

Although there have been a lot of published results in the field of sports scheduling [Knust, 2010], practical results have been modest. Some reasons for this might be that the real-world problems are too complicated, the understanding of the optimization is not good enough from the customer's side, and that researchers tend not to be good business men or even not interested to do business. At the time of writing we only know of a dozen cases where the researchers have been able to close a contract with a sports league owner [Nurmi et al., 2010].

The biggest sports leagues in various countries are big businesses. The goal for every team in a league, besides winning, is to maximize the income. Maximizing the income is very closely related to the number of spectators in games. This includes the spectators at the venue and also the spectators watching the game on TV. TV and other media require the attractive games to be scheduled at desired times. In addition, decreasing the traveling costs is important for some teams. Finally, professional sports leagues want to generate schedules that increase fairness; all teams should play under the same conditions.

The Finnish major ice hockey league (SM-Liiga) is the biggest sports league in Finland. It has most spectators, the players are all professionals and the media pays much attention to the league. The league has had about 5000 spectators per game for several years now. The second and third most attracting leagues, football (soccer) and Finnish baseball, have an average of 2000-3000 spectators per game.

In the halfway through the decade the schedule generation for SM-Liiga had become a challenging task. For various reasons there have been an increasing number of venues that cannot be used every day during the season. Furthermore, two of the teams cannot play on the same day because they share the venue. An increased number of additional constraints had complicated the schedule generation even more which is why it became almost impossible to make the schedule manually.

As an example of the quality of the major league schedules we generated (and 1st division schedules respectively), the number of rounds reduced from previous years average of 107 (96) down to 61 (48). As another example, the number of times a team plays three consecutive home games reduced from previous years average of 14 (23) down to 0 (2). Note that, as a result of the automated scheduler, the latest 1st division schedule had no three home games in a row.

The good results in sports scheduling triggered us the idea to gather a group of sports scheduling researchers to create a framework for scheduling professional sports leagues [P5]. The researchers were academics that had closed a contract with a sports league owner. The framework was modeled from the requirements of various professional sports leagues: Argentine volleyball, Austrian soccer, Belgian soccer, Brazilian soccer, Chilean soccer, Finnish ice hockey, German soccer and German handball. We have proposed a set of artificial and real-world

instances derived from the actual problems solved for professional sports league owners. The best solutions found thus far have been published, and the sports scheduling community has been invited to search for solutions to the unsolved instances, see [Nurmi, 2009].

The research on the topic is currently going on for scheduling the Finnish national ice hockey league for players aged under 20 [Nurmi et al., 2011]. Our algorithm found a feasible and acceptable schedule for the 2009-2010 season. To obtain this schedule, four distinct combinatorial problems were solved.

## 6. Staff Scheduling

Staff scheduling, and all of its subcategories, are becoming important problems in Finland and all around the world. A clear reason for this is that human resources are one of the most critical and most expensive resources for public institutions and private companies. In the past it was very common that the employee stayed at one company for his/her life span which made it quite easy to construct rosters. Nowadays the staff is under constant variation. In the past the employer dictated the rosters but today the employees have requirements for their rosters. Therefore, institutions and companies need decision support technologies for rostering. Careful planning can lead to significant improvements both in productivity and the welfare of the staff. Besides increasing employee satisfaction, effective labor scheduling can also improve customer satisfaction.

Staff scheduling generally includes *days-off scheduling* and *shift scheduling*. Days-off scheduling is quite a new research area [Alfares, 2001; Costa et al.,1997; Elshafei and Alfares, 2008; Pedrosa and Constantino, 2001]. In days-off scheduling the goal is to schedule the off-work days for each staff member over a given time period. Shift scheduling [Alfares, 2004; Brunner, 2009; Ernst et al., 2004; Lau, 1996], deals with the assignment of staff members to shifts. It can also specify the starting time and duration of shifts for a given day. In other words, days-off scheduling deals with working days and shift scheduling deals with the working times of day.

Consider a simple case of minimizing the number of workers to be hired. Let $W$ represent the set of $n$ workers and $D$ the set of $m$ working days. Let $w_i$ be the number of workers needed to work on day $i$. Let $A$ be the matrix of elements $a_{ij}$ which are equal to 1 if worker $i$ is available to work on day $j$, and zero otherwise. Finally, let $c_j$ be the cost of hiring the worker $j$. Decision variables $x_j$ indicate whether the worker $j$ is hired or not.

The function to be minimized is:

$$\min \sum_{j \in W} c_j x_j, \tag{5.1}$$

subject to constraints

$$\sum_{j \in W} a_{ij} x_j \geq w_i, \qquad i \in D$$

$$x_j \in \{0,1\}, \qquad j \in W$$

The first constraint guarantees that at least $w_i$ workers are working on day $i$. This formulation of the problem is the same as that of the well-known set covering problem. This problem belongs to the group of NP-complete problems [Beasley, 1987]. Minimizing workers at work is the foundation of staff scheduling. Also the very simple forms of shift scheduling are NP-hard [Lau, 1996].

There can be a very large number of requirements which must be met in staff scheduling. The coverage requirements ensure that there is a sufficient number of drivers on duty at all times. The regulatory requirements ensure that the driver's work contract and government regulations are respected. The operational requirements ensure that employees at work have correct competences and that they have acceptable days-off and shift structures. Some important requirements a company must consider include, see [P7]:

1) A minimum number of employees of particular competences must be assigned to each shift or each timeslot.

2) The required number of free weekends (both Saturday and Sunday free) within a timeframe must be respected.
3) Employees cannot work consecutively for more than *k* days.
4) At least *k* working days must be assigned between two separate days-off.
5) An employee assigned to a late shift type must not be assigned to an early shift on the following day.

Personnel's requests are very important and should be met as far as possible; this leads to greater staff satisfaction and commitment, and reduces staff turnover. Some important requests a company should consider include:

1) Single days-off and single working days should be avoided.
2) A balanced assignment of single days-off and single working days must be guaranteed between the drivers.
3) A balanced assignment of weekdays must be guaranteed between drivers.
4) Assign a requested day-on or avoid a requested day-off.

Most of the staff scheduling cases in which academic researchers have announced that they have closed a contract with a customer concern nurse rostering [Burke et al., 2004]. We started with shift scheduling for a Finnish bus transportation company. The company had had a lot of variation in the number of employees during the last couple of years. This had led to the situation where it had become impossible to use cyclic schedules. Cyclic schedules are such that all employees have the same basic schedule but start with a different day. In cyclic scheduling the goal is to find a schedule that is optimal for all employees. This is the current scheduling mechanism in almost all public institutions and private companies. Non-cyclic schedules are individual schedules, but they are extremely difficult to generate. Our shift scheduling research focuses on non-cyclic schedules. We solved the company's shift scheduling problem, see [P7].

However, it turned out that we should solve the days-off scheduling first. Two different days-off schedules were determined: one for the summer-time and one for the rest of the year. At the time of the scheduling the staff consisted of 65 employees. Nine employees formed three groups of three people. One of these groups always had to be at work. Furthermore, six employees could not work on weekends. These two requirements and a dozen other ones made the days-off scheduling quite challenging but still solvable with PEAST [Nurmi and Kyngäs, 2011].

As was the case in sports scheduling, the good results in staff scheduling triggered us to gather a small group of staff scheduling researchers to create a framework for implementation-oriented staff scheduling [P6]. The two researchers invited to this collaboration had solved real-world staff scheduling instances. The framework was modeled from the requirements of various lines of business and industry. We proposed a set of artificial and real-world instances derived from the actual problems solved for various companies. The best solutions found were published and the staff scheduling community was invited to challenge these results. The instances will be available online.

A days-off schedule on a yearly basis and shifts on a monthly basis in one of the Finnish bus transportation companies are discussed in [P7]. The generated software is integrated with a third-party vendor product. The vehicle scheduling and the driver scheduling phases turn out to be very interesting and hard topics of further studies.

## 7. Summary and Conclusions

The idea of this thesis was to show that challenging school timetabling, sports scheduling and staff scheduling problems can be solved to customer satisfaction using an intelligent algorithm and sophisticated programming.

The difference of hard and easy problems was first discussed. A literature summary was also given. Next, an introduction to general metaheuristics was given, followed by the description of the algorithm used to solve challenging scheduling problems. The algorithm is called PEAST. It owns features from hill-climbing, ejection chain method, tabu search, simulated annealing, population-based methods, cooperative local search scheme and hyper-heuristics. Next, the details of the school timetabling, sports scheduling and staff scheduling problems were discussed. Mathematical models, implementation-oriented models, practical results and cooperation partners were presented for each problem type.

The most important conclusion from the above considerations is that scheduling problems can be solved to customer satisfaction. The best action plan for implementation-oriented research is to cooperate both with a problem owner and with a third-party vendor. A researcher should maybe not work with user interfaces, financial management links, customer reports, help desks, etc. Instead, one should concentrate on algorithmic power. Though, this is only an opinion based on experience.

It is apparent that a profound understanding of the relevant requests and requirements presented by customers is a prerequisite for implementing and solving real-world scheduling problems. The scheduling research should concentrate on the acceptance and satisfaction of both the problem owner (company), the managers using the implemented software and the end-users. It should be noted that it is not easy to incorporate the experience and expertise of the managers into a working scheduling system. The managers often have extremely valuable knowledge, experience and detailed understanding of their specific problem, which will vary from company to company. To formalize this knowledge to the software is not an easy task. Still, the results of this research have shown that it can be done successfully, and that the intelligent algorithms can produce superior results.

The final conclusion is that the academics focusing on implementation-oriented research should create systems that

1) generate solutions that are not too accurate compared to the input data and real-world use,
2) generate just a few solutions to choose from,
3) generate clearly different solutions to choose from,
4) allow users to specify the importance of requests and requirements,
5) minimize the scheduling time,
6) run on any modern computer with any operating system,
7) do not use third-party mathematical software packages with expensive licensing policies and
8) are integratable with existing industry software.

## References

Alfares H.K. 2001. Efficient optimization of cyclic labor days-off scheduling. *OR Spektrum 23*, 283-294.

Alfares H.K. 2004. Survey, categorization and comparison of recent tour scheduling literature, *Annals of Operations Research* 127, 145-175.

Applegate D., Bixby R.E. and Cook W.J. 2007. The Traveling Salesman Problem: A Computational Study, *Princeton University Press*, Princeton, NJ.

Beasley J.E. 1987. An algorithm for set covering problem. *European Journal of Operational Research 31*, 85-93.

Birattari M., Stützle T., Paquete L., Varrentrapp K. 2002. A racing algorithm for configuring metaheuristics. *Proceedings of the Genetic and Evolutionary Computation Conference,* Morgan Kaufmann, San Francisco, CA, USA, 11-18.

Blazewicz J., Domschke W. and Pesch E., 1996. Job Shop Scheduling Problem: Conventional and New Solution Techniques, *European Journal of Operational Research 93*, 1-33.

Blum C., Roli A. 2003. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, *ACM Computing Surveys 35(3),* 268-308.

Briskorn D., Drexl A., Spieksma F.C.R. 2006. Round robin tournaments and three index assignment, *Working Paper.*

Brunner J.O., Bard J.F., and Kolisch R. 2009. Flexible shift scheduling of physicians, *Health Care Management Science 12(3),* 285-305.

Burke E.K., De Causmaecker P. and van den Berghe G. 2004. The State of the Art of Nurse Rostering, *Journal of Scheduling 7*, 441-499.

Burke E.K., Hart E., Kendall G., Newall J., Ross P., and Schulenburg S. 2003. Hyper-heuristics: An emerging direction in modern search technology, *Handbook of Metaheuristics (F. Glover and G. Kochenberger, eds.)*, Kluwer, 457–474.

Carter M.W. 1986. A Survey of Practical Applications of Examination Timetabling Algorithms. *OR Practice 34,* 193-202.

Colorni A., Dorigo M., Maniezzo V. 1992. Distributed Optimization by Ant Colonies. In F. J. Varela and P. Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 134-142. MIT Press, Cambridge, MA.

Conover W.J. 1999. Practical nonparametric statistics, John Wiley & Sons, USA.

Costa M.C., Jarray F., Picouleau C. 2006. An acyclic days-off scheduling problem, *4 OR 4*, 73-85.

Easton K. 2002. Using integer programming and constraint programming to solve sports scheduling problems, *Ph.D. thesis*, Georgia Institute of Technology, USA.

Easton K., Nemhauser G. and Trick M. 2001. The Traveling Tournament Problem: description and benchmarks. *Proceedings of the 7th. International Conference on Principles and Practice of Constraint Programming,* 580–584, Paphos.

Easton K., Nemhauser G. and Trick M. 2004. Sports scheduling. *In Handbook of Scheduling: Algorithms, Models and Performance Analysis (Leung, Ed.),* CRC Press Inc, Florida, USA, 1–19.

Eberhart R.C., Kennedy J. 1995. A New Optimizer Using Particle Swarm Theory, *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, Nagoya, Japan, 39-43.

Elshafei M., Alfares H.K. 2008. A dynamic programming algorithm for days-off scheduling with sequence dependent labor costs, *J Sched 11*, 85-93.

Ernst A. T., Jiang H., Krishnamoorthy M. and Sier D. 2004. Staff scheduling and rostering: A review of applications, methods and models, *European Journal of Operational Research 153*, 3-27.

Garey M.R., Johnson D.S. 1979. Computers and Intractability - A Guide to NP-completeness. W.H. Freeman.

Glover F., McMillan C., Novick B. 1985. Interactive Decision Software and Computer Graphics for Architectural and Space Palnning, *Annals of Operations Research 5,* 557-573.

Glover F. 1992. New ejection chain and alternating path methods for traveling salesman problems. *In Computer Science and Operations Research: New Developments in Their Interfaces, edited by Sharda, Balci and Zenios*, Elsevier, 449–509.

Goldberg D.E. 1989. Genetic Algorithms in Search, *Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA, USA.

Hansen P., Mladenovi´c N. 2001. Variable neighbourhood search: Principles and applications. E*uropean Journal of Operations Research 130*, 449–467.

Hecht-Nielsen R. 1990. Neurocomputing. Addison Wesley.

Holland J.H. 1975. Adaption in Natural and Artificial Systems, University of Michigan Press.

Jain A.S. and Meeran S. 1999. Deterministic Job-Shop Scheduling: Past, Present and Future, *European Journal of Operational Research 113*, 390-434.

Kirkpatrick S., Gelatt C.D. Jr., Vecchi M.P. 1983. Optimization by Simulated Annealing, *Science 220,* 671-680.

Kennedy J., Eberhart R.C. 1995. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks,* Piscataway, 1942-1948.

Kohonen T. 1984. Self-organization and Associative Memory. Springer-Verlag: New York. (2nd Edition: 1988; 3rd edition: 1989).

Knust S. (Last update 12.10.2010). Classification of Literature on Sports Scheduling [Online], Available: http://www.informatik.uni-osnabrueck.de/knust/sportlit_class/, Accessed 3.1.2011.

Landa Silva J.D. 2003. Metaheuristic and Multiobjective Approaches for Space Allocation, *Ph.D. Dissertation*, School of Computer Science and Information Technology, University of Nottingham, UK.

Lau H.C. 1996. On the Complexity of Manpower Shift Scheduling, *Computers and Operations Research 23(1)*, 93-102.

McCollum B. 2006. University Timetabling: Bridging the Gap between Research and Practice, *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling,* Brno, Czech Republic, 15-35.

McCollum B. and McMullan P. (Last update 10.5.2010). The 2nd International Timetabling Competition [Online], Available: http://www.cs.qub.ac.uk/itc2007/. Accessed 3.1.2011.

Moscato P. 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Technical Report 826*, California Institute of Technology, Pasadena, California.

Muhlenbein H. 1989. Parallel genetic algorithms, population genetics and ombinatorial optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, 416-421.

Neveu B., Trombettoni G. and Glover F. 2004. Idwalk : A candidate list strategy with a simple diversification device. *Lecture Notes in Computer Science 3258*, 423–437.

Nurmi K. 1998. Genetic Algorithms for Timetabling and Traveling Salesman Problems, *Ph.D. dissertation*, Dept. Applied Math., University of Turku, Finland.

Nurmi K. and Kyngäs J. 2008. A Conversion Scheme for Turning a Curriculum-based Timetabling Problem into a School Timetabling Problem in *Proc of the 7th Conference on the Practice and Theory of Automated Timetabling (PATAT)*, Montreal, Canada.

Nurmi K. 2009. Sports Scheduling Problem [Online]. Available: http://www.samk.fi/ssp/, Accessed 11.1.2011.

Nurmi K. and Kyngäs J. 2009. Improving the Schedule of the Finnish Major Ice Hockey League in *Proc of the 2nd International Conference on the Mathematics in Sport*, Groningen, Netherlands.

Nurmi K., Goossens D. and Kyngäs J. 2011. Scheduling a Triple Round Robin Tournament for the Finnish National Ice Hockey League for Players Under 20 in *Proc of the IEEE Symposium on Computational Intelligence in Scheduling*, Paris, France.

Nurmi K. and Kyngäs J. Days-off Scheduling for a Bus Transportation Staff, *International Journal of Innovative Computing and Applications*, Inderscience, UK, to be published 2011.

Osman I.H., Laporte G. 1996. Metaheuristics: A bibliography, *Annals of Operations Research 63,* 513-623.

Pedrosa D. and Constantino M. 2001. Days-off scheduling in public transport companies, *Computer-aided scheduling of public transport, Voß, S. et al (eds.), Lect. Notes Econ. Math. Syst. 505*, 215-232.

Post G., Kingston J. H., Ahmadi S., Daskalaki S., Gogos C., Kyngäs J., Nurmi K., Santos H., Rorije B. and Schaerf A. 2010. An improved XML format and benchmark problems for High School Timetabling in *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling (PATAT),* Belfast, Ireland.

Preux Ph., Talbi E-G. 1999. Towards Hybrid Evolutionary Algorithms, *International Transactions in Operational Research 6*, 557-570.

Rasmussen P. and Trick M. 2008. Round robin scheduling - A survey. *European Journal of Operational Research 188*, 617–636.

Rumelhart D.E. and McClelland J.L. 1986. Parallel Distributed Processing: Explorations in *Microstructure of Cognition (volumes 1 & 2)*, MIT Press.

Schaerf A. 1999. A Survey of Automated Timetabling. *Artificial Intelligence Review 13/2*, 87-127.

Schreuder J.A.M. 1980. Constructing timetables for sport competitions, *Mathematical Programming Study 13,* 58–67.

Syswerda G. 1989. Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 2-9.

Toth P. and Vigo D. 2001. The Vehicle Routing Problem, *Monographs on Discrete Mathematics and Applications*, SIAM, Philadelphia.

Tripathy A. 1992. Computerised Decision Aid for Timetabling – A Case Analysis. *Discrete Applied Mathematics 35,* 313-323.

de Werra D. (1985): An introduction to timetabling. *European Journal of Operations Research 19,* 151-162.

# Publication 1

K. Nurmi and J. Kyngäs, ”A Framework for School Timetabling Problem”. In Proc of the 3rd Multidisciplinary Int. Scheduling Conf.: Theory and Applications (MISTA), Paris, France, 2007, pp. 386-393.

# A Framework for School Timetabling Problem

Kimmo Nurmi, Jari Kyngäs
Satakunta University of Applied Sciences, Finland, Tiedepuisto 3, FIN-28600 Pori
{kimmo.nurmi, jari.kyngas}@samk.fi

**Abstract:** This paper introduces a framework for a highly constrained school timetabling problem, which was modeled from the requirements of various Finnish school levels. We present a successful algorithm to solve real-world problems as well as artificial test problems. Moreover, we find the best configuration for this algorithm using brute force and statistical analyses. Finally, we propose a set of benchmark problems that we hope the researchers of the timetabling problems would adopt.

*Keywords:* Evolutionary Algorithms, Heuristic Search, Real World Scheduling, Timetabling.

## 1. Introduction

In recent years many solution approaches for different timetabling problems have been introduced. Most of the work has concentrated on examination timetabling, university timetabling and course timetabling [1,2,3,4,5,6]. Timetabling researchers have obtained very promising and practicable results in these problem areas. However, school timetabling has not been as extensively studied, and widely usable results are not yet available. The school timetabling problem [7] consists of assigning lectures to periods in such a way that no teacher, class or room is involved in more than one lecture at a time and other constraints (hard and soft) are satisfied.

The main focus of this paper is to introduce a successful algorithm to tackle highly constrained school timetabling problems [8,9]. The algorithm is controlled with nine different parameters. For seven of these parameters we searched the best values using both brute force and statistical analyses. Two different methods were mainly used because we wanted to confirm that the chosen parameter values were the best ones. Two parameters were excluded because they have a great impact on the running time and therefore make the results incomparable (see Chapters 2-4).

Another point of this paper is to propose a set of benchmark instances and publish them as well as our results on the web. We wanted to lay out the foundation of comparable results (see Chapter 5). This idea is presented (by Schaerf and Di Gaspero) in [10].

## 2. Problem Description

We describe a school timetabling problem that arises in various Finnish school levels: secondary school, high school and college. This problem description is representative of many timetabling scenarios within the area of school timetabling. The timetabling is based on the following conditions:

- The timetable frame consists of $n$ weeks; each week has $m$ days and each day $t$ periods (timeslots).
- The lecture is a predefined combination of a teacher, a class (or course), a room and the length of the lecture in periods.
- The set of teachers, classes/courses and rooms is fixed.
- More than one teacher can teach a particular class at the same time.
- Each class should have at least a given number of periods in a day, but should not have more than another given number of periods.
- The classes/courses can have common students.
- Some classes/courses must precede other classes/courses
- The rooms can be classified to certain room types.

*The school timetabling problem* consists of scheduling the predefined lectures in such a way that the solution is feasible and mostly acceptable to both school staff and teachers. A feasible solution satisfies the following *hard constraints*:

1. No teacher, class/course or room is scheduled to the same period more than once (basic model).
2. No class/course is scheduled to the same period, if they have common students.
3. The given lectures are scheduled to the same period.
4. The given lectures are scheduled for predetermined periods.
5. For each class the daily minimum and maximum number of periods is respected.
6. A class/course $c_1$ is scheduled to earlier in the week than class/course $c_2$, if $c_1$ must precede $c_2$.
7. A lecture cannot be scheduled to periods where the teacher, the class or the room is unavailable.

A solution is most acceptable if it satisfies the following *soft constraints*:

1. The timetable of each class should have as few idle (leap) periods as possible.
2. The school day of some classes should not start in the first period.
3. The school day of some classes should end as early as possible.
4. The timetable of some teachers should have as few idle periods as possible.
5. For some teachers the preferred daily minimum and maximum number of periods is given.
6. Some teachers prefer not to be scheduled in certain time periods.
7. Some teachers should be scheduled only on a limited number of days.
8. The lessons of a subject should be on different days.

The above formulation covers school timetabling problems occurring in Finnish secondary schools, high schools and colleges. In lower school levels the problem reminds the basic school timetabling problem and in higher levels it is more similar to a combination of the school timetabling problem and the course timetabling problem [11].

## 3. The Algorithm

The algorithm presented here is an extension of the *h-HCGA* algorithm presented in [12]. We have spent "quiet timetabling research life" since the development of the algorithm and returned to the problem only recently, when we were asked to schedule the timetables of one secondary school and one high school. First we spent quite a lot of time trying to improve our original algorithm with the ideas of Ant Algorithms [13, 14]. Unfortunately we were not able to find competing results. We moved into improving the parameter values of the *h-HCGA* algorithm. Table 1 summarizes the parameters and the values from which we were searching the best configuration. Because two of the parameters have a great effect on computational time, we fixed their values in order to keep the results comparable.

| **Extended h-HCGA parameters** | |
|---|---|
| | |
| F. Population size | 20 *(fixed)* |
| 1. Reproduction/selection | Random-average, Random-best, Marriage-best |
| | |
| F. Maximum move sequence | 10 *(fixed)* |
| 2. Tabulist size for the GHCM-operator | 0, 5, 10 |
| 3. Tabulist size for overall moves | 0x, 1.5x, 2x, 4x (maximum move sequence) |
| | |
| 4. Type of removal check | All, Hard, Hard or Soft |
| 5. Fitness calculation | Absolute, Weighted |
| 6. Update frequency of hard constraint weights | 5, 10, 20 |
| | |
| 7. Prevention of same lessons | No, Yes |

**Table 1:** Parameter summary of the extended h-HCGA algorithm.

The algorithm is a genetic algorithm [15,16] with one mutation operator and no recombination operators. The two most important features of the algorithm are *the greedy hill-climbing mutation (GHCM) operator*, which generates a new solution candidate from the current solution and *the*

*adaptive genetic penalty method (ADAGEN)*, which is a multiobjective optimization method [17,18,19,20].

The GHCM operator is based on moving a lecture $l_1$ from its old period $p_1$ to a new period $p_2$, moving another lecture $l_2$ from period $p_2$ to a new period $p_3$ and so on, ending up with a sequence of moves. The initial lecture selection is random. The new period for the lecture is selected considering all possible periods and selecting the one which causes the least increase in the cost function when considering the relocation cost only. Moreover, the new lecture from that period is again selected considering all the lectures in that period and picking the one for which the removal causes the most decrease in the cost function when considering the removal cost only. The operator stops if the last move causes an increase in the cost function value and if the value is larger than that of the previous non-improving move.

We noted in [12] that we can improve the GHCM operator by introducing a tabu list, which prevents reverse order moves in the same sequence of moves. We can also prevent only some of the moves (parameter 2). To further widen the use of the tabu list, we can prevent reverse order moves in consecutive move sequences (parameter 3), that is in consecutive applications of the GHCM operator.

The ADAGEN method is an adaptive penalty method for multiobjective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints. The weights are updated in every $k^{th}$ (parameter 6) generation using a somewhat complicated formula [12].

The reproduction operation [21] of the (genetic) algorithm is based to a certain extent on the steady-state reproduction [22]. We select randomly a timetable from the population of timetables for single GHCM operation. The new timetable replaces the old one if it has better or equal fitness. We have three variations (parameter 1). In the first one the new timetable replaces also the least fit in the current population if it is better than the current population average. In the second one the least fit is replaced with the best one, when $n$ better timetables have been found, where $n$ is the size of the population. In the third one we use the second variation but select a timetable using a marriage selection [23].

When selecting a lecture to be removed from a period (parameter 4), we can consider either all the lectures in that period or only those that have at least one constraint violation with the lecture that was moved to that period previously. We can also consider only hard constraint violations. When calculating the best lecture to be removed and the best period to move to (parameter 5), we can use either absolute number of constraint violations or the weighted penalty value of the ADAGEN method.

We also tried (parameter 7) to programmatically prevent lessons of a subject to be on the same day (soft constraint 8). Another possibility would have been to change it to a hard constraint.

In preliminary experiments the extended h-HCGA algorithm found encouraging results. Some parameter values seemed to produce better results than others. Our next goal was to find the best configuration for the parameter values.

## 4. Finding the Configuration

We had two possible choices for the process of finding out the best configuration of parameters: brute force and statistical methods. The brute force approach was very attempting because we had the possibility to use up to 78 computers. We decided to use brute force but also analyze the runs using standard statistical methods. We also performed "what-if" analyses on the runs using the

Friedman test [24] in similar fashion to Birattari & al. [25]. We used the highly constrained benchmark instance C3 (see chapter 5) in our test runs.

For the runs to be comparable in computing time we had to fix two parameter values (see table 1). The rest of the parameter values were such that they did not influence the convergence time. The total number of configurations was 1296 but because the *tabulist size for overall moves* has no effect when *tabulist size for GHCM-operator* is zero, we were left with 972 configurations.

Our goal was to find a configuration which would minimize the hard constraints to zero. The configurations were ranked according to hard constraints. We decided to run every configuration five times and for two hours each, on an AMD Athlon 1700+ with 512Mb of memory. For every generation of the algorithm we saved the number of hard constraint violations.

At first we only considered those runs that had ended up with zero hard constraint violations ─ 218 out of 972 runs met this criterion.

## 4.1. Choosing the Configuration

We started analyzing the results by counting how many times each value of each parameter occurred in the best runs. Table 2 summarizes the results.

| Parameter | Values and occurrences |
|---|---|
| 1. Reproduction/selection | **Random-average = 38%**, **Random-best = 40%**, Marriage-best = 22% |
| 2. Tabulist size for GHCM-operator | 0 = 20%, **5 = 42%**, **10 = 38%** |
| 3. Tabulist size for overall moves | **0x = 18%**, **1.5x = 25%**, **2x = 26%**, **4x = 31%** |
| 4. Type of removal check | All = 29%, **Hard = 63%**, Hard or soft = 8% |
| 5. Fitness calculation | **Absolute = 64%**, Weighted = 36% |
| 6. Update frequency of hard constraint weights | **5 = 48%**, 10 = 30%, 20 = 22% |
| 7. Prevention of same lessons | **No = 77%**, Yes = 23% |

**Table 2:** The chosen values (bolded) after brute force runs.

The best value for the last four parameters could easily be chosen (tested with the t-test). The first three, on the other hand, required further investigation.

Next we ran the chi-squared test [24] pairwise between all parameters. We were especially interested in those test cases which included the chosen parameter values. Unfortunately none of the chi-squared tests showed statistically significant dependencies among the parameter values. We enlarged the chi-squared test to include also those runs that had ended up with one hard constraint violation, but that did not change the test results. (These statistical tests can be obtained from us at request.)

At this phase we were still not able to find the values for the first three parameters. However, we were able to reduce the number of free parameters from 972 to 16 (the combination of chosen parameter values of each parameter). For the *reproduction/selection*, the *tabulist size for GHCM-operator* and the *tabulist size for overall moves* we were not able to choose one specific value. That strengthened our decision to use statistical methods.

## 4.2. Statistical Methods

Birattari & al. [25] have developed a process, F-Race, in which they start with a considerable large population and eliminate individuals from it with the help of statistical methods. The idea is to eliminate an individual from the population as soon as it can be statistically significantly proven that the individual is not performing good enough. The process can be thought of as a race where

every individual competes against each other. At specific intervals the performance of each individual is checked and the poorly performing ones are dropped out from the population. This process is well suited for problems with extensive number of parameters.

We tested this method to see how it would have worked in this case. However, we did not use the actual racing algorithm ─ we only tested the runs to see which ones would have dropped out after two hours of running. From the output of the runs we sampled ten evenly spaced observations. This meant that we had 972 treatments (configurations) and 10 blocks (samples) for the Friedman test.

Table 3 summarizes the results of the Friedman test for the case of the eliminated runs. When we compare this with table 2 we can see that these results are very compatible with each other. For example, table 2 tells us that in the 218 runs 63% of the configurations had parameter *type of removal check* set to value *Hard*. Table 3, on the other hand, tells us that only 7% of the eliminated runs included configurations where this parameter was set to value *Hard*. The information in the two tables supports each other very well.

| Parameter | Values and occurrences |
|---|---|
| 1. Reproduction/selection | **Random-average = 21%**,<br>Random-best = 34%, Marriage-best = 45% |
| 2. Tabulist size for GHCM-operator | 0 = 45%, **5 = 24%**, 10 = 31% |
| 3. Tabulist size for overall moves | 0x = 29%, 1.5x = 20%, 2x = 28%, 4x = 23% |
| 4. Type of removal check | All = 21%, Hard = 7%, Hard or soft = 72% |
| 5. Fitness calculation | Absolute = 14%, Weighted = 86% |
| 6. Update frequency of hard constraint weights | 5 = 16%, 10 = 28%, 20 = 56% |
| 7. Prevention of same lessons | No = 30%, Yes = 70% |

**Table 3:** The chosen values (bolded) in eliminated configurations.

In the brute force runs we were not able to choose the best values for the first three parameters. Here the best value for the *reproduction/selection* is *Random-average*. The difference between the values *Random-average* and *Random-best* is statistically significant at 0.001 level when measured with the t-test. Therefore we chose the value *Random-average* for parameter *reproduction/selection.*

The other two parameters show no statistical difference between the values. We chose value *5* for the parameter *tabulist size for GHCM-operator* because both tables indicate that this could be the best value (though not statistically significantly differing from value *10*). The parameter *tabulist size for overall moves* shows no difference in goodness of the values. The value was chosen to be *2* because we *felt* it could be the best value.

## 5. Standard Benchmark Instances

In school timetabling we do not have a set of standard test problems, as is the case in examination timetabling [10]. We now introduce some test instances and hope that they will lay the foundation for the standard universal benchmark instances for school timetabling problems.

The first set of test problems consists of *all-N-problems* introduced in [12]. These artificial problems have a timeframe of $N$ days with $N$ periods in one day totaling $N^2$ periods. There are $N$ teachers, $N$ classes and $N$ rooms. All possible combinations of a teacher, a class and a room are to be scheduled. This will give us $N^3$ lectures to be scheduled. The problem is quite difficult to solve since a feasible solution has no idle periods for any teacher, class or room.

The second set of test problems consists of *Abramson-N-problems* introduced in [26]. These artificial problems have a timeframe of 30 periods. The lectures are built by first choosing a random

teacher and class and room identifiers and then placing that lecture in the first possible period. If the lecture cannot be placed in an available period, then it is discarded and another one is generated. This process continues until all teacher, class and room identifiers have been used. The resulting timetable has again no idle periods and thus is tightly constrained.

Both *all-N-problems* and *Abramson-N-problems* are excellent test instances since

a) every single school timetabling algorithm can tackle them because they are the simplest version of the problem (basic model) and
b) we can easily increase the problem difficulty by increasing *N*.

| Problem identifier | B3 | S3 | H3 | C3 |
|---|---|---|---|---|
| School level | Artificial school | Secondary school | High school | College |
| | | | | |
| #Weeks | 1 | 1 | 1 | 1 |
| #Days | 5 | 5 | 5 | 5 |
| #Periods per day | 4 | 7 | 7 | 8 |
| | | | | |
| #Teachers | 21 | 25 | 18 | 46 |
| #Classes/courses | 11 | 14 | 50 | 41 |
| #Rooms/room classes | 3 | 25 | 13 | 34 |
| | | | | |
| #Lectures | 169 | 280 | 219 | 387 |
| #Periods in lectures | 200 | 306 | 320 | 854 |
| | | | | |
| #Clashes (hard constraint 1) | 3020 | 8590 | 6420 | 28574 |
| #Overlapping classes/courses (hard constraint 2) | 11 | 0 | 52 | 20 |
| #Unavailabilities (hard constraint 7) | 108 | 600 | 72 | 328 |
| | | | | |
| Soft constraint 1 in use | Yes | Yes | No | Yes |
| Soft constraint 4 in use | Yes | Yes | Yes | Yes |
| Soft constraint 6 in use | No | Yes | Yes | No |
| Soft constraint 8 in use | Yes | Yes | Yes | Yes |

**Table 5:** Summary of one artificial and three real-world problems.

The third set of problems consists of one artificial problem and three small to medium-sized real-world problems in Finnish schools, one from each school level. Table 5 summarizes these four problems. The data for these problems has been published on the web [27]. In all of these problems we are searching for a feasible solution which optimizes soft constraints 1, 4, 6 and 8. Other soft constraints are not in use. These four problems are good test instances since

a) most of the school timetabling algorithms should be able to tackle them because they are not too complicated versions of the problem,
b) they are quite moderate-sized and
c) they are different from each other.

The C3 problem is the most constrained and thus the most interesting one. We solved six problems from the standard benchmark instances: All-11-problem, Abramson-15-problem, B3, S3, H3 and C3. We used the parameter values found in the statistical analysis. To keep the results fair we did no further fine-tuning to the extended h-HCGA algorithm. The test runs were performed using the same computers as in Chapter 4. Table 6 summarizes the results.

Furthermore, we decided to run the algorithm using a simulated annealing refinement introduced in [12]. When the GHCM operator selects a new period for a lecture $l_i$, we apply the following selection mechanism. Periods are examined in random order. A period is selected as the current best one if its fitness is less than the current best one or if the current annealing temperature [28] ac-

cepts their difference. The same mechanism is used in the GHCM operator while removing a lecture. Note that the GHCM operator does not accept upward (increasing cost) move sequences, even if a single move can be upward. Table 6 summarizes the results using the simulated annealing refinement. It also shows the best manual solution the staff of the schools were able to find. The best solutions we have found in all of our experiments can be found in [27].

| Problem identifier | Nbr of runs | Single run time | Best | Median | Worst | Best manual |
|---|---|---|---|---|---|---|
| Extended algorithm | | | | | | |
| All-11 | 8 | 8 hours | 0-0 | 0-0 | 0-0 | |
| Abramson-15 | 8 | 8 | 2-0 | 3-1 | 4-0 | |
| B3 | 8 | 8 | 0-3 | 1-0 | 1-6 | |
| S3 | 8 | 8 | 0-2 | 0-2 | 0-3 | |
| H3 | 8 | 8 | 0-3 | 0-3 | 0-3 | |
| C3 | 8 | 8 | 0-2 | 0-4 | 0-6 | |
| With SA refinement | | | | | | |
| All-11 | 8 | 24 hours | 0-0 | 0-0 | 0-0 | |
| Abramson-15 | | | 2-0 | 2-0 | 3-1 | |
| B3 | | | 0-2 | 0-5 | 1-1 | |
| S3 | | | 0-2 | 0-2 | 0-2 | 0-8 |
| H3 | | | 0-3 | 0-3 | 0-3 | 0-12 |
| C3 | | | 0-1 | 0-1 | 0-3 | 0-10 |

**Table 6:** Results for standard benchmark instances and the best manual solutions. For example, the value 1-6 stands for one hard constraint and six soft constraint violations.

## 6. Conclusions and further work

In this paper we considered a highly constrained school timetabling problem and presented a successful algorithm to solve real-world problems as well as artificial test problems. Our algorithm performs very well in the timetabling problems presented in this paper. In order to find out if statistical methods could be of any help in solving the parameter values we ran our algorithm with all possible configurations and analyzed the results using the Friedman test. The Friedman test clearly implied that it is able to extract poor configurations in favor of the good ones. In the near future, we will build the racing process into our program. Our research has shown that the chosen parameter values were very good for the problem C3, but we do not know if they are the best for all problems. Therefore it is best to start with as many configurations as possible and reduce them in time. We have developed a web page [27] that allows other researchers to download the problem description, the standard benchmark instances and the computational results.

## References

[1] E.K. Burke and P. De Causmaecker (2003). The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th International conference, Gent 2002, Springer Lecture Notes in Computer Science, vol. 2740, Springer.

[2] E.K. Burke and M. Trick (2005). The Practice and Theory of Automated Timetabling V: Revised Selected Papers from the 5th International conference, Pittsburgh 2004, Springer Lecture Notes in Computer Science, vol. 3616, Springer.

[3] E.K. Burke and Hana Rudová (2006). Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Masaryk University.

[4] A. Schaerf (1999), A Survey of Automated Timetabling, Artificial Intelligence Review **13**(2), 87 – 127.

[5] E.K. Burke and S. Petrovic (2002). Recent Research Directions in Automated Timetabling, European Journal of Operational Research **140**(2), 266 – 280.

[6] B. McCollum (2006). University Timetabling: Bridging the Gap between Research and Practice. In *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, 15 – 35.
[7] D de Werra D (1985). An Introduction to Timetabling. European Journal of Operations Research **19**, 151–162.
[8] S. Even, A. Itai, and A. Shamir (1976). On the complexity of timetable and multicommodity flow problems. SIAM Journal on Computing **5**, 691 – 703.
[9] T.B. Cooper and J.H. Kingston (1996). The Complexity of Timetable Construction Problems, in the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, Springer-Verlag (Lecture Notes in Computer Science), 283 – 295.
[10] A. Schaerf and L. Di Gaspero (2006). Measurability and Reproducibility in Timetabling Research: State-of-the-Art and Discussion. In *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, 53 – 62.
[11] A. Schaerf (1999). A Survey of Automated Timetabling. Artificial Intelligence Review **13**(2), 87 – 127.
[12] K. Nurmi (1998). Genetic Algorithms for Timetabling and Traveling Salesman Problems. *Ph.D. dissertation*, University of Turku, Finland.
[13] M. Dorigo, V. Maniezzo, and A. Colorni (1996). The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, **26**(1), 29 – 41.
[14] M. Dorigo, G. Di Caro, and L. M. Gambardella (1999). Ant Algorithms for Discrete Optimization. Artificial Life, **5**(2), 137 – 172.
[15] Goldberg, David E (1989), Genetic Algorithms in Search, Optimization and Machine Learning, Kluwer Academic Publishers, Boston, MA.
[16] Vose, Michael D (1999). The Simple Genetic Algorithm: Foundations and Theory, MIT Press, Cambridge, MA.
[17] J.D. Landa Silva, E.K. Burke and S. Petrovic (2004). An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling, MetaHeuristics for Multiobjective Optimisation (edited by X.Gandibleux, M.Sevaux, K.Sorensen and V.T'Kindt), Springer Lecture Notes in Economics and Mathematical Systems Vol. 535, 91 – 129.
[18] Fonseca C.M, Fleming P.J (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation **3**(1), 1 – 16.
[19] Smith A.E, Tate D.M (1993). Genetic Optimisation using a Penalty Function. In *Proceedings of the 5th International Conference on Genetic Algorithms*, 499 – 503.
[20] Richardson J.T, Palmer M.R, Liepins G, Hilliard M (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 191 – 197.
[21] D Goldberg (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.
[22] G Syswerda (1989). Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 2 – 9.
[23] P. Ross and G.H Ballinger (1993) PGA - Parallel Genetic Algorithm Testbed. Department of Articial Intelligence. University of Edinburgh.
[24] W.J. Conover (1999), *Practical Nonparametric Statistics*, John Wiley & Sons, New York.
[25] M. Birattari & al. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 11– 18.
[26] D. Abramson, M. Krishnamoorthy M, H. Dang (1999). Simulated Annealing Cooling Schedules for the School Timetabling Problem. Asia-Pacific Journal of Operational Research **16**, 1 – 22.
[27] K. Nurmi, J.Kyngäs (2007). School Timetabling Problem – Formulation, Instances and Computional Results. URL: www.samk.fi/sttp.
[28] P.J.M van Laarhoven, E.H.L Aarts E.H.L (1987). Simulated annealing: Theory and applications. Kluwer Academic Publishers.

# Publication 2

G. Post, S. Ahmadi, S. Daskalaki, J. H. Kingston, J. Kyngäs, K. Nurmi, D. Ranson and H. Ruizenaar, “An XML Format for Benchmarks in High School Timetabling”, Annals of Operations Research, Springer, USA, 2010. Reprinted with the permission from Springer.

Ann Oper Res
DOI 10.1007/s10479-010-0699-9

# An XML format for benchmarks in High School Timetabling

**Gerhard Post · Samad Ahmadi · Sophia Daskalaki · Jeffrey H. Kingston · Jari Kyngas · Cimmo Nurmi · David Ranson**



**Abstract** The High School Timetabling Problem is amongst the most widely used timetabling problems. This problem has varying structures in different high schools even within the same country or educational system. Due to lack of standard benchmarks and data formats this problem has been studied less than other timetabling problems in the literature. In this paper we describe the High School Timetabling Problem in several countries in order to find a common set of constraints and objectives. Our main goal is to provide exchangeable benchmarks for this problem. To achieve this we propose a standard data format suitable for different countries and educational systems, defined by an XML schema. The schema and datasets are available online.

This research has been supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

G. Post (✉)
Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: g.f.post@math.utwente.nl

G. Post
ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands

S. Ahmadi
School of Computer Science, De Montfort University, The Gateway, Leicester, LE1 9BH, UK

S. Daskalaki
Engineering Sciences Department, University of Patras, 26500 Rio Patras, Greece

J.H. Kingston
School of Information Technologies, The University of Sydney, Sydney, Australia

J. Kyngas · C. Nurmi
Satakunta University of Applied Sciences, Tiedepuisto 3, 28600 Pori, Finland

D. Ranson
Department of Informatics, University of Sussex, Falmer, Brighton, BN1 9QH, UK

Published online: 06 February 2010

**Keywords** Timetabling · High school · Benchmark · Xml · Scheduling

## 1 Introduction

One of the best known timetabling problems is the High School Timetabling Problem; most people have some experience of timetables from their school days, with probably a not always positive opinion on it. There is an implicit belief that all High School Timetabling instances are similar, and that a computer program can always 'solve' it.

In reality, the research in this area is still very active and we are nowhere near solving all the instances of the High School Timetabling Problem to optimality. Moreover, continuous reforms in educational systems throughout the world generate new problems to tackle. On the literature side there are papers on general timetabling problems such as de Werra (1985), Cooper and Kingston (1993), Schaerf (1999), Carter and Laporte (1998), de Werra (1999), Burke and Petrovic (2002). Some of their concepts and/or methods can be used in real-life timetabling problems. On the other hand, there are also case studies of high schools in particular countries, some of which are mentioned in Sect. 2.

It is surprising that no standard format for exchanging datasets in the field of High School Timetabling has emerged until now. An accepted format would greatly facilitate the progress in the field. We try to fill in this gap by providing an XML format for existing and new High School Timetabling datasets. As a starting point, the authors of this paper have agreed to contribute at least one of their datasets in this XML format. These datasets will be made available on the internet at the web site devoted to this project (Post 2008).

XML has been chosen because it is extensively used nowadays in web services, or as a means of exchanging data between applications. The power of XML lies in the fact that the data is structured, and therefore it can easily be adjusted to changes, unlike plain text files. Programming languages have software libraries which make it easy to handle XML. Mathematically speaking, XML is a rooted tree, with information attached to the nodes. This information can contain cross references. Examples of XML are presented in Sect. 4; see also http://www.w3schools.com/xml/; http://en.wikipedia.org/wiki/XML.

The problem of exchanging timetabling data was discussed at the first International Conference on the Theory and Practice of Automated Timetabling (Cumming and Paechter 2005), where it became clear that the principal difficulty was the precise expression of the many different kinds of constraints. There may be ten or more kinds in any one instance, and they vary widely between institutions.

This complexity of specification has been addressed in several ways. Some papers have tried to generalize and unify the constraints (Chand 2004; Kitagawa and Ikeda 1988). Others have adapted existing technologies in which constraints may be expressed, such as XML and the semantic web (Custers et al. 2005; De Causmaecker et al. 2000; De Causmaecker et al. 2002; Özcan 2005), or object-oriented modeling and frameworks (Gröbner et al. 2003; Ranson and Ahmadi 2006). Others have expressed constraints as logic expressions within specifically designed specification languages (Burke et al. 1998; Kingston 2001; Monteiro da Mata et al. 1997). There has been at least one attempt to simply enumerate every possible constraint (Reis and Oliveira 2001). While much of this work is more general than our approach (for example, the use of MathML functions in Özcan (2005) allows essentially arbitrary constraints to be expressed in XML), none of it has led to significant data exchange.

Another approach is to restrict the problem domain to one particular kind of timetabling, then use judicious simplification to further reduce the specification burden while maintaining the essence of the problem. The Carter data sets for examination timetabling (Carter

et al. 1996) omit many details, notably all data related to rooms, and similar simplifications appear in the Traveling Tournament Problem (Easton et al. 2001) and the International Timetabling Competition (Paechter 2003). These are some of the most successful examples of timetabling data exchange so far. However, judicious simplification has been criticized for contributing to the gap between research and practice (Burke et al. 2006), at least in examination timetabling; and the data transfer has almost always been in one direction only.

## 2 The timetabling problem in different countries

Throughout the world, timetables are constructed for many different types of high schools. In the following subsections we describe the situation in the schools of Australia, England, Finland, Greece and the Netherlands. However, before getting into the details of their problems, we provide an overview of the problem and their principal differences. Not only the problems, but also the terminology varies among countries. For more details we refer to Sect. 3.1.

### 2.1 General aspects of the problem

The timetabling process in a given school is influenced by the school organization, and specifically the handling of the three groups of resources: students, teachers, and rooms. In the heart of the high school timetabling problem lie the students, who are usually organized in *base groups*, called 'Student divisions' or 'Student groups'. These base groups form a partitioning of the students, i.e. each student belongs to exactly one base group. We may consider two types of timetabling problems:

1. Problems for which the base groups never split up over different lessons, i.e. if two students of a base group have a lesson at the same time, they are in the same lesson.
2. Problems for which the students of a base group may attend different lessons for some times.

In practice not many schools will fall completely into the first category. Some schools, for example, will split two base groups of boys and girls for gym lessons. Similarly, other schools may split base groups for reasons such as religion, different levels of ability in a subject, etc.

Another important difference between countries is the compactness of the students' schedules. Compact schedules are schedules with no idle times. An *idle time* for a student is a time (slot) when the student has no lesson, between others on the same day where the student does have a lesson. In most cases compact schedules are required for the students. In some countries this is automatic, as a student has as many lessons as there are times. However, when a school has a large variety of elective subjects it is almost impossible to achieve compact schedules. Then the objective would be to minimize the number of idle times.

The teachers form the second most important group of resources affected by the timetable. They are normally preassigned to lessons by the school management. The assignments take into consideration all limiting constraints like the number of staff, their expertise and other aspects which may be difficult to model for an automated system. Examples of such difficult constraints would be assigning new teachers to lessons that take place in parallel, balancing teachers assigned to a base group of students (male/female, experienced/less experienced), or avoiding parent-child combinations in lessons.

Idle times in teachers' schedules also form a point of differentiation among different countries. If teachers are preassigned to lessons, it is extremely hard (except only in simple

cases) to find schedules with no idle times for both students and teachers. When idle times are not allowed for students, typically one can ask only to minimize the number of idles for teachers.

Another important issue is the spread of lessons for the teachers. Sometimes, as with students, this is almost automatic for full-time teachers. For part-time teachers, however, particular kinds of spreading may be important. For example, a teacher with an assignment of only 5 times per week will probably be unhappy with a schedule that spreads them over all days of the week.

The rooms of a school form the third major group of resources. Although most lessons will take place in a room, this does not necessarily imply that room assignment is an issue. In many schools, a room is preassigned to each base group of students, or to each teacher. It is also possible that the number of rooms is so large that assigning rooms is never a problem. While in some schools this may be the case for general-purpose rooms, specialized rooms may be scarce and in need of careful scheduling. Examples are specialized rooms for Physical Education, Music, Arts, or Labs for physics, biology and chemistry.

We now proceed with the descriptions of the problems that the different countries face in their schools.

### 2.2 Australia

Australian high schools have short school days, with correspondingly high teacher utilisations, and consequently no demand for compactness in their timetables. Most teachers (those with no special duties) teach for 75% of the times of the cycle, and the total workload of all teachers is almost 100% of the available workload permitted by the teachers' workload limits. These conditions commonly prevent teachers from being preassigned, except to a few key subjects (e.g. the senior ones), and force some courses to be shared between two teachers. Minimizing the number of these *split assignments* is a key goal. There is a very high utilization of specialized rooms, notably Science and Computer laboratories.

The above describes the situation in government funded high schools. Privately funded high schools are often better resourced, with lower teacher workloads and more laboratories. However, they typically have lower student numbers and a commitment to high student choice, leading to peculiarities such as *composite classes*, in which students of different ages study a common subject together under a single teacher, creating other difficulties.

Another Australian problem concerns the number and distribution of lessons for each subject. In the senior years this is very simple: for example, a senior student might attend six subjects, each occupying six times per week, spread across the five days, including a double period (i.e. two adjacent times not separated by a break) on one of the days. Student choice is catered for by means of *electives*: lists of subjects running simultaneously from which each student chooses one. In the junior years a desire to permit students to sample a wide range of subjects leads to a chaotic curriculum from the timetabling point of view. A few subjects (English, Mathematics, and Science) follow the senior pattern, but the remainder of the cycle is occupied by many small subjects with more or less random numbers of times.

For previous research and additional details we refer to Abramson (1991), Kingston (2005).

### 2.3 England

In English Secondary Schools students study a mixture of compulsory and optional subjects. For the 11- to 14-year-olds all subjects are compulsory and students study 10 main subjects

plus a modern foreign language, citizenship education, careers education and sex education. There are fewer compulsory subjects for 14- to 16-year-olds. In addition, schools must offer students some work-related learning during this phase.

Depending on the policy of the school, within each year group the students may be divided into base groups with a designated teacher who has organizational and pastoral care responsibilities.

Another common grouping structure used in British schools is a Band, which is a collection of several tutorial groups. Normally, all the base groups in a year are allocated to fewer bands. The mixing of base groups is done within a band. Subjects are grouped into *blocks*, which are equivalent to the Australian electives: a block consists of one or more subjects for different groups which will be scheduled at the same time. Blocks are constructed manually before the scheduling process starts.

Except in the Sixth Form, British school students receive compact schedules automatically, since they have as many lessons as there are times.

In general there is a large variety of different length of lessons, one or two weekly timetabling cycles, length of school day and hence different number of lessons in different schools. This requires any timetabling system for English schools to be highly parameterised to accommodate different patterns of delivery in different schools.

For previous research and additional details we refer to Wright (1996).

### 2.4 Finland

A basic goal in Finnish school timetabling is to construct a one-week schedule, which is repeated the whole season. A timetable consists of lessons of subjects, where a lesson is a predefined combination of a student group, a teacher, a room type and the duration of the lesson. Every student belongs to one base group and most of the lessons are scheduled based on this group. In addition, the student can belong to a number of optional groups, which are built according to the enrolment of students in optional courses. Every lesson is assigned to one and only one group, either base or optional. Base and optional groups define a compatibility matrix, which determines which groups can or cannot have lessons at the same time.

In all Finnish school levels teachers are preassigned to lessons. Rooms are preassigned to most lessons, based on the preferences of the teachers.

In a typical curriculum a student attends ten subjects, each taught for two to six hours each week. Lessons of a subject can take one, two, or (in exceptional cases) three hours. Hence, most subjects are taught either within two, three or four days each week.

Compact schedules for the students (student groups) are necessary and idle times are either strictly prohibited or highly inappropriate. Conversely, idle times for the teachers are allowed, but still are very unappreciated. Several teachers also prefer not to teach more than a given number of hours in a day.

Preassignment of teachers and rooms, a somewhat complicated structure of student groups and the demand for compact scheduling make Finnish School Timetabling Problem a challenge for both a manual solver and a computer software. A further description of the Finnish problem can be found in Nurmi and Kyngas (2007).

### 2.5 Greece

Secondary education in Greece is divided in two portions: *lyceum* (grades 7 to 9) and *gymnasium* (grades 10 to 12). The last two years in the lyceum are considered as preparatory for

the higher level of education and the students are asked to choose one of three directions. Similarly, in vocational lyceums the students are asked to choose one of several specializations.

Aspects common to the two school types are the five days of the week, the number of times per day (six to seven), the preassignment of lessons to teachers, who may be full-time or part-time, and the requirement for compact student schedules.

In the gymnasium all students of a given base group attend the same lessons during most of the time in their dedicated classroom. However, for a small portion of the weekly timetable some base groups split into sub-groups or reshuffle with some other base group and split again for attending certain 'special' subjects. For these subjects it is required that two teachers are teaching simultaneously to two different sub-groups of students, or alternatively that two teachers are collaboratively teaching to the same group of students.

In the lyceum the general practice is to keep students in their regular base groups for a portion of the day to attend the lessons that are common to all directions or specializations, then reshuffle and split again based on specializations, directions or elective courses until the end of the day.

Important objectives for the timetabling in the gymnasium are to schedule core courses evenly during the week, preferably during the prime times, and to maintain a minimal number of idle times for the teachers. For the lyceums, however, the objectives change because of all the scheduling considerations mentioned previously. Balanced distribution of the core courses during the week is still important; however, the other two objectives cannot be achieved.

For previous research and additional details we refer to Birbas et al. (1997), Valouxis and Housos (2003).

### 2.6 The Netherlands

In Dutch secondary schools different levels of education are offered within one school. Teachers are shared among these levels. In the lower years the students in a base group follow the same courses, while in the last years, the students choose a specialization, with some compulsory and some optional subjects. The timetable is usually weekly, and valid for a period of 6 weeks, a trimester, semester, or the whole year. In the lower levels compact schedules often are compulsory. In the higher levels compact schedules are impossible to realize for all students, and in the opinion of the school administration not necessary. However, avoiding idle times as much as possible for students as well as teachers is still important.

Usually the teachers are preassigned to the lessons by the school administration, to ensure a good spreading of the teachers. The majority of the teachers work part-time. According to collective labour agreements these teachers are entitled to one or more days off.

The rooms are generally not preassigned to lessons. A lesson requires a room of a certain type, and lessons should be assigned such that enough rooms of each type are available. High utilization occurs for specialized rooms, like the gyms.

Some schools have special arrangements for students good at sports, dance or ballet, allowing them to skip lessons at the beginning or the end of some days. Some schools experiment with large groups of students (50 to 60 students) in a learning studio. Effectively there are two teachers visiting them in a period, of which one acts as supervisor, and the other one teaches the subject. All these special constructions make the task of timetabling even more challenging.

For previous research and additional details we refer to de Gans (1981), Willemen (2002), de Haan et al. (2007).

## 3 Modeling the problem

The basic game we play in modeling is between abstraction and concreteness. In an abstract sense, we could only view events, and express all constraints in the events. For accessibility we decided to define Times and Resources as well. Groups of times, resources and events may be defined: TimeGroups, ResourceGroups, and EventGroups. This makes the formulation of constraints much more readable and compact. For example, an instance will usually contain the 'AvoidClashConstraint' as a hard constraint for all resources, i.e. for the resource group consisting of all resources.

### 3.1 Times and resources

We divide the time period into *times*. In many constraints it is essential to know the relation between groups of times, for example for idle times per day or week, or the number of lesson per day. Days and weeks are time groups, which are treated on the same basis as other time groups in the constraints. They can be introduced to make a dataset more understandable.

A *resource* is an entity which attends events. The most common resources in the timetabling problem are the students, teachers, and rooms. Other kinds of resources, such as equipment (video projectors, etc.) are possible but uncommon. Resources may be classified into subgroups, for example rooms of certain type or student groups of a certain form. It is a basic hard constraint of most timetabling problems that no resource may attend two events scheduled for the same time. A violation of this constraint is called a *clash*.

We attach no additional properties to resources; all additional requirements will be modeled with the help of the constraints, where the involved resources will be selected.

As stated before, we distinguish three main kinds of resources:

- *Students*. Usually a student group is preassigned to each events. Constraints that can be important for students are controlling the number of idle times, the number of lessons per day.
- *Teachers*. As mentioned earlier, in some countries teachers are preassigned to events, while in others they are not. In the latter case the choice of assignment will be restricted by teachers' qualifications and workload limits. Important issues for the teachers are the total number of idle times, the number of assigned times per day, the number of days with events, and unavailable days or times.
- *Rooms*. Most events take place in a room. If rooms abound, we can disregard them, if not they can be a bottleneck, especially for specialized rooms.

### 3.2 Events

Events are the basic scheduling objects. An event can represent either a single lesson, or a set of ('linked') lessons, that have to be taught at the same time.

We use the term *course* for a collection of events taught to a group of students in a subject. Events that refer to the same course, are called the *lessons* of the course. In other papers such a set of lessons might be called a class. The most common constraint related to courses concerns the distribution of the lessons of a given course in the week (or in different weeks). In situations where the teachers are not preassigned to courses, an important constraint is that all lessons of a given course should be assigned to the same teacher. In our format a course is a specialized event group, like the day and week are specialized time groups.

Other types of events, such as staff meetings, are also permitted. An event has the following properties.

- The *duration*, which is the number of times that have to be assigned to the event. These times must be consecutive.
- The *course* related to an event.
- The *time* that the event is scheduled to. If the duration is larger than one, the event will occupy some following times as well. Constructing a timetable involves assigning a time to all events.
- The *workload* that the event will contribute to a teacher's total workload. Often this is the same as the duration. The workload is only needed when teachers are not preassigned, and consequently total workloads must be calculated.
- The *resource groups* are preassigned.
- Any number of *resources*, either preassigned, or to be assigned. These each have a 'role', used to identify them. A role might have value 'room', 'teacher', 'senior teacher', and so on. A resource may be constrained to be from a particular resource group, for example a Science laboratory or an English teacher.

## 3.3 Constraints

Our XML format currently defines 13 constraint types. We expect that this number will grow in the future. A constraint can be hard ('Required') or soft. In both cases the constraint generates a cost, which either contributes to the infeasibility value, or to the objective value. The constraints describe all aspects of the scheduling, even the (elementary) assignment part. The advantage of this approach is that we can distinguish between levels of infeasibility: if not all events can be scheduled, we can give preferences among them. In Sect. 4.2 we give an example how a constraint contributes to the objective function.

The cost of a schedule consists of three parts: the cost of a resource, cost of an event group, cost of an event.

We group the constraints into three groups: constraints describing the basic scheduling problem, other constraints for events, and constraints for resources.

### *3.3.1 Basic scheduling constraints*

- *AssignTimeConstraint* (Cost per event). Assign a time to each of the selected events.
- *AssignResourceConstraint* (Cost per event). Assign a resource to the role in each of the selected events.

Both constraints have a variant expressing the preference for the time, respectively the resource: *PreferTimesConstraint* and *PreferResourcesConstraint*.

### *3.3.2 Event constraints*

- *LinkEventsConstraint* (Cost per event group). Schedule the selected event groups at the same (starting) time.
- *SpreadEventsConstraint*. (Cost per event group). Schedule the events of the selected event groups to the selected time groups between a minimum and a maximum number of times.
- *AvoidSplitAssignmentsConstraint*. (Cost per event group). For each selected event group, schedule the selected role of each event of this group to the same resource.

### *3.3.3 Resource constraints*

Resource constraints describe the quality of the timetable of a single resource; the corresponding cost is attributed to the resource.

- *AvoidClashesConstraint*. Schedule the selected resources without clashes. This is one of the basic (hard) constraints.
- *AvoidUnavailableTimesConstraint*. Avoid that the selected resources are busy in the selected times.
- *LimitWorkloadConstraint*. Schedule workload to the selected resources between a minimum and a maximum.
- *LimitIdleTimesConstraint*. The number of idle times in the selected time groups should lie between a minimum and a maximum for each of the selected resources. Typically the time groups are a day or all days.
- *LimitBusyTimesConstraint*. The number of occupied times for the selected resources should lie between a minimum and a maximum for each of the selected time groups. Typically the time groups are the days.
- *ClusterBusyTimesConstraint*. The number of time *groups* with an assigned time should lie between a minimum and a maximum for the selected resources. Typically the time groups are days; for example a teacher requiring at most 3 days with lessons.

## 4 The XML format for benchmarks

XML (http://www.w3schools.com/xml/; http://en.wikipedia.org/wiki/XML) is a mark-up language similar to LaTeX and HTML. It is used extensively for exchanging data between applications. Most object-oriented languages, like Java, C++, C#, and Delphi, have extensive libraries to work with XML. For this reason XML seems to be very useful for benchmarking. For a comparable project in nurse rostering, see (Curtois 2006).

The XML format is defined by an XML schema, which is an XSD file (itself written in XML). The schema defines what elements must be and may be present in the XML file. The format is straightforward and to present it all would be tedious. Instead, we give three examples which should make the general flavour clear: the top-level format, one constraint, and the format of solutions. For a complete description, see (Post 2008), where a list with terms used can be found, as well as a detailed description of all constraints. As an introductory example the 4x4-Sudoku formulated as timetabling problem (without preassignments, and hence multiple solutions) is available at this website.

### 4.1 The top-level format

At the top level the XML file looks like this:

```
<HighSchoolTimetableArchive>
  <Instances>
    <Instance Id="Sudoku4x4">
      <Times>
         ...
      </Times>
      <Resources>
         ...
```

```
      </Resources>
      <Events>
         ...
      </Events>
      <Constraints>
         ...
      </Constraints>
    </Instance>
  </Instances>
  <SolutionGroups>
      ...
  </SolutionGroups>
</HighSchoolTimetableArchive>
```

The file contains one or more instances and any number of grouped solutions. The instance contains 'Times', 'Resources', 'Events', and 'Constraints' sections, as in the data model (Sect. 3).

### 4.2 Constraints

We give one example of a constraint, penalizing idle times. An idle time is relevant to a resource and to a group of times. If there is a time without event scheduled to the resource, but with scheduled events before and after in the same time group, we call it an idle time for the resource within this time group. Usually the time group will consist of all times of a day. In the constraints section of the XML document, we could have, among other constraints, the following:

```
<LimitIdleTimesConstraint Id="StudentIdleTimes">
  <Name>At most three idle times for students per week</Name>
  <Required>false</Required>
  <Weight>10</Weight>
  <CostFunction>SquareSum</CostFunction>
  <AppliesTo>
    <ResourceGroups>
      <ResourceGroup Reference="Students"/>
    <ResourceGroups>
  </AppliesTo>
  <TimeGroups>
    <TimeGroup Reference="Monday"/>
    <TimeGroup Reference="Tuesday"/>
    <TimeGroup Reference="Wednesday"/>
    <TimeGroup Reference="Thursday"/>
    <TimeGroup Reference="Friday"/>
  </TimeGroups>
  <Minimum>0</Minimum>
  <Maximum>3</Maximum>
</IdleTimesConstraint>
```

The attribute 'Id' and the first four fields are present in all constraints. They define a unique reference (Id) and a display name (Name), and give information on how to interpret violations this constraint. Required = "false" means that this constraint is a soft constraint, and

hence any cost will be added to the objective value. The Weight and CostFunction explain how to calculate the cost for the deviation of the violation. In this constraint a violation is that one of the students in the resource group 'Students' has more than 3 idle times in the week. The deviation is the surplus (per week), so the number of idle times minus 3. If a student has 1, 0, 2, 3, 1 idle times, on the five days of the week respectively, the total number of idle times is 7, and the deviation is 4. Since we use a quadratic term for this cost and multiply by the weight, we obtain the cost 160. Hence the (constraint, student) combination adds an amount of 160 to the objective function.

This example sums the idle times per day. If we only are interested in the idle times on Monday, we can omit the time groups for the other days.

### 4.3 Solutions

The solution in XML is presented as:

```
<Solution Id="Sudoku4x4">
  <Events>
    <Event Reference="Event1">
      <Time Reference="Day_1"/>
      <Resources>
        <Resource Reference="R1">
          <Role>Room</Role>
        </Resource>
      </Resources>
    </Event>
    <Event Reference="Event2">
        ...
    </Event>
      ...
  </Events>
</Solution>
```

The main thing a solution should do is tell the result of the two scheduling constraints. For this each Event section is filled with the time and resources of the corresponding assignment constraints. Once these aspects are known, all other information can be calculated, and a summary can be added. For debugging purposes, we introduce the possibility to give a full report on the violations. This report gives for each resource, event group, and event the constraints that are violated, and for each of the violated constraints the deviation, cost, and a text explaining the violation. At Kingston (2009) an evaluator is available which validates the XML in the dataset, and returns the report on the submitted solutions.

## 5 Conclusion

The XML format advocated here has been designed in order to better model the complete high school timetabling problem and facilitate data exchange between high school timetabling researchers. We used as input the situation in five different countries, and we think that the presented format can deal with these countries. This belief is based on several datasets we have in our 'old' formats. It is hoped that researchers and practitioners will consider adopting this format in their work and that this will lead to further discussions and

improvements to the model. As previously stated we fully expect the number of included constraints to expand. We actively will encourage others to contribute datasets in this format, and use the evaluator (Kingston 2009) to validate their datasets. This should not only lead to improvements of the model but also to the development of better and/or more general algorithms.



## References

Abramson, D. (1991). Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, *37*, 98–113.

Birbas, T., Daskalaki, S., & Housos, E. (1997). Timetabling for Greek high schools. *Journal of the Operational Research Society*, *48*, 1191–1200.

Burke, E. K., & Petrovic, S. (2002). Recent research directions in automated timetabling. *European Journal of Operational Research*, *140*, 266–280.

Burke, E. K., Kingston, J. H., & Pepper, P. A. (1998). A standard data format for timetabling instances. In E. Burke & M. Carter (Eds.), *Lecture notes in computer science: Vol. 1408. Practice and theory of automated timetabling II* (pp. 213–222). Berlin: Springer.

Burke, E. K., McCollum, B., McMullan, P., & Qu, R. (2006). Examination timetabling: a new formulation. In: *Proceedings of the sixth international conference of the practice and theory of automated timetabling (PATAT 2006)*, Brno, 2006 (pp. 373–375).

Carter, M., Laporte, G., & Lee, S. T. (1996). Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society*, *47*, 373–383.

Carter, M. W., & Laporte, G. (1998). Recent developments in practical course timetabling. In E. Burke & M. Carter (Eds.), *Lecture notes in computer science: Vol. 1408. Practice and theory of automated timetabling II* (pp. 3–19). Berlin: Springer.

Chand, A. (2004). A constraint based generic model for representing complete university timetabling data. In: *Proceedings of the fifth international conference on the practice and theory of automated timetabling (PATAT 2004)*, Pittsburgh, 2004 (pp. 125–148).

Cooper, T. B., & Kingston, J. (1993). The solution of real instances of the timetabling problem. *The Computer Journal*, *36*, 645–653.

Cumming, A., & Paechter, B. (2005). *Standard formats for timetabling data*. Unpublished discussion session at the first international conference on the practice and theory of automated timetabling, Edinburgh, 2005.

Curtois, T. (2006). Nurse rostering web site. http://www.cs.nott.ac.uk/~tec/NRP/.

Custers, N., De Causmaecker, P., Demeester, P., & Vanden Berghe, G. (2005). Semantic components for timetabling. In E. Burke & M. Trick (Eds.), *Lecture notes in computer science: Vol. 3616. Practice and Theory of Automated Timetabling V'* (pp. 17–33). Berlin: Springer.

De Causmaecker, P., Demeester, P., De Pauw-Waterschoot, P., & Vanden Berghe, G. (2000). Ontology for timetabling. In: *Proceedings of the third international conference on the practice and theory of automated timetabling (PATAT 2000)*, Konstanz, 2000 (pp. 481–482).

De Causmaecker, P., Demeester, P., Lu, Y., & Vanden Berghe, G. (2002). Using web standards for timetabling. In: *Proceedings of the fourth international conference on the practice and theory of automated timetabling (PATAT 2002)*, Gent, 2002 (pp. 238–257).

de Gans, O. B. (1981). A computer timetabling system for secondary schools in the Netherlands. *European Journal of Operational Research*, *7*, 175–182.

de Haan, P., Landman, R., Post, G., & Ruizenaar, H. (2007). A case study for timetabling in a Dutch secondary school. In E. Burke & H. Rudová (Eds.), *Lecture notes in computer science: Vol. 3867. Practice and theory of automated timetabling VI* (pp. 267–279). Berlin: Springer.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research*, *19*, 151–162.

de Werra, D. (1999). On a multiconstrained model for chromatic scheduling. *Discrete Applied Mathematics*, *94*, 171–180.

Easton, K., Nemhauser, G. L., & Trick, M. A. (2001). The travelling tournament problem: description and benchmarks. In *Lecture notes in computer science: Vol. 2239. Principles and practice of constraint programming (CP 2001)* (pp. 580–585). Berlin: Springer.

Gröbner, M., Wilke, P., & Büttcher, S. (2003). A standard framework for timetabling problems. In E. Burke & P. De Causmaecker (Eds.), *Lecture notes in computer science: Vol. 2740. Practice and theory of automated timetabling IV* (pp. 24–38). Berlin: Springer.

Kingston, J. H. (2001). Modelling timetabling problems with STTL. In E. K. Burke & W. Erben (Eds.), *Lecture notes in computer science: Vol. 2079. Practice and theory of automated rimetabling III* (pp. 309–321). Berlin: Springer.

Kingston, J. H. (2005). A tiling algorithm for high school timetabling. In E. Burke & M. Trick (Eds.), *Lecture notes in computer science: Vol. 3616. Practice and theory of automated timetabling V* (pp. 208–225). Berlin: Springer.

Kingston, J. H. (2009). The HSEval High School Timetable Evaluator. http://www.it.usyd.edu.au/~jeff/hseval.cgi.

Kitagawa, F., & Ikeda, H. (1988). An existential problem of a weight-controlled subset and its application to school timetable construction'. *Discrete Mathematics*, *72*, 195–211.

Monteiro da Mata, J., Luiz de Senna, A., & Augusto de Andrade, M. (1997). Towards a language for the specification of timetabling problems. In *Proceedings of the second international conference on the practice and theory of automated timetabling (PATAT'97)*, Toronto, 1997 (pp. 330–333).

Nurmi, K., & Kyngas, J. (2007). A framework for school timetabling problem. In: *Proceedings of the 3rd multidisciplinary international scheduling conference: theory and applications*, Paris, 2007 (pp. 386–393).

Özcan, E. (2003). Towards an XML-based standard for timetabling problems: TTML, multidisciplinary Scheduling: theory and applications. In *First international conference, MISTA '03*, Nottingham, Selected Papers (2005) (pp. 163–185).

Paechter, B. (2003). International timetabling competition. http://www.idsia.ch/Files/ttcomp2002/.

Post, G. (2008). High school timetabling web site. http://wwwhome.math.utwente.nl/~postgf/BenchmarkSchoolTimetabling/.

Ranson, D., & Ahmadi, S. (2006). An extensible modelling framework for the examination timetabling problem. In E. Burke & H. Rudová (Eds.) *Lecture notes in computer science: Vol. 3867. Practice and theory of automated timetabling VI* (pp. 383–393). Berlin: Springer.

Reis, L. P., & Oliveira, E. (2001). A language for specifying complete timetabling problems. In E. K. Burke & W. Erben (Eds.) *Lecture notes in computer science: Vol. 2079. Practice and theory of automated timetabling III* (pp. 322–341). Berlin: Springer.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, *13*(2), 87–127.

Valouxis, C., & Housos, E. (2003). Constraint programming approach for school timetabling. *Computers & Operations Research*, *30*, 1555–1572.

Willemen, R. J. (2002). *School timetable construction; algorithms and complexity*. PhD thesis, Technical University Eindhoven, The Netherlands.

Wright, M. (1996). School timetabling using heuristic search. *Journal of Operational Research Society*, *47*, 347–357.

# Publication 3

J. Kyngäs and K. Nurmi, “Scheduling the Finnish Major Ice Hockey League”, in Proc of the IEEE Symposium on Computational Intelligence in Scheduling, Nashville, USA, 2009, pp. 84-89.

# Scheduling the Finnish Major Ice Hockey League

Jari Kyngäs, Kimmo Nurmi

*Abstract*—**Generating a schedule for a professional sports league is an extremely demanding task. Good schedules have many benefits for the league, such as higher incomes, lower costs and more interesting and fairer seasons. This paper presents a successful way to schedule the Finnish major ice hockey league. The method is a combination of local search heuristics and evolutionary methods. The generated schedule is currently in use for the 2008-2009 season.**

## I. Introduction

Many new timetabling problems have been introduced in recent years. Most of the timetabling research used to concentrate on university and school timetabling, but rostering and sports scheduling have been quite extensively studied recently. Excellent overviews of sports scheduling can be found in [1] and [2]. The most important theoretical results can be found in [3]-[11]. An extensive summary of the theoretical results can be found in [2].

Successful algorithms for sports scheduling include tabu search [12]-[14], simulated annealing [15]-[17], graph coloring and branch and bound [18], constrained programming [19]-[22], integer linear programming [23]-[25] and hybrid integer/constrained programming [8],[26].

Even if quite efficient algorithms have recently been designed for sports scheduling problems, to the best of our knowledge, there are only a few papers where academic researchers have announced that they have closed a contract with a sports league owner: the major baseball league in the USA [27], the major soccer league in Austria [18], 1st division soccer in Chile [24], the major basketball league in New Zealand [17], the major soccer league in Belgium [22], the major soccer league in Denmark [28] and the major volleyball league in Argentina [29]. This paper presents a new case: the major ice hockey league in Finland.

There are basically three reasons for the current interest in sports scheduling. First, sports leagues are organized more professionally than before and they have realized that a good schedule is vital for their league's success. Second, sports leagues have faced so many new requirements and requests that they can no longer handle the schedules manually. Finally, new algorithms have been developed to tackle intractable problems, and, at the same time, computer power has increased to such a level that researchers are able to solve real-world problems. For these reasons, the league authorities contact universities and researchers to make their league more successful and more profitable.

Professional sports leagues are big businesses. An increase in revenue comes from many factors: an increased number of spectators both in stadiums and in TV networks, reduced traveling costs for teams, a more interesting tournament for the media and sports fans, and a fairer tournament for the teams. Furthermore, TV networks buy the rights to broadcast the games and in return want the most attractive games to be scheduled at certain times.

The focus of this paper is to solve a highly constrained sports scheduling problem. In Section II we define a sports scheduling problem and introduce the necessary terminology. Sections III and IV detail the requirements and the requests of the problem and our solution method. Even if there is a clear tendency to use integer and constrained programming models, our algorithm uses a mixture of evolutionary and local search methods. In Sections V and VI we present the Finnish major ice hockey league problem and the difficulty of the scheduling process. Finally, in Section VII we report our computational results. It will be seen that our approach produces excellent results compared to the manual schedule used in the previous season. One of the schedules we generated was accepted by the league authorities and the cooperation will continue in the years to come.

## II. Sports scheduling

In a sports competition, *n teams* play against each other over a period of time according to a given timetable. The teams belong to a *league*. In general, *n* is assumed to be an even number. A dummy team is added if a league has an odd number of teams. The league organizes *games* between the teams. Each game consists of an ordered pair of teams *(i, j)*. The first team, *i,* plays *at home* - that is, uses its own *venue* (stadium) for a game - and the second team, *j,* plays *away*. Games are scheduled in *rounds*. Each round is played on a given *day*. A *schedule* consists of games assigned to rounds. A schedule is *compact* if each team plays exactly one game in each round; otherwise it is *relaxed*. If a team has no game in a round, it is said to have a *bye*. If a team plays two home or two away games in two consecutive rounds, it is said to have a *break*.

Most professional leagues use a round robin tournament, where every team plays against every other team a fixed number of times. Most sports leagues play a double round robin tournament (*2RR*), where the teams meet twice, but quadruple round robin tournaments (*4RR*) are also quite common. In a 2RR, every team plays against every other team once at home and once away. The number of rounds in a compact single round robin tournament (*1RR*) is $n - 1$ and the number of games is $n\,(n-1)/2$.

978-1-4244-2757-4/09/$25.00 ©2009 IEEE

Scheduling a 1RR without any requirements is an easy task. In the 1980s, a constructive method was created by Schreuder [3] for generating a 1RR with a minimum number of breaks. Later [4] he showed that if $n$ is even, it is always possible to construct a schedule with $n - 2$ breaks, and that this number is the minimum. For a mirrored 2RR, it is always possible to construct a schedule with exactly $3n - 6$ breaks [5].

Schreuder also presented an efficient algorithm to compute a minimum break schedule. However, the algorithm can only be used directly when no additional requirements exist. Professional sports leagues have many requirements, constraints and requests that exclude the use of the algorithm.

The problem of finding a schedule with the minimum number of breaks and, at the same time, take additional requirements and requests into account is known as the *constrained minimum break problem,* see, e.g., [28].

The sports scheduling problem presented in this paper is such a problem. There are three main reasons why we have to minimize the number of breaks: first, the fans do not like long periods without home games; second, consecutive home games reduce gate receipts; and third, long sequences of home or away games might influence the team's current position in the tournament.

Table I shows an example of a compact 2RR with $n = 6$. The second part of the schedule is actually mirrored from the first part. The schedule has no breaks for teams 1 and 5, three breaks for teams 2 and 3, three-in-a-row home games for team 6 and five-in-a-row away games for team 4.

TABLE I
A MIRRORED DOUBLE ROUND ROBIN TOURNAMENT WITH SIX TEAMS

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 1 – 6 | 3 – 1 | 1 – 5 | 2 – 1 | 1 – 4 |
| 2 – 5 | 6 – 2 | 2 – 4 | 5 – 3 | 3 – 2 |
| 4 – 3 | 5 – 4 | 3 – 6 | 6 – 4 | 6 – 5 |
| R6 | R7 | R8 | R9 | R10 |
| 6 – 1 | 1 – 3 | 5 – 1 | 1 – 2 | 4 – 1 |
| 5 – 2 | 2 – 6 | 4 – 2 | 3 – 5 | 2 – 3 |
| 3 – 4 | 4 – 5 | 6 – 3 | 4 – 6 | 5 – 6 |

When the teams do not return home after each away game but instead travel from one away game to the next, we are no longer minimizing the number of breaks. Now we have to design a schedule that minimizes the distances the teams must travel. As an academic problem it is known as the *Traveling Tournament Problem (TTP)* [8]. This paper was one of the reasons for the increased interest in sports scheduling.

## III. REQUIREMENTS AND REQUESTS

In this section we will give an outline of typical hard and soft constraints of a sports scheduling problem. The problem is to construct a round robin tournament that minimizes the number of breaks and the number of soft constraint violations. The games should be scheduled in rounds in such a way that the solution is feasible and mostly acceptable to both the league authorities and the teams.

A league can use some of the following requirements (hard constraints) for their feasible schedule:

- *H1*. Every team plays exactly once in every round (if a compact schedule is required).
- *H2*. A team cannot play at home on a certain day (e.g. a venue is unavailable).
- *H3*. A team cannot play away on a certain day (e.g. the team has an anniversary).
- *H4*. A team cannot play at all on a certain day (e.g. the team has a game in another league).
- *H5*. Two teams cannot play at home on the same day (e.g. they share a venue).
- *H6*. A team cannot play at home on two consecutive calendar days.
- *H7*. A game must be pre-assigned to a certain round.
- *H8*. A break cannot occur in the second round.
- *H9*. A maximum of $m$ games can be assigned to round $r$ (if a relaxed schedule is used, i.e. there are more than $n - 1$ rounds available for scheduling).

Furthermore, there are a wide variety of requests presented by the league and the teams. They prefer to optimize many goals at the same time. A league uses some of the following requests (soft constraints) for their acceptable schedule:

- *S1*. Every round should be as compact as possible (if a relaxed schedule is used).
- *S2*. A team cannot have more than two consecutive home games.
- *S3*. A team cannot have more than two consecutive away games.
- *S4*. A team wishes to play two or more consecutive away games (away tours).
- *S5*. There must be at least $k$ rounds before two teams meet again.
- *S6*. Two teams play against each other at home and away in turn (in a 3RR or more).
- *S7*. A team wishes to play most of its home games on certain weekdays.
- *S8*. Two teams do not want to play at home on the same day (e.g. they are located in the same region).
- *S9*. A team does not want to play at home on a certain day (e.g. a team in another league in the same region already has a home game scheduled for that day).
- *S10*. A team should not play two consecutive games against opponents in the same strength group.
- *S11*. A game cannot be played before round r.
- *S12*. The difference between the number of home games and the number of away games for each team should be as small as possible after each round.
- *S13*. The difference in the number of games played between the teams should be as small as possible after each round (while playing a relaxed schedule).

Note that if a compact schedule is not required, the hard constraint *H1* turns into the soft constraint *S1*. In that case, we are still searching for a compact schedule, but if it turns

out to be impossible, we can either introduce extra rounds or move some games away from the incorrect rounds. Then we may also use the hard constraint *H9*.

It should be noted here that hard and soft constraints of sports scheduling problems vary quite substantially depending on the problem instance in hand. The up-to-date collection of the most typical constraints can be found on the web [30].

## IV. SOLUTION METHOD

Our solution method is not dependent on a complete round robin tournament arrangement. Any number of games can be added to the tournament. Likewise, any number of rounds can be added to the tournament. In fact, there are no restrictions on the kind of tournament one wants to make.

The solution method is a modification of the h-HCGA algorithm presented in [31] and further extended in [32]. The algorithm is a genetic algorithm [33] with one mutation operator and no recombination operators. The two most important features of the algorithm are the greedy hill-climbing mutation (GHCM) operator, which generates a new solution candidate from the current solution, and the adaptive genetic penalty method (ADAGEN), which is a multi-objective optimization method. The pseudo code of the h-HCGA algorithm is given in Figure 1.

```
Set the time limit t and the population size n
Generate initial population of schedules by randomly
assigning games to rounds
Set better_found = 0
WHILE elapsed-time < t
   REPEAT n times
      Select a schedule S by using a marriage selection
      Apply the GHCM operator to S to get a new schedule S'
      Calculate the change Δ in fitness value
      IF Δ < = 0 THEN
         Replace S with S'
         IF Δ < 0 THEN
            better_found = better_found + 1
         ENDIF
      ENDIF
   ENDREPEAT
   IF better_found > n THEN
      Replace the worst schedule with the best schedule
      Set better_found = 0
   ENDIF
   Update the dynamic weights of the hard constraints (ADAGEN)
ENDWHILE
Choose the fittest schedule from the population
```

Fig. 1. The pseudo code of the h-HCGA algorithm.

The GHCM operator moves a game, *g1,* from its old round, *r1,* to a new round, *r2*, and then moves another game, *g2,* from round *r2* to a new round, *r3,* and so on, ending up with a sequence of moves. The initial game selection is random. The new round for the game is selected considering all possible rounds and selecting the one which causes the least increase in the cost function value when considering the relocation cost only. Moreover, the new game from that round is again selected considering all the games in that round and picking the one for which the removal causes the most decrease in the cost function value when considering the removal cost only. The operator stops if the last move causes an increase in the cost function value and if the value is larger than that of the previous non-improving move.

We improve the GHCM operator further by introducing a tabu list which prevents reverse order moves in the same sequence of moves. That is, if we move a game $g$ from round $r_1$ to round $r_2$ we do not allow $g$ to be moved back to round $r_1$ before a new sequence of moves begins. Finally, the operator is fine-tuned with the following improvements:

- sequence of moves with zero-cost are accepted
- in the case of equal cost games (or rounds) the selection is made randomly
- in the case of equal costs the longest sequence of moves will be used.

The ADAGEN method is an adaptive penalty method for multi-objective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints. This means that we are searching for a solution that minimizes the (penalty) function

$$\Sigma_i \alpha_i f_i(x) + \Sigma_i g_i(x)$$

where

$f_i(x)$ = cost of hard constraint $i$
$g_i(x)$ = cost of soft constraint $i$
$\alpha_i$ = a dynamically adjusted weight for hard constraint $i$.

The weights are updated in every $k$th generation using a somewhat complicated formula given in [31].

The reproduction operation of the algorithm is, to a certain extent, based on the steady-state reproduction [35]. We use marriage selection [34] to select a schedule from the population of schedules for a single GHCM operation. The new schedule replaces the old one if it has a better or equal fitness. Furthermore, the least fit is replaced with the best one when $n$ better schedules have been found, where $n$ is the size of the population.

The parameters of the algorithm are the same that were found to work best in [32]:

- population size = 20
- maximum move sequence in the GHCM = 10
- size of the tabulist = 5
- maximum time limit = 8 hours.

A very detailed description of the algorithm is given in [31].

## V. THE FINNISH MAJOR ICE HOCKEY LEAGUE

Ice hockey is the biggest sport in Finland, both in terms of revenue and number of spectators. The Finnish major ice hockey league is a private company with fifteen shareholders, one for each team in the league and one for the Finnish Ice Hockey Association. Each team is also a private company. The CEO of the team is responsible for getting the best possible schedule for his team.

The CEO of the league is responsible for producing the

schedule. Prior to the 2008-2009 season, the schedule was produced manually. After making the schedule for the 2007-2008 season with an increasing number of requirements and requests, the CEO realized that they were no longer able to handle the schedule manually.

Seven of the teams in the league are located in big cities (over 100,000 citizens) and the rest in smaller cities. One team is quite a long way up north, two are located in the east and the rest in the south (see Figure 2).

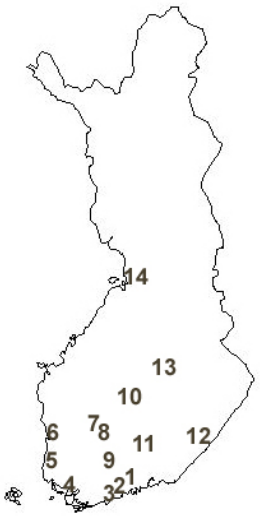


Fig. 2. The map of Finland and the fourteen teams in the Finnish major ice hockey league.

The schedule format for the league has been quite stable for many years. The base of the schedule is a quadruple round robin tournament resulting in 52 games for each team. In addition, the teams are divided into two groups of seven teams to get a few more games to play. These teams play a single round robin tournament resulting in 6 games. Therefore, there are 58 games for each team and a total of 406 games to be scheduled. The games should be scheduled on Tuesdays, Thursdays and Saturdays.

The league first fixes the dates on which the rounds will be played. In addition to the 58 rounds/days needed to generate a compact schedule, only one extra round/day was reserved to help the scheduling process.

Often, there are also parties other than the league and the teams involved in the scheduling process. Examples of such parties include TV networks and other leagues. In the case of the Finnish major ice hockey league, the TV network chooses the games to show from the final schedule, thus not affecting the scheduling process. Two of the teams also play in the Champions Hockey League.

The league and the teams gave the following requirements for the 2008-2009 season (see Section III):

*H1*. Every team plays in every round exactly once (excluding one bye per team).
*H2*. 35 home games are forbidden on certain days (e.g. one team cannot play at home in the first 13 rounds).
*H3*. 5 forbidden away games.
*H4*. 2 forbidden days.
*H5*. The Tappara and Ilves teams cannot play at home on the same day (they share a venue).
*H6*. A team cannot play at home on two consecutive calendar days (some games scheduled on Thursday are actually played on Friday).
*H7*. 45 preassigned games.
*H8*. There cannot be a break in the second round.

In addition, the league and the teams gave the following requests:

*S2*. A team cannot have more than two consecutive home games.
*S3*. A team cannot have more than two consecutive away games.
*S4*. The Kärpät team (team 14, see Fig. 1) wishes to play as many away tours (two consecutive away games) as possible. We were able to construct nine such tours.
*S5*. There must be at least eight rounds before two teams meet again.
*S7*. Four teams wish to play their home games on Thursdays and the rest on Saturdays.
*S8*. The Jokerit and HIFK teams do not want to play at home on the same day. In addition, the Blues team must play at home in an equal number of rounds with Jokerit and HIFK.
*S12*. The difference between the number of home games and the number of away games for each team should be as small as possible after each round.
*S13*. The difference between the number of games between the teams should be as small as possible after each round.

The most important requests from the league were assigned a larger weight in the ADAGEN method. The following dynamic weights were used for hard constraints:

- 50−100 for *H1*
- 5−10 for *H2* to *H5*
- *H6* to *H8* were preassigned.

## VI. THE DIFFICULTY OF THE PROCESS

Constructing a single, double or quadruple round robin tournament is quite an easy task nowadays, but when we introduce requirements and requests, the problem becomes intractable. Furthermore, being able to produce an acceptable schedule is not only about first defining the requirements and requests and then developing a suitable solution method. An essential part of the problem is the process of consulting with the various parties.

In 2007 we interviewed four team CEOs (Jokerit, Kärpät, Tappara and Ässät) and the CEO of the league. The teams were chosen so that we would get a clear picture of the requirements and requests: Jokerit is a big team with a home venue that is used for a lot of other events, Kärpät has to travel the most, Tappara plays at the same venue as another team and Ässät is a good representative of a small team

located in a small city.

The interviews gave us quite a good picture of the different requests the teams have and might have. The rest of the teams gave their requests direct to the CEO of the league. He, in turn, sent them to us. At this stage we thought we knew all the requests the teams had.

Every team CEO agreed that it is very important that the requirements and the requests are considered in the final schedule. However, when we generated the first schedule the team CEOs "discovered" that some of their requests had not been considered. The simple reason was that we were not aware of them. For some reason, not all of the requests had reached us. One reason could be that the team CEO had not actually given the requests to the CEO of the league. Another reason could be that the CEO of the league did not inform us.

Therefore, we had to generate a second schedule. We were somewhat surprised that the same thing occurred again. We got some new requests, but not all of them. We believe that the team CEOs are used to getting an unsatisfactory schedule (in the past years), which they then try to modify to better fit their requests. We base this thought on two things: first, we did not get the requests from the team CEOs, and second, the schedules prior to this season have been not so good. We will have to concentrate on this problem in the future. We have already sent them a questionnaire concerning their requests and we are arranging a meeting with them to approach this problem.

## VII. THE SCHEDULE

In this section we present our computational results for the Finnish major ice hockey league problem presented in Section V. We solved the problem using the algorithm originally presented in [31] and described in Section IV. The algorithm was run on an Intel Core 2 Duo PC with a 3.8GHz processor and 2GB of random access memory running Windows XP.

Table II shows the best solution we were able to achieve for the 2008-2009 season and the manual solution used in the previous season. Our solution (best of the 100 runs) was found in four hours of computer time, whereas the manual solution took several weeks to construct.

TABLE II
THE MANUAL SOLUTION FOR THE 2007-2008 SEASON AND OUR BEST SOLUTION FOR THE 2008-2009 SEASON

| | 2007-2008 (manual) | 2008-2009 (algorithm) |
|---|---|---|
| *H2,H3,H4,H7*: number of forbidden and preassigned games | < 50 | 87 |
| *S1*: number of rounds needed | 100 | 59 |
| *S2*: number of 3-breaks at home | 15 | 2 |
| *S4*: number of away tours | 2 | 9 |
| *S5*: less than six rounds before two teams meet again | 3 | 3 |
| *S7*: minimum number of home games on weekends | 7 | 9 |
| *S8*: Jokerit and HIFK play at home on the same day | 2 | 4 |

The CEO of the league was very satisfied with the schedules we generated and we closed a long-term contract with the league.

## VIII. CONCLUSIONS AND FUTURE WORK

We scheduled the Finnish major ice hockey league. Our algorithm found a feasible and acceptable schedule for the 2008-2009 season and the generated schedule is currently in use. We believe this is the first paper to solve a constrained minimum break problem for an ice hockey league.

Our direction for future research will be to compare our solution method to previously published methods. Furthermore, we will improve the presented algorithm. Our goal is to schedule the Finnish 1st division ice hockey league as well.

## REFERENCES


[1] K. Easton, G. Nemhauser and M. Trick, "Sports scheduling" in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, J. T. Leung, Ed. CRC Press Inc, Florida, USA, 2004, pp 1–19.

[2] R. Rasmussen and M. Trick, "Round robin scheduling - A survey", *European Journal of Operational Research* 188, 2008, pp. 617–636.

[3] J.A..M. Schreuder, "Constructing timetables for sport competitions", *Mathematical Programming Study* 13, 1980, 58–67.

[4] J. A. M. Schreuder, "Combinatorial aspects of construction of competition Dutch Professional Football Leagues", *Discrete Applied Mathematics* 35, 1992, pp. 301–312.

[5] D. de Werra, "Scheduling in sports" in *Studies on graphs and discrete programming. Amsterdam*, P. Hansen, Ed. North-Holland, 1981, pp. 381–95.

[6] D. de Werra, "Some models of graphs for scheduling sports competitions", *Discrete Applied Mathematics* 21, 1988, pp. 47–65.

[7] D. de Werra, L. Jacot-Descombes, and P. Masson, "A constrained sports scheduling problem", *Discrete Applied Mathematics* 26, 1990, pp. 41–49.

[8] K. Easton, G. Nemhauser, and M. Trick "The traveling tournament problem: description and benchmarks" in *Proc of the 7th. International Conference on Principles and Practice of Constraint Programming*, Paphos, 2001, pp. 580–584.

[9] M. Elf, M. Jünger, and G. Rinaldi, "Minimizing breaks by maximizing cuts", *Operations Research Letters* 31, 2003, pp. 343–349.

[10] S. Urrutia and C.C. Ribeiro, "Minimizing travels by maximizing breaks in round robin tournament schedules", *Electron Notes Disc Math* 18-C, 2004, pp. 227–233.

[11] J. H. Dinitz, D. Froncek, E. R. Lamken and W. D. Wallis, "Scheduling a tournament" in *The CRC Handbook of Combinatorial Designs*, C.J. Colbourn and J. H. Dinitz, Eds. CRC Press Inc, Florida, USA, (previous edition of this book was published in 1996), 2007, pp. 591–606.

[12] M.B. Wright, "Timetabling county cricket fixtures using a form of tabu search", *Journal of the Operational Research Society* 45(7), 1994, pp. 758–770.

[13] D. Costa, "An evolutionary tabu search algorithm and the NHL scheduling problem", *INFOR* 33 (3), 1995, pp. 161–178.

[14] J.P. Hamiez and J.K. Hao, "Solving the sports league scheduling problem with tabu search" in *Lecture Notes in Artificial Intelligence* 2148, 2001, pp. 24–36.

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

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

[17] M.B. Wright, "Scheduling fixtures for basketball New Zealand", *Computers & Operations Research* 33, 2006, pp. 1875–1893.

[18] T. Bartsch, A. Drexl, S. Kröger, “Scheduling the professional soccer leagues of Austria and Germany”, *Computers and Operations Research* 33(7), 2006, pp. 1907–1937.
[19] A. Schaerf, “Scheduling sport tournaments using constraint logic programming”, *Constraints* 4, 1999, pp. 43–65.
[20] M. Henz, "Scheduling a Major College Basketball Conference: Revisited", *Operations Research* 49, 2001, pp. 163-168.
[21] J.C. Régin, “Minimization of the number of breaks in sports scheduling problems using constraint programming”, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 57, 2001, pp. 115–130.
[22] D. Goossens and F. C. R. Spieksma, “Scheduling the Belgian soccer league” in *Proc of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, Brno, Czech Republic, 2006, pp. 420 - 422.
[23] F. D. Croce and D. Oliveri, “Scheduling the Italian football league: an ILP-based approach”, *Computers and Operations Research* 33(7), 2006, pp. 1963–1974.
[24] G. Durán, M. Guajardo, J. Miranda, D. Sauré, S. Souyris, A. Weintraub, “Scheduling the Chilean Soccer League by Integer Programming”, *INTERFACES* 37, 2007, pp. 539—552.
[25] R. A. Melo, S. Urrutia, C. C. Ribeiro, “Scheduling single round robin tournaments with fixed venues” in *Proc of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications,* Paris, 2007, pp. 431–438.
[26] R. Rasmussen and M. Trick, “A Benders approach for the constrained minimum break problem”, *European Journal of Operational Research* 177, 2007, pp. 198–213.
[27] G. Nemhauser and M. Trick. Scheduling a major college basketball conference, *Operations Research* 46(1), 1998, pp. 1-8.
[28] R. Rasmussen, “Scheduling a triple round robin tournament for the best Danish soccer league”, *European Journal of Operational Research* 185(2), 2008, pp. 795-810.
[29] F. Bonomo, A. Burzyn, A. Cardemil, G. Durán, J. Marenc, “An application of the traveling tournament problem: the Argentine volleyball league” *in Proc of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT),* Montreal (Canada), 2008.
[30] K. Nurmi et. al. (2008, December 28). Sports Scheduling Problem [Online]. Available: http://www.samk.fi/ssp.
[31] K. Nurmi, “Genetic Algorithms for Timetabling and Traveling Salesman Problems”, Ph.D. dissertation, Dept. Applied Math., University of Turku, Finland, 1998.
[32] K. Nurmi and J. Kyngäs, ”A Framework for School Timetabling Problem” in *Proc of the 3rd Multidisciplinary Int. Scheduling Conf.: Theory and Applications*, Paris, France, 2007, pp. 386-393.
[33] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA, 1989.
[34] P. Ross and G.H. Ballinger, “PGA - Parallel Genetic Algorithm Testbed”, Department of Articial Intelligence, University of Edinburgh, England, 1993.
[35] G. Syswerda, “Uniform Crossover in Genetic Algorithms” in *Proc of the 3rd International Conference on Genetic Algorithms*, 1989, pp. 2-9.

# Publication 4

J. Kyngäs and K. Nurmi, “Scheduling the Finnish 1st Division Ice Hockey League”, in Proc of the 22nd Florida Artificial Intelligence Research Society Conference, Florida, USA, 2009, pp. 195-200. Reprinted with the permission from the American Association for Artificial Intelligence.

# Scheduling the Finnish 1st Division Ice Hockey League

**Jari Kyngäs, Kimmo Nurmi**

Satakunta University of Applied Sciences
Tiedepuisto 3, 28600 Pori, Finland
jari.kyngas@samk.fi, cimmo.nurmi@samk.fi

**Abstract**

Generating a schedule for a professional sports league is an extremely demanding task. Good schedules have many benefits for the league, for example higher incomes, lower costs and more interesting and fairer seasons. This paper presents a successful solution method to schedule the Finnish 1st division ice hockey league. The solution method is an improved version of the method used to schedule the Finnish major ice hockey league. The method is a combination of local search heuristics and evolutionary methods. An analyzer for the quality of the produced schedules will be introduced. Finally, we propose a set of test instances that we hope the researchers of the sports scheduling problems would adopt. The generated schedule for the Finnish 1st division ice hockey league is currently in use for the season 2008-2009.

## 1. Introduction

Many new timetabling problems have been introduced in recent years. Most of the timetabling research used to concentrate on university and school timetabling, but especially rostering and sports scheduling have been quite extensively studied recently. Excellent overviews on sports scheduling can be found in (Easton et al. 2004) and (Rasmussen and Trick 2008).

In the last decade, the sports scheduling focus has moved from theoretical results to practical applications. Some of the most important theoretical results can be found in (Schreuder 1980, 1992; de Werra 1981, 1988, 1990; Easton et al. 2001; Elf et al. 2003; Urrutia and Ribeiro 2004; Dinitz et al. 2007). An extensive summary of the theoretical results can be found in (Rasmussen and Trick, 2008).

Even if quite efficient algorithms have recently been designed for sports scheduling problems, to the best of our knowledge, there are only a few cases where academic researchers have been able to close a contract with a sports league owner: the major baseball league in USA (Nemhauser and Trick 1998), the major soccer league in Austria (Bartsch et al. 2006), the 1st division soccer in Chile (Durán et al. 2006), the major basketball league in New Zealand (Wright 2006), the major soccer league in Belgium (Goossens and Spieksma 2006), the major soccer league in Denmark (Rasmussen 2008), the major volleyball league in Argentina (Bonomo et al. 2008) and the major ice hockey league in Finland (Kyngäs and Nurmi 2009). This paper presents a new case: the Finnish 1st division ice hockey league.



The sports league scheduling problems solved in this paper are constrained minimum break problems (see e.g. Rasmussen 2008). The problem is to find a schedule with the minimum number of breaks and at the same time take additional requirements and requests into account. For the sports scheduling terminology used in this paper we refer to (Kyngäs and Nurmi 2009).

The focus of this paper is to solve a highly constrained sports scheduling problem. In Section 2 we give an overview of our earlier sports scheduling algorithm. Then we present an improved version of the algorithm. Section 3 presents the Finnish 1st division ice hockey league problem. The problem is extremely difficult both in terms of finding a feasible solution and of optimizing the requests from the league. Computational results are reported in this section. It will be seen that our algorithm produces excellent results compared to the manual schedule used in the previous season. An analyzer for the quality of the produced schedules will be introduced in Section 4. The use of the analyzer is vital in producing the final schedule for the league authorities. Finally in Section 5, we propose a set of test instances that we hope the researchers of the sports scheduling problems will adopt. It will be seen that our solutions for these instances are competitive

## 2. The Improved Algorithm

Our basic algorithm for solving sports scheduling problems is presented in (Kyngäs and Nurmi 2009), (Nurmi and Kyngäs 2007) and (Nurmi 1998). The algorithm is a genetic algorithm (Goldberg 1989) with one mutation operator and no recombination operators. The two most important features of the algorithm are the greedy hill-climbing mutation (GHCM) operator, which generates a new solution candidate from the current solution, and the adaptive genetic penalty method (ADAGEN), which is a multi-objective optimization method. The algorithm uses three mechanisms to help the search procedure to avoid

local optima: genetic reproduction (Syswerda 1989), tabu search (Glover et al. 1985) and simulated annealing (Kirkpatrick et al. 1983). The use of these methods differs somewhat from their usual application (see Nurmi 1998).

The GHCM operator moves a game, *g1,* from its old round, *r1,* to a new round, *r2*, and then moves another game, *g2,* from round *r2* to a new round, *r3,* and so on, ending up with a sequence of moves. The initial game selection is random. The new round for the game is selected considering all possible rounds and selecting the one which causes the least increase in the cost function value when considering the relocation cost only. Moreover, the new game from that round is again selected considering all the games in that round and picking the one for which the removal causes the most decrease in the cost function value when considering the removal cost only.

The ADAGEN method is an adaptive penalty method for multi-objective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints.

The reproduction operation of the algorithm is, to a certain extent, based on the steady-state reproduction (Syswerda 1989). We use marriage selection (Ross and Ballinger 1993) to select a schedule from the population of schedules for a single GHCM operation. The new schedule replaces the old one if it has a better or equal fitness. Furthermore, the least fit is replaced with the best one when $n$ better schedules have been found, where $n$ is the size of the population.

Next we present two changes to the original algorithm. These changes will help the search procedure to escape from local optima as well as better explore the fitness landscape.

The original algorithm uses a simulated annealing refinement. The initial temperature $T_0$ is calculated by

$$T_0 = C^+ / \log(1/X_0) ,$$

where $X_0$ is the degree to which we want to accept an increase in the cost function (we use a value of 0.75) and $C^+$ is an average increment in the cost function for 100 random moves. This method was proposed by (van Laarhoven and Aarts 1987). The exponential cooling scheme is used to decrement the temperature:

$$T_k = \alpha T_{k-1} ,$$

where $\alpha$ is usually chosen between 0.8 and 0.995. Our new test runs showed that a good strategy is to stop the cooling at some predefined temperature. Therefore, after a certain number of iterations $m$ we will continue to accept an increase in the cost function with some constant probability $p$. Using the initial temperature given above and the exponential cooling scheme, we can calculate the value:

$$\alpha = (-1/( T_0 \log p))^{-m} .$$

Our preliminary test runs showed that we can get surprisingly good results by choosing $m$ equal to the maximum number of iterations with no improvement to the cost function and $p$ equal to 0.0015. The new annealing schedule seems to produce superior solutions compared to the well-known annealing schedules. The reason might be that it enables the search procedure to continue to escape from local optima. We will study this method more closely in our next paper.

The other change to the original algorithm concerns shuffling the current solution. A hyperheuristic (Cowling et. al. 2000) is a mechanism that chooses a heuristic from a set of simple heuristics, applies it to the current solution, then chooses another heuristic and applies it, and continues this iterative cycle until the termination criterion is satisfied. We use the same idea, but the other way around. We introduce a number of simple heuristics that are normally used to improve the current solution but, instead, we use them to shuffle the current solution - that is, we allow worse solution candidates to replace better ones in the current population. We use five shuffling operations:

1. Select a random game and move it to a random round, and do this $k_1$ times
2. Swap two random games, and do this $k_2$ times
3. Select a random round and move $k_3$ random games from that round to random rounds
4. Swap all the games in two random rounds
5. Select a random game A-B and swap it with the game B-A, and do this $k_4$ times.

We select one random shuffling operation in every $m$/20th iteration of the algorithm, where $m$ equals the maximum number of iterations with no improvement to the cost function. The best results have been obtained using the values $k_1 = 3$, $k_2 = 2$, $k_3 = 3$ and $k_4 = 2$. The shuffling seems to produce better solutions than without shuffling. The reason might again be that it enables the search procedure to continue to escape from local optima. We will again study this method more closely in our next paper.

Table 1 shows the result of the comparison between the original and the improved algorithms. Three different problems were used to compare their performance: one that minimizes just number of breaks, one that includes further restrictions (intermediate test problem) and finally a complex real-world problem. The improved algorithm using simulated annealing and shuffling refinements performs clearly better than the original algorithm.

Our algorithm uses random initial solutions. It has been claimed in many different contexts that better initial

solutions lead to better final solutions. It has also been argued that it is a good idea to use canonical schedules (Schreuder 1980) as initial solutions for sports scheduling methods since canonical schedules minimize the number of breaks. We tested our algorithm using canonical starting schedules thus producing good initial solutions. We ran the algorithm several times using both artificial problems and real-world problems. The results were clear. Canonical schedules were unable to produce better final solutions than random schedules as initial solutions to our algorithm.

Table 1: The percentage of the best solutions found for the original algorithm and for the improved algorithm. The improved algorithm uses simulated annealing and shuffling refinements.

| Problem type | Original algorithm | Improved Algorithm |
|---|---|---|
| Break optimization only | 1% | 8% |
| Intermediate test problem | 13% | 21% |
| Complex real-world problem | 7% | 14% |

## 3. The Finnish 1st Division Ice Hockey League

Ice hockey is the biggest sport in Finland, both in revenue and number of spectators. The Finnish 1st division ice hockey league is managed by the Finnish Ice Hockey Association. The Competition Manager of the league is responsible for producing the schedule. Prior to the 2008−2009 season, the schedule was produced manually. When the manager heard that we had scheduled the major league (Kyngäs and Nurmi 2009), he contacted us. He told us that the 1st division is an even more difficult problem than the major league. We agreed to generate a sample schedule for them.

The league has twelve teams. Two of the teams are located in big cities (over 100 000 citizens) and the rest in smaller cities. One team is quite far up north, one on the west coast, three teams are located in the east and the rest in Central Finland (see Figure 2).

The schedule format for the league has been stable for many years. The basis of the schedule is a quadruple round robin tournament concluding in 44 games for each team. In addition, each team plays at home against the Finnish U20 team (national team for players under 20 years of age). Therefore, there are 45 games for each team and a total of 276 games to be scheduled. The games should be scheduled on Wednesdays and Saturdays.

The league first fixes the dates on which the rounds will be played. They only fix 44 dates - that is. the basic schedule should be a compact schedule. The U20 games are preassigned to given dates.

Often, there are also parties other than the league and the teams involved in the scheduling process. Examples of such parties include TV networks and other leagues. In the case of the Finnish 1st division ice hockey league the TV network chooses the games to show before the scheduling process. These games are preassigned to given rounds. The Finnish major ice hockey league introduces further requirements. Five teams in the 1st division are located in (or very close to) the same cities as the teams in the major league. The major league is scheduled first and these five teams should not play at home on the same days as their counterparts in the major league. Furthermore, three other teams are competing with the Finnish major basketball league for the same spectators. This league is again scheduled first, so these three teams should not play at home on certain days.

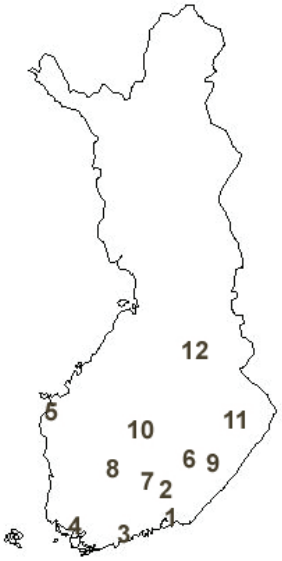


Figure 2: The map of Finland and the twelve teams in the Finnish 1st division ice hockey league.

For the following terminology and notation we refer to (Kyngäs and Nurmi 2009). The league and the teams gave the following requirements for the 2008−2009 season:

*H1*. Every team plays in every round exactly once (a compact schedule).
*H2*. 36 home games are forbidden on certain days.
*H6*. A team cannot play at home on two consecutive calendar days.
*H7*. 61 preassigned games.
*H8*. There cannot be a break in the second round.

In addition, the league and the teams gave the following requests:

*S2*. A team cannot have more than two consecutive home games.
*S3*. A team cannot have more than two consecutive away games.
*S4*. The LeKi, Hokki, Jokipojat, Kiekko-Vantaa and TuTo teams wish to play a few away tours.

*S5*. There must be at least six rounds before two teams meet again.
*S7*. All teams wish to play their home games on Saturdays.
*S9*. The D-Team, HeKi, Kiekko-Vantaa, LeKi and TuTo teams do not want to play at home on the same day as their major league counterparts.
*S12*. The difference between the number of home games and the number of away games for each team should be as small as possible after each round.

The most important requests from the league were assigned a larger weight in the ADAGEN method. The following weights were used for hard constraints:

- 3-25 for *H1*.
- 3-10 for *H2*.
- *H6* to *H8* were preassigned.

The construction of the schedule was quite a difficult task. First of all, there were a considerable number of restrictions − over 10% more than in the Finnish major ice hockey league (Kyngäs and Nurmi 2009). Secondly, the teams had many more wishes than the teams in the major league.

Table 2: The manual solution for the 2007−2008 season and our best solution for the 2008−2009 season. Unfortunately it is not possible to calculate values for other constraints from the 2007−2008 manual schedule.

| | 2007-2008 (manual) | 2008-2009 (algorithm) |
|---|---|---|
| *H1*: number of rounds (does not include those needed for the away tours) | 64 | 44 |
| *H2,H7*: number of forbidden and preassigned games | < 60 | 97 |
| *S2,S3*: number of 3-breaks (at home + away) | 68 | 2 + 4 |
| *S4*: number of away tours | 13 | 17 |
| *S7*: minimum number of home games on weekends | 9 | 12 |

We generated two schedules as we did for the major league. After the generation of the first schedule the Competition Manager discovered that he had forgotten a few restrictions. We added them to the program and produced a second schedule. The second schedule was accepted and only two games were relocated both due the fact that the teams would have had to play three consecutive away games because of home venue unavailability. So the schedule is in use almost as generated. Table 2 shows our solution for the 2008−2009 season and a comparison with the solution produced manually for the 2007−2008 season. The algorithm was run on an Intel Core 2 Duo PC with a 3.8GHz processor and 2GB of random access memory running Windows XP. Our solution (best of the 50 runs) was found in six hours of computer time, whereas the manual solution took several weeks to construct.

The Competition Manager was very satisfied with the schedules we generated and we closed a contract with the league.

## 4. The Analyzer

It is very difficult to examine the quality of the generated schedule. Even if someone were to have the patience to do it, it would take quite a long time. To ease this process we have made an analyzer for analyzing sports schedules.

Actually we have made two analyzers: one for analyzing the output file of our program (algorithm) and one for analyzing any general sports schedule. The analyzer takes the schedule, requirements and requests as input, given as text files and produces a simple text file as output, where it is very easy to examine the conflicts in the schedule. The output file details each restriction and possible conflicts for each team in a readable form. A few examples:

- All breaks
- Weekday preferences are listed day by day (given lower bound, given upper bound, actual value).
- Every game that has a violation in the k-value.

The output file is excellent for presenting the results to the customer. When we introduced our program to the Competition Manager we generated three somewhat different schedules for the first half of the season. The manager could very quickly see the benefits of our program just by inspecting the output file. We claim that the use of the analyzer is vital in producing the final schedule for the league authorities.

## 5. A Set of Test Instances

The generation of standard test problems does not receive much attention. Some benchmark instances for round robin tournaments have been introduced in Henz (2000). For the Traveling Tournament Problem (Easton et al. 2001), test instances can be found in (Trick 2008). No set of standard test instances exists for the constrained minimum break problem.

Researchers quite often only solve some special artificial cases or one real-world case. The strength of random test instances is the ability to produce many problems with many different properties. The strength of practical cases is

self-explanatory. However, an algorithm performing well on one practical problem may not perform satisfactorily on another practical problem. Our future work will present a set of both artificial and practical test instances for the constrained minimum break problem. In this section we present a collection of test instances found in the literature as well as some new test instances.

Table 3: Twelve 2RR test instances: R50 (1), R100 (2), B10 (3), B12 (4), B14 (5), B10K3 (6), B12K3 (7), B12K10 (8), R14K7P208 (9), B8K0P30 (10), B8K2P30 (11), B10K2C4 (12).

| ID | n | Break min. | k | #Constr. | Optimal #breaks | Our solution |
|---|---|---|---|---|---|---|
| **1** | 50 | No | 0 | 0 | – | found |
| **2** | 100 | No | 0 | 0 | – | found |
| **3** | 10 | Yes | 0 | 0 | 8 | 8 |
| **4** | 12 | Yes | 0 | 0 | 10 | 10 |
| **5** | 14 | Yes | 0 | 0 | 12 | 14 |
| **6** | 10 | Yes | 3 | 0 | ? | 16 |
| **7** | 12 | Yes | 3 | 0 | 16 | 22 |
| **8** | 12 | Yes | 10 | 0 | ? | 26 |
| **9** | 14 | No | 7 | 208 P | – | found |
| **10** | 8 | Yes | 0 | 30 P | ? | 10 |
| **11** | 8 | Yes | 2 | 30 P | ? | 12 |
| **12** | 10 | Yes | 2 | 4 C | ? | 10 |

To the best of our knowledge, the best test instances presented in the literature so far are those by (Rasmussen and Trick 2007). We use four of their problems, two of which are slight modifications. All twelve test instances are double round robins. Table 3 shows the instances. The first two (abbreviated as Rn) are simple round robin problems where the only challenge is to find a round robin tournament. In the next three instances (Bn) the challenge is to find the minimum number of breaks. The next three instances (BnK) are also break minimization problems, but in these instances two games with the same opponents must be separated by at least $k = 3$ or $k = 10$ rounds. In the instance R14K7P208 the challenge is again just to find a round robin tournament, but now with $k = 7$. Furthermore, there are four home game restrictions and four away game restrictions in each round totaling a number of 208 restrictions (place constraints). The next two instances (B8) are break minimization problems with place constraints and the other one with $k = 2$. Here we had to modify the original problems by (Rasmussen and Trick 2007) because their place constraints would have caused extra breaks to occur. The final instance B10K2C4 introduces complementary constraints – that is, two teams cannot play at home at the same day. The instance includes four complementary constraints.

It should be noted that our algorithm was not designed to merely minimize the number of breaks but to solve complex real-world problems. However, we claim that our algorithm also works very well for the artificial test instances. We were able to find the best possible solution for five of the test instances. For three of the instances the optimum is not yet known. The up-to-date collection of test instances can be found on the web (Nurmi and Kyngäs 2009).

## 6. Conclusions and Future Work

We scheduled the Finnish 1st division ice hockey league. Our algorithm found a feasible and an acceptable schedule for the the 2008−2009 season. The generated schedule is currently in use. We also proposed a set of test instances that we hope the researchers of the sports scheduling problems will adopt. Our solutions to the test instances were competitive.

Our direction for future research will be to further study the improved algorithm and its various parameters. We will also publish an extensive set of both real-world instances and test instances for the constrained minimum break problem. We have already set up a group of collaborators for this goal.

## References


Bartsch, T., Drexl, A., and Kröger, S. 2006. “Scheduling the professional soccer leagues of Austria and Germany”, *Computers and Operations Research* 33(7): 1907–1937.

Bonomo, F., Burzyn, A., Cardemil, A., Durán, G., and Marenc, J. 2008. “An application of the traveling tournament problem: the Argentine volleyball league”. In *Proc. of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT),* Montreal (Canada).

Cowling, P., Kendall, G., and Soubeiga, E. 2000. A hyperheuristic Approach to Scheduling a Sales Summit. In *Proc. of the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 176-190.

Dinitz, J. H. , Froncek, D. , Lamken, E. R. , and Wallis, W. D. 2007. “Scheduling a tournament” in *The CRC Handbook of Combinatorial Designs*, C.J. Colbourn and J. H. Dinitz, Eds. CRC Press Inc, Florida, USA, 591–606 (previous edition of this book was published in 1996).

Durán, G., Guajardo, M., Miranda, J., Sauré, D., Souyris, S., and Weintraub, A. 2007. “Scheduling the Chilean

Soccer League by Integer Programming", *INTERFACES* 37: 539–552.

Easton, K., Nemhauser, G., and Trick, M. 2001. "The traveling tournament problem: description and benchmarks". In *Proc. of the 7th. International Conference on Principles and Practice of Constraint Programming*, Paphos, 580–584.

Easton, K., Nemhauser G., and M. Trick. 2004. "Sports scheduling" in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, J. T. Leung, Ed. CRC Press Inc, Florida, USA, pp 1–19.

Elf, M., Jünger, M., and Rinaldi, G. 2003. "Minimizing breaks by maximizing cuts", *Operations Research Letters* 31: 343–349.

Glover, F., McMillan, C., and Novick, B. 1985. Intercative Decision Software and Computer Graphics for Architectural and Space Planning. *Annals of Operations Research 5*: 557-573.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA.

Goossens, D., and Spieksma, F. C. R. 2006. "Scheduling the Belgian soccer league". In *Proc. of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, Brno, Czech Republic, 420–422.

Henz, M., Müller, T., Thiel, S., and van Brandenburg, M. (Created 2000). Benchmarks and results for round robin tournaments. http://www.comp.nus.edu.sg/~henz/roundrobin_benchmarks/.

Kirkpatrick, S., Gelatt, C.D. Jr, and Vecchi, M.P. 1983. Optimization by Simulated Annealing. *Science 220*: 671-680.

Kyngäs, J., and Nurmi, K. 2009. Scheduling the Finnish Major Ice Hockey League. IEEE Symposium on Computational Intelligence in Scheduling, Nashville, USA, accepted for publication.

van Laarhoven, P.J.M., and Aarts, E.H.L. 1987. *Simulated annealing: Theory and applications*. Kluwer Academic Publishers.

Nemhauser, G., and Trick, M. 1998. Scheduling a major college basketball conference, *Operations Research* 46(1): 1–8.

Nurmi, K., and Kyngäs, J. (Last update 28.1.2009). Sports Scheduling Problem [Online]. Available: http://www.samk.fi/ssp.

Nurmi, K. 1998. "Genetic Algorithms for Timetabling and Traveling Salesman Problems". Ph.D. diss., Dept. of Applied Math., University of Turku, Finland.

Nurmi, K., and Kyngäs, J. 2007. "A Framework for School Timetabling Problem". In *Proc. of the 3rd Multidisciplinary Int. Scheduling Conf.: Theory and Applications*, Paris, France, 386–393.

Rasmussen, R. 2008. "Scheduling a triple round robin tournament for the best Danish soccer league", *European Journal of Operational Research* 185(2): 795–810.

Rasmussen, R. and Trick, M. 2007. "A Benders approach for the constrained minimum break problem", *European Journal of Operational Research* 177: 198–213.

Rasmussen, P., and Trick, M. 2008. "Round robin scheduling - A survey", *European Journal of Operational Research* 188, pp. 617–636.

Ross, P. and Ballinger, G.H. 1993. "PGA - Parallel Genetic Algorithm Testbed", Department of Articial Intelligence, University of Edinburgh, England.

Schreuder, J.A..M. 1980. "Constructing timetables for sport competitions", *Mathematical Programming Study* 13, 58–67.

Schreuder, J. A. M. 1992. "Combinatorial aspects of construction of competition Dutch Professional Football Leagues", *Discrete Applied Mathematics* 35: 301–312.

Syswerda, G. 1989. "Uniform Crossover in Genetic Algorithms". In *Proc of the 3rd International Conference on Genetic Algorithms*, 2–9.

Trick, M. (Last update 31.7.2008). Challenge Traveling Tournament Instances, http://mat.gsia.cmu.edu/TOURN.

Urrutia, S., and Ribeiro, C.C. 2004. "Minimizing travels by maximizing breaks in round robin tournament schedules", *Electron Notes Disc Math* 18-C: 227–233.

de Werra, D. 1981. "Scheduling in sports" in *Studies on graphs and discrete programming. Amsterdam*, P. Hansen, Ed. North-Holland, pp. 381–95.

de Werra, D. 1988. "Some models of graphs for scheduling sports competitions", *Discrete Applied Mathematics* 21: 47–65.

de Werra, D., Jacot-Descombes, L., and Masson, P. 1990. "A constrained sports scheduling problem", *Discrete Applied Mathematics* 26: 41–49.

Wright, M.B. 2006. "Scheduling fixtures for basketball New Zealand", *Computers & Operations Research* 33, 1875–1893.

# Publication 5

K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngäs, J. Marenco, C.C. Ribeiro, F. Spieksma, S. Urrutia and R. Wolf-Yadlin, “A Framework for Scheduling Professional Sports Leagues”, in Ao, Sio-Iong (ed.): IAENG Transactions on Engineering Technologies Volume 5, Springer, USA, 2010, pp. 14-28. Reprinted with the permission from Springer.

# A Framework for Scheduling Professional Sports Leagues

Kimmo Nurmi[a], Dries Goossens[b], Thomas Bartsch[c], Flavia Bonomo[d], Dirk Briskorn[e], Guillermo Duran[d], Jari Kyngäs[a], Javier Marenco[g], Celso C. Ribeiro[h], Frits Spieksma[b], Sebastián Urrutia[i] and Rodrigo Wolf-Yadlin[f]

[a]*Satakunta University of Applied Sciences, Tiedepuisto 3, 28600 Pori, Finland, email: kimmo.nurmi@samk.fi, jari.kyngas@samk.fi*
[b]*Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium, email: dries.goossens@econ.kuleuven.be, frits.spieksma@econ.kuleuven.be*
[c]*SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, email: thomas.bartsch@sap.com*
[d]*CONICET and FCEN, University of Buenos Aires, UBA Ciudad Universitaria, pab I, Int. Guiraldes s/n (1428) Buenos Aires, Argentina, email: fbonomo@dc.uba.ar, gduran@dm.uba.ar*
[e]*Christian-Albrechts-Universität, Olshausenstr. 40, 24098 Kiel, Germany, email: briskorn@wiso.uni-koeln.de*
[f]*DII, University of Chile, República 701,Santiago, Chile, email: rwolf@dii.uchile.cl*
[g]*National University of General Sarmiento, J. M. Gutiérrez 1150 (1613) Los Polvorines, Buenos Aires, Argentina, email: jmarenco@ungs.edu.ar*
[h]*Universidade Federal Fluminense, Department of Computer Science, Rua Passo da Pátria 156, Niterói, RJ 2431-240, Brazil, email: celso@ic.uff.br*
[i]*Federal University of Minas Gerais, Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-010, Brazil, email: surrutia@dcc.ufmg.br*

**Abstract.** This paper introduces a framework for a highly constrained sports scheduling problem which is modeled from the requirements of various professional sports leagues. We define a sports scheduling problem, introduce the necessary terminology and detail the constraints of the problem. A set of artificial and real-world instances derived from the actual problems solved for the professional sports league owners are proposed. We publish the best solutions we have found, and invite the sports scheduling community to find solutions to the unsolved instances. We believe that the instances will help researchers to test the value of their solution methods. The instances are available online.

**Keywords:** Sports Scheduling, Real-World Scheduling.
**PACS:** 02.10.Ox

## INTRODUCTION

Professional sports leagues are big businesses. An increase in revenue comes from many factors: an increased number of spectators both in stadiums and via TV networks, reduced traveling costs for teams, a more interesting tournament for the media and sports fans, and a fairer tournament for the teams. Furthermore, TV networks buy the rights to broadcast the games and in return want the most attractive games to be scheduled at certain times.

One major reason for the increased academic interest in sports scheduling was the introduction of the traveling tournament problem [1], where the total distance traveled by the teams is minimized. Since the 1990s the evolution of sports scheduling has closely tracked the development of computers. In recent years microcomputers have reached a level of being powerful enough for demanding computational tasks in practical areas of sports scheduling. This is the second of the four reasons for the current interest in sports scheduling. The third reason is that new efficient algorithmic techniques have been developed to tackle previously intractable problems, and the fourth is that sports leagues are now organized more professionally than before and it has been realized that a good schedule is vital for a league's success.

Excellent overviews of sports scheduling can be found in [2]-[5]. An extensive bibliography can be found in [6] and an annotated bibliography in [7]. Successful methods of solving sports scheduling problems include ant algorithms [8],[9], constraint programming [2],[10]-[11], evolutionary algorithms [12]-[14], integer programming [15]-[20], metaheuristics [21]-[23], simulated annealing [24]-[26] and tabu search [27]-[29].

To the best of our knowledge, there are not many cases where academic researchers have been able to close a contract with a sports league owner. We are aware of the following: the major soccer league in The Netherlands [30], the major baseball league in the USA [31], the major soccer league in Austria [32], the 1st division soccer league in Chile [33], the major basketball league in New Zealand [26], the major soccer league in Belgium [34], the major soccer league in Denmark [35], the major volleyball league in Argentina [36], the major and 1st division ice hockey leagues in Finland [37],[38] and the major soccer league in Brazil [19].

## SPORTS SCHEDULING TERMINOLOGY

In a sports competition, *n teams* play against each other over a period of time according to a given timetable. The teams belong to a *league*. In general, *n* is assumed to be an even number. A dummy team is added if a league has an odd number of teams. The league organizes *games* between the teams. Each game consists of an ordered pair of teams *(i, j)*. The first team, *i*, plays *at home* - that is, uses its own *venue* (stadium) for a game - and the second team, *j*, plays *away*. Games are scheduled in *rounds*. Each round is played on a given *day*. A *schedule* consists of games assigned to rounds. A schedule is *compact* if each team plays exactly one game in each round; otherwise it is *relaxed*. If a team has no game in a round, it is said to have a *bye*.

If a team plays two home or two away games in two consecutive rounds, it is said to have a *break*. In general, for reasons of fairness, breaks are to be avoided. The problem of finding a schedule with the minimum number of breaks is the *minimum break problem*. However, a team can prefer to have two or more consecutive away games if it is located far from the opponent's venues, and the venues of these opponents are close to each other. A series of consecutive away games is called an *away tour*. We call a schedule *k-balanced* if the numbers of home and away games for each team differ by at most *k* in any stage of the tournament. Teams can be partitioned into *strength groups*. Strength groups can be formed on the basis of the expected strengths of the teams. Teams can also be grouped by their location.

In a *round robin tournament* every team plays against every other team a fixed number of times. Most sports leagues play a double round robin tournament (*2RR*), where the teams meet twice (once at home, once away), but quadruple round robin tournaments (*4RR*) are also quite common. The number of rounds in a compact single round robin tournament (*1RR*) is $n - 1$ and the number of games is $n\ (n - 1)/2$. If $n$ is even, it is always possible to construct a schedule with $n - 2$ breaks, and this number is the minimum [30]. A *mirrored* double round robin tournament (*M2RR*) is a tournament where every team plays against every other team once in the first $n - 1$ rounds, followed by the same games with reversed venues in the last $n - 1$ rounds. For an M2RR, it is always possible to construct a schedule with exactly $3n - 6$ breaks [39].

Table I shows an example of a compact mirrored 2RR with n = 6. The schedule has no breaks for teams 1 and 5, three breaks for teams 2 and 3, three-in-a-row home games for team 6 and five-in-a-row away games for team 4.

**TABLE 1.** A compact mirrored double round robin tournament with six teams.

| **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** | **R10** |
|---|---|---|---|---|---|---|---|---|---|
| 1 – 6 | 3 – 1 | 1 – 5 | 2 – 1 | 1 – 4 | 6 – 1 | 1 – 3 | 5 – 1 | 1 – 2 | 4 – 1 |
| 2 – 5 | 6 – 2 | 2 – 4 | 5 – 3 | 3 – 2 | 5 – 2 | 2 – 6 | 4 – 2 | 3 – 5 | 2 – 3 |
| 4 – 3 | 5 – 4 | 3 – 6 | 6 – 4 | 6 – 5 | 3 – 4 | 4 – 5 | 6 – 3 | 4 – 6 | 5 – 6 |

If a team plays against team $i$ in one round, and against team $j$ in the next round, we say that team $i$ gives a *carry-over effect* (COE) to team $j$. If we define $c_{ij}$ as the number of carry-over effects that $i$ gives to $j$, we can compute the so-called *COE value* of the schedule as $\sum_{i,j} c_{ij}^2$. The problem of finding a schedule with the minimum COE value is the carry-over effects value minimization problem. A lower bound value is $rn\ (n - 1)$, where $r$ is the number of round robins; schedules that attain this lower bound are called *balanced schedules*. Brazil [19].

## THE SPORTS SCHEDULING PROBLEM

To solve a real-world sports scheduling problem it is apparent that a profound understanding of the relevant requests and requirements presented by the league is a prerequisite for developing an effective solution method. In most cases the most important goal is to minimize the number of breaks. There are various reasons why breaks should be minimized in a sports schedule: fans do not like long periods without home games, consecutive home games reduce gate receipts, and long sequences of home or away games might influence the team's current position in the tournament. Apart from minimizing the number of breaks, several other issues play a role in sports scheduling, e.g. minimizing the total traveling distance, creating a compact schedule, avoiding a team playing against all the strong teams consecutively.

We give next an outline of the typical constraints of the *sports scheduling problem*. We believe that these constraints are representative of many scheduling scenarios within the area of sports scheduling. We make no strict distinction between hard and soft constraints. They will be given by the instances themselves. The goal is to find a feasible solution that is the most acceptable for the sports league owner. That is, a solution that has no hard constraint violations and that minimizes the weighted sum of

the soft constraint violations. The weights will also be given by the instances themselves. A league can use a mixture of the following constraints as a framework for its schedule generation. The constraints were first introduced in [40]. Here we group the constraints to improve the readability.

Basis

C01. There are at most $R$ rounds available for the tournament.
C02. A maximum of $m$ games can be assigned to round $r$.
C03. Each team plays at least $m_1$ and at most $m_2$ games at home.
C22. Two teams play against each other at home and in turn away in 3RR or more.

Home and Away

C04. Team $t$ cannot play at home in round $r$.
C05. Team $t$ cannot play away in round $r$.
C06. Team $t$ cannot play at all in round $r$.
C07. There should be at least $m_1$ and at most $m_2$ home games for teams $t_1$, $t_2$, … on the same day.
C08. Team $t$ cannot play at home on two consecutive calendar days.
C09. Team $t$ wants to play at least $m_1$ and at most $m_2$ away tours on two consecutive calendar days.
C23. Team $t$ wishes to play at least $m1$ and at most $m_2$ home games on $weekday_1$, $m_3 - m_4$ on $weekday_2$ and so on.

Break

C12. A break cannot occur in round $r$.
C13. Teams cannot have more than $k$ consecutive home games.
C14. Teams cannot have more than $k$ consecutive away games.
C15. The total number of breaks must not be larger than $k$.
C16. The total number of breaks per team must not be larger than $k$.
C17. Every team must have an even number of breaks.
C18. Every team must have exactly $k$ number of breaks.
C35. A break of type *A/H* for team $t$ must occur between rounds $r_1$ and $r_2$.

Game

C10. Game *h-team* against *a-team* must be preassigned to round r.
C11. Game *h-team* against *a-team* must not be assigned to round $r$.
C24. Game *h-team* against *a-team* cannot be played before round $r$.
C25. Game *h-team* against *a-team* cannot be played after round $r$.
C34. Game *h-team* against *a-team* can only be carried out in a subset of rounds $r_1$, $r_2$, $r_3$, ...

Tournament quality

C19. There must be at least $k$ rounds between two games with the same opponents.
C20. There must be at most $k$ rounds between two games with the same opponents.
C21. There must be at least $k$ rounds between two games involving team $t_1$ and any team from the subset $t_2$, $t_3$, ...
C26. The difference between the number of played home and away games for each team must not be larger than $k$ in any stage of the tournament (a $k$-balanced schedule).
C27. The difference in the number of played games between the teams must not be

larger than $k$ in any stage of the tournament (in a relaxed schedule).

C36. The carry-over effects value must not be larger than $c$.

Strength group

C28. Teams should not play more than $k$ consecutive games against opponents in the same strength group.

C29. Teams should not play more than $k$ consecutive games against opponents in the strength group $s$.

C30. At most $m$ teams in strength group $s$ should have a home game in round $r$.

C31. There should be at most $m$ games between the teams in strength group $s$ between rounds $r_1$ and $r_2$.

C32. Team $t$ should play at least $m_1$ and at most $m_2$ home games against opponents in strength group $s$ between rounds $r_1$ and $r_2$.

C33. Team $t$ should play at least $m_1$ and at most $m_2$ games against opponents in strength group $s$ between rounds $r_1$ and $r_2$.

Next we consider some examples of these constraints. If the number of available rounds specified in constraint C01 is higher than the minimal number of rounds needed to complete the tournament, a relaxed schedule is allowed, and constraint C02 can be used to set the maximum number of games for each round. Constraint C03 is used when the number of home and away games is not the same for all teams (valid for 1RR and 3RR). A team cannot play at home (C04) if its venue is unavailable due to some other event. A team cannot play away (C05) if it has an anniversary on that day and it requests to play at home. If a team has a game in another league, it cannot play at all on certain round (C06). If two teams share a venue, constraint C07 can be used to avoid the two teams playing at home in the same round, by setting $m_1 = 0$ and $m_2 = 1$ for this pair of teams. Constraints C08 and C09 are used to schedule away tours.

Some games can be preassigned to certain rounds using constraint C10. The constraint is also useful for preassigning away tours or preassigning special mini-tournaments between some teams on weekends. When a game should not necessarily be played in a specific round, but rather in some period of the season, this can be expressed using constraint C34. When there is another important event on a specific day (round) that can compete in interest with a league game, there should not be any "popular" game in that round (C11).

Even if the main goal often is to find a schedule with the minimum number of breaks, constraints from C12 to C18 can also be used to set requirements concerning the number of breaks. Furthermore, quite often a break is not allowed in the second or in the last round (C12). In some cases, a break is desirable in some period of the season, which can be enforced using constraint C35.Two games between the same opponents cannot usually be played on close days (C19). Constraints C19 and C20 used together results in a mirrored schedule if $k$ is set to $n - 1$. If a triple or quadruple round robin tournament is played, it's common that two teams should play against each other at home and in turn away (C22).

Most of the teams prefer to play their home games at weekends to maximize the number of spectators. However, some teams might prefer weekdays to maximize the number of business spectators. Constraint C23 is used to limit a team's number of

home games on a weekday (e.g. Wednesday), assuming that the day on which it will be played is known for every round. Constraint C24 can be used in the latter 2RR when 4RR is solved by splitting it into two 2RRs. The constraint ensures that there are at least a given number of rounds between two games with the same opponents (see also C19). A team might also prefer home games against important opponents in the second half of the season, as these games are likely to be more attractive near the end of the competition. For the same reason, a game between local rivals might be preferred to be scheduled early in the season (C25).

Minimizing the difference between the number of played home and away games for each team at any stage of the season (C26) is an important fairness criterion. If not all teams play in each round, i.e. a relaxed schedule is to be generated, another fairness criterion is to minimize the difference in the number of played games between the teams (C27).

Another goal can be to avoid a team playing against extremely weak or extremely strong teams in consecutive rounds (C29), or to avoid consecutive games between teams located nearby (C28). A TV broadcaster might require that the most interesting teams should not all play at home on the same day (C30). Constraint C31 can be used to enforce a balanced spread of games between top teams over the season. Constraint C32 can be used to ensure that a team has a home game against a top team in each half of the season. Constraint C33 can be used to make sure that each team plays against a strong team in the first rounds of the season. Finally, for sports where carry-over effects could influence the result of the tournament, these effects can be balanced using constraint C36.

We model the sports scheduling problem using a simple text file format. The file format consists of a header section and a constraint section. The header section has eight elements:

# *benchmark instance*, the name of the instance
# *number of teams*
# *team names*
# *number of round robins*
# *additional games*, which can be used to set other games besides those in the round robins
# *number of rounds*
# *weekdays for rounds*
# *strength groups*

The constraint section has one element for each constraint in use:

*# C04. Team t cannot play at home in round r*
*# C07. There should be at least $m_1$ and at most $m_2$ home games for teams $t_1$, $t_2$, …*
*# and so on.*

The detailed and up-to-date information on the file format and sample files can be found in [41]. We believe that this model helps researchers to evaluate, compare and exchange their solution methods.

Notice that some constraints are in fact generalizations of others (e.g. constraint C34 is a generalization of C24 and C25), or could be expressed using a series of other constraints (e.g. ensuring that a team does not play at all in a particular round (C06) can also be done by specifying that a team does not play at home (C04) or away (C05)

in that round). However, we chose not to reduce the set of constraints to its most compact form because we think these redundant constraints make it easier to understand what the requirements for a tournament are, and/or reduce the number of lines in the described text file format.

## ARTIFICIAL BENCHMARK INSTANCES

The generation of standard benchmark problems has not received much attention. Some test instances for round robin tournaments have been introduced in [42]. Kyngäs and Nurmi [38] presented a set of artificial test instances for the constrained minimum break problem. For the traveling tournament problem, test instances can be found in [43]. No set of standard test instances has previously been published for the real-world constrained minimum break problem.

Researchers quite often only solve some special artificial cases or one real-world case. The strength of random test instances is the ability to produce many problems with many different properties. Still, they should be sufficiently simple for each researcher to be able to use them in their test environment. The strength of practical cases is self-explanatory. However, an algorithm performing well on one practical problem may not perform satisfactorily on another practical problem; which is why we present a collection of test instances for both artificial and real-world cases. We start with artificial cases.

Table II shows 22 test instances some of which have earlier been introduced in [38]. All but two must be compact schedules (see constraint C01). Most of the instances are double round robin tournaments (RR = 2). The number of teams ($n$) varies between 8 and 100. The challenge is to find either a round robin tournament (C15 = empty) or a round robin tournament that minimizes the number of breaks (C15 = Min). In some instances there must be at least k rounds before two teams meet again (see constraint C19). Additional constraints may include place constraints (see constraints C04 and C05) and complementary constraints (see constraint C07). The only soft constraints in these instances are C15, all other constraints are hard.

Also note the following points:

- R14K7P208 has four home game restrictions and four away game restrictions in each round totaling a number of 208 C04 and C05 constraints.
- In the instances where C07 constraints exist, teams 1 and 2, teams 3 and 4, and so on cannot play at home at the same day - that is, $m_1 = 0$ and $m_2 = 1$.
- R16P116C23 and B16K12P116C1 are constructed using data from one season of the Finnish major ice hockey league for players under 20 years of age. The games should be scheduled for 57 rounds instead of the 45 rounds needed for a compact schedule. Furthermore, the home teams for the third round-robin are given.
- The home teams for B16C30 are given.

The optimal number of breaks is only known for seven instances [7]. The other best solutions were found while preparing this article. These solutions provide a good starting point for communications between sports scheduling researchers. We hope these test instances will lay the foundation for the standard benchmark instances for

sports scheduling problems.The next section introduces another set of instances which further widens these aims.

**TABLE 2.** Artificial benchmark instances: R14K7P208 (1), R16P116C23 (2), R100C8 (3), B8 (4), B8K0P30 (5), B8K2P30 (6), B10 (7), B10K2C4 (8), B10K3 (9), B12 (10), B12K3 (11), B12K8 (12), B12K8C4 (13), B12K8P30 (14), B12K8P30C3 (15), B12K8P30C4 (16), B12K10 (17), B14 (18), B16 (19), B16K3 (20), B16C30 (21), B16K12P116C1 (22).

| ID | RR | *n* | C01 | C15 | C19 | C04+05 | C07 | C10 | Best sol |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 14 | | | 7 | 208 | | | found |
| 2 | 3 | 16 | 57 | | | 116 | 23 | | found |
| 3 | 2 | 100 | | | | | 8 | | found |
| 4 | 2 | 8 | | Min | | | | | 6* |
| 5 | 2 | 8 | | Min | | 30 | | | 10 |
| 6 | 2 | 8 | | Min | 2 | 30 | | | 12 |
| 7 | 2 | 10 | | Min | | | | | 8* |
| 8 | 2 | 10 | | Min | 2 | | 4 | | 10 |
| 9 | 2 | 10 | | Min | 3 | | | | 16 |
| 10 | 2 | 12 | | Min | | | | | 10* |
| 11 | 2 | 12 | | Min | 3 | | | | 16* |
| 12 | 2 | 12 | | Min | 8 | | | | 24 |
| 13 | 2 | 12 | | Min | 8 | | 4 | | 24 |
| 14 | 2 | 12 | | Min | 8 | 30 | | | 30 |
| 15 | 2 | 12 | | Min | 8 | 30 | 3 | | 34 |
| 16 | 2 | 12 | | Min | 8 | 30 | 4 | | 1H+42 |
| 17 | 2 | 12 | | Min | 10 | | | | 30 |
| 18 | 2 | 14 | | Min | | | | | 12* |
| 19 | 2 | 16 | | Min | | | | | 14* |
| 20 | 2 | 16 | | Min | 3 | | | | 20* |
| 21 | 1 | 16 | | Min | | 30 | | | 36 |
| 22 | 3 | 16 | 57 | Min | 12 | 116 | 1 | 45 | 30 |

* known optimum

## REAL-WORLD BENCHMARK INSTANCES

There are not many cases where academic researchers have been able to close a contract with a sports league owner. The real-world instances introduced in this section are based on such cases. In order not to reveal league secrets the instances might slightly differ from the actual problems solved for the league owners. The instances are derived from Finnish, Austrian, German, Argentine, Chilean, Belgian and Brazilian leagues. We give a short description of these leagues. In all the leagues the most important goal is to minimize the number of breaks.

The Finnish Major Ice Hockey League (FIN1) has 14 teams. The basis of the schedule is a quadruple round robin tournament resulting in 52 games for each team. In addition, the teams are divided into two groups of seven teams to get a few more games to play. These teams play a single round robin tournament resulting in 6 games. Therefore, there are 58 games for each team and a total of 406 games to be scheduled. The three most important goals are to have no home games on the same day for some team pairs (C07), to have at least 5 rounds between two games with the same opponents (C19) and to have an equal number of home games on Saturdays for all teams (C23). For more details, refer to [37]. The Finnish 1st Division Ice Hockey

League (FIN2) has 12 teams. The basis of the schedule is a quadruple round robin tournament resulting in 44 games for each team. In addition, each team plays at home against the Finnish U20 team (national team for players under 20 years of age). Therefore, there are 45 games for each team and a total of 276 games to be scheduled. Distances between the home venues of some of the teams are quite significant. The three most important goals are to generate away tours (C09), to have at least 7 rounds between two games with the same opponents (C19) and to have an equal number of home games at weekends for all teams (C23). For more details, refer to [38].

**TABLE 3.** Real-world benchmark instances

| ID | RR | *n* | Mirrored | Hard constraints | Soft constraints |
|---|---|---|---|---|---|
| **FIN1** | 2* | 14 | No, k=7 | 01,04,07,10,12 | 07,13,14,15,16,19, 22,23,24,26,27 |
| **FIN2** | 2 | 12 | No, k=5 | 01,04,10,12 | 04,13,14,15, 16,19,23,26 |
| **AUS1** | 2 | 18 | Yes | 01,07,15, 19,20,34 | 04 |
| **GER1** | 2 | 18 | Yes | 01,05,07,15, 19,20,34 | 04 |
| **GER2** | 2 | 18 | Yes | 01,05,15,19, 20,31,34,35 | 04 |
| **ARG1** | 2 | 12 | Yes | 01,04,07,10,12, 19,20,23,26 | 13,14,15 |
| **CHI1** | 1 | 20 | Yes | 01,03,04,05,12, 16,24,25,31 | 13,14 |
| **BEL1** | 2 | 18 | Yes | 01,04,07,12,13,14, 15,16,19,20,31 | 04,05,07,24,25, 29,30,32,33,34 |
| **BEL2** | 2 | 18 | Yes | 01,04,07,12,13,14, 15,16,19,20,31 | 04,05,07,24,25, 29,30,32,33,34 |
| **BEL3** | 2 | 18 | Yes | 01,04,07,12,13,14, 15,16,19,20,31 | 04,05,07,24,25, 29,30,32,33,34 |
| **BRA1** | 2 | 20 | Yes | 01,03,07,13,14,15,17,18, 24,25,28,29,31,32,34 | 11 |

The basis of the schedule for the Austrian Soccer Championship (AUS1), the German Soccer Championship (GER1) and the German Handball Championship (GER2) are mirrored double round robin tournaments. The number of teams is 10, 18 and 18, respectively. The most important goal is to reach the minimum number of breaks. The set of rounds teams can play at home or away may be restricted. Specific matches of a home team against an away team can only be carried out in a subset of rounds. In AUS1 some pairs of teams cannot play at home in parallel; thus one of them must play at home in each round. In GER1, subsets of teams cannot play an arbitrary number of home games in parallel in some rounds. In GER2 the number of matches between the six strongest teams is restricted to one per round. For more details, refer to [32].

The major volleyball league in Argentina was composed of 12 teams in 2007/2008, and 11 teams in 2008/2009 and 2009/2010. Although the main interest of the schedule design is the minimization of the global travel distances [36], the instance ARG1 has been adapted to suit the framework introduced in this work. Another feature of this league that has been simplified consists of a paired schedule design. In such a

schedule, the teams' respective matches are grouped into pairs called couples. Each weekend, one couple visits another couple, and the four possible matches between the corresponding teams are played.

The Chilean first division tournament was composed of 20 teams until 2008; since 2009, there have been just 18 teams. There are two tournaments per year: the Opening Tournament and the Closing Tournament. Both competitions consist of a single round robin tournament and then the eight teams with the highest points advance to the playoffs of the championship (until 2008 the teams were divided into groups and the best two teams in each group would advance to the playoffs). The problem has some constraints that are not considered in the test instance, mainly related to the Chilean geography. Chile is a very long and thin country, and for that reason there are constraints related to the trips that teams should or should not make. There are also constraints related to security issues and international competitions. For more details, refer to [33].

The instances BEL1, BEL2, and BEL3 represent the scheduling problem in the highest soccer league in Belgium for the seasons 2006-2007, 2007-2008, and 2008-2009 respectively. This league is played as a mirrored double round robin tournament with 18 teams, involving 306 games that need to be played in 34 rounds. It is imperative that these schedules have the minimal number of 48 breaks (C15), and no team should start or end the league with a break (C12). Furthermore, the two teams that share a stadium cannot play at home in the same round (C07). Apart from that, there are various constraints, originating from Belgacom TV, (the company that broadcasts the league), the police, the clubs and the association itself. For instance, a mayor can forbid a game being played in his or her city in one or more rounds if he/she feels public safety cannot be guaranteed (C04). Clubs may have a number of wishes related to the fairness of the schedule, especially related to the timing of their encounter with strong teams: no team wishes to face all traditionally strong opponents in a row (C33), or likes to host a top game in the summer, when many fans are abroad for holidays (C24). According to Belgacom TV, one way to increase the viewing figures is a schedule where at least one (and preferably two) of four teams that are considered to be top teams plays an away game in each round (C30). The underlying motivation is that a top team's home games are less interesting, since the top team tends to win these games without much effort. Moreover, the top games should be spread over the season (C31). The association itself requests, among other things, that every team receives a top team at home at least once in each half of the season (C32). For more details on the constraints that play a role, and the motivation for these constraints, we refer to [34].

Soccer is the most widely practiced sport in Brazil. The Brazilian national soccer tournament organized every year by the Brazilian Soccer Confederation (CBF) is the most important sporting event in the country. Its major sponsor is TV Globo, the largest media group and television network in Brazil. The most attractive games are those involving teams with more fans and better players, and, consequently, also with larger broadcast shares [19]. Games involving teams from São Paulo and Rio de Janeiro are of special interest for broadcasting through open TV channels due to their corresponding larger revenues from advertising. The competition lasts for seven months (from May to December) and is structured as a compact mirrored double

round robin tournament played by 20 teams. Every team has a home city and some cities host more than one team. There are at most two rounds of games per week: mid-week rounds are played on Wednesdays and Thursdays, while weekend rounds are played on Saturdays and Sundays. Elite teams are those with larger numbers of fans, better records of previous participations in the tournament, and more valuable players. The most important games involve elite teams and, as far as possible, should be played during the weekends, when they can attract larger attendances and TV audiences. The participating teams and the dates available for playing the games change from one year to the next.

Table III shows the above mentioned eleven real-world instances. Most of the instances are mirrored double round robin tournaments. In non-mirrored instances there must be at least k rounds before two teams meet again. The number of teams varies between 12 and 20. The table lists all the hard and soft constraints that are in action for the instances.

## BEST SOLUTIONS TO FIVE REAL-WORLD INSTANCES

In this section we publish the best solutions we have found for some of the real-world instances introduced in Section 5. We invite the sports scheduling community to find solutions to the unsolved instances. We also briefly discuss our solution methods.

The overall goal of the real-world cases is to find a feasible solution that is the most acceptable for the sports league owner. That is, a solution that has no hard constraint violations and that minimizes the weighted sum of the soft constraint violations. The importance of the soft constraints is handled by giving them different weights. The values of the weights are decided based on the negotiations with the league owner and the teams. We refer to [41] for what the values of the weights for the real-world instances were when they were solved. The constraint violations are calculated based on the following:

C01. One violation for each round more than $R$.
C02. One violation for each game more than $m$.
C03. One violation for each home game less than $m_1$ or more than $m_2$.
C04. A violation if team $t$ plays at home in round $r$.
C05. A violation if team $t$ plays away in round $r$.
C06. A violation if team $t$ plays in round $r$.
C07. One violation for each home game less than $m_1$ or more than $m_2$.
C08. One violation for each home game on two consecutive calendar days.
C09. One violation for each away tour less than $m_1$ or more than $m_2$
C10. One violation for each game not preassigned.
C11. One violation for each game assigned.
C12. One violation for each break in round $r$.
C13. One violation for each consecutive home game more than $k$.
C14. One violation for each consecutive away game more than $k$.
C15. One violation for each break more than $k$.
C16. One violation for each break more than $k$.
C17. One violation for each team having an odd number of breaks.

C18. One violation for each team and each break less/more than *k*.
C19. One violation for each round less than *k*.
C20. One violation for each round more than *k*.
C21. One violation for each round more than *k*.
C22. One violation for each consecutive game pair where the home and away teams are the same.
C23. One violation for each home game on $weekday_1$ less than $m_1$ or more than $m_2$ and so on.
C24. One violation for each round less than *r*.
C25. One violation for each round more than *r*.
C26. One violation for each difference more than *k*.
C27. One violation for each difference more than *k*.
C28. One violation for each consecutive game.
C29. One violation for each consecutive game.
C30. One violation for each home game more than *m*.
C31. One violation for each game more than *m*.
C32. One violation for each home game less than $m_1$ or more than $m_2$.
C33. One violation for each home game less than $m_1$ or more than $m_2$.
C34. A violation if the game is not carried out in the given rounds.
C35. A violation if there is no given break for team *t* in the given rounds.
C36. One violation for each multiple of *c* over *c*.

FIN1 and FIN2 were solved by a cooperative local search metaheuristic [22]. BEL1, BEL2, BEL3 were solved using a multiphase decomposition approach ending up with a mixed integer programming model that was solved using CPLEX [34]. Table IV shows the best solutions we have found for FIN and BEL instances. Together with the solutions for the artificial benchmark instances in Section 4, these solutions provide a good starting point for communications between sports scheduling researchers.

**TABLE 4.** Best solutions for five of the real-world benchmark instances.

| ID | RR | *n* | Mirrored | Hard constraints | Soft constraints |
|---|---|---|---|---|---|
| **FIN1** | 2* | 14 | No, k=7 | 01,04,07,10,12 | 07,13,14,15,16,19, 22,23,24,26,27 |
| **FIN2** | 2 | 12 | No, k=5 | 01,04,10,12 | 04,13,14,15, 16,19,23,26 |
| **BEL1** | 2 | 18 | Yes | 01,04,07,12,13,14, 15,16,19,20,31 | 04,05,07,24,25, 29,30,32,33,34 |
| **BEL2** | 2 | 18 | Yes | 01,04,07,12,13,14, 15,16,19,20,31 | 04,05,07,24,25, 29,30,32,33,34 |
| **BEL3** | 2 | 18 | Yes | 01,04,07,12,13,14, 15,16,19,20,31 | 04,05,07,24,25, 29,30,32,33,34 |

We also briefly mention how the other instances were originally solved. AUS1, GER1 and GER2 instances were solved using a combination of a graph coloring algorithm, a semi-greedy algorithm and a truncated branch-and-bound algorithm [32]. ARG1 was solved by a straightforward integer programming model, resorting to ILOG CPLEX for the computational solution of the model [36]. CHI1 was solved

using a two phase approach. First, home-away patterns were created and then the constraint programming model was solved using CPLEX [33]. BRA1 was solved using a multiphase approach similar to that presented in [31]. The key to success is in considering the constraints as early as possible in order to reduce the number of patterns as much as possible, leaving the IP solver with as light a problem possible [20].

## CONCLUSIONS

We defined a framework for scheduling professional sports leagues. A set of artificial and real-world instances were introduced. We have published the best solutions for the artificial instances and five of the real-world instances. The sports scheduling community is invited to challenge our solutions as well as to find solutions to the unsolved instances. The instances are available online.

## REFERENCES


1. K. Easton, G. Nemhauser and M. Trick, "The traveling tournament problem: description and benchmarks", *Proc of the 7th. International Conference on Principles and Practice of Constraint Programming*, Paphos, pp. 580-584 (2001).
2. K. Easton, G. Nemhauser and M. Trick, "Sports scheduling", in *Handbook of Scheduling,* edited by Leung, Florida, USA: CRC Press, 2004, pp 52.1-52.19.
3. J.H. Dinitz, D. Froncek, E.R. Lamken and W.D. Wallis, "Scheduling a tournament", in *Handbook of Combinatorial Designs*, edited by Colbourn and Dinitz, Florida, USA: CRC Press, 2006, pp. 591-606.
4. A. Drexl and S. Knust, "Sports league scheduling: graph- and resource-based models", *Omega* **35**, pp. 465-471 (2007).
5. P. Rasmussen and M. Trick, "Round robin scheduling - A survey", *European Journal of Operational Research* **188**, pp. 617-636 (2008).
6. S. Knust, "Sports Scheduling Bibliography" [Online], Available: http://www.inf.uos.de/knust/sportssched/sportlit_class/, (Last update 31.05.2010).
7. G. Kendall, S. Knust, C.C. Ribeiro and S. Urrutia, "Scheduling in Sports: An annotated bibliography", *Computers and Operations Research* **37**, pp. 1-19 (2010).
8. M. Adriaen, N. Custers and G. Vanden Berghe, "An agent based metaheuristic for the traveling tournament problem", Working Paper, KaHo Sint-Lieven, Gent, Belgium, 2003.
9. H. Crauwels and D. Van Oudheusden, "Ant colony optimization and local improvement", Workshop of Real-Life Applications of Metaheuristics, Antwerp, Belgium, 2003.
10. A. Aggoun and A. Vazacopoulos, "Solving sports scheduling and timetabling problems with constraint programming" in *Economics, Management and Optimization in Sports*, edited by Butenko et al., Springer, 2004, pp. 243-264.
11. R.A. Russell and T.I. Urban, "A constraint programming approach to the multiple-venue sport-scheduling problem", *Computers and Operations Research* **33**, pp. 1895-1906 (2006).
12. D. Costa, "An evolutionary tabu search algorithm and the NHL scheduling problem", *INFOR* **33**, pp. 161-178 (1995).
13. H.-D. Huang, J.T. Yang, S. Shen and J.-T. Horng, "An evolutionary strategy to solve sports scheduling problems", *Proc of the Genetic and Evolutionary Computation Conference*, Los Altos, CA (1999).
14. J. Schönberger, D.C. Mattfeld and H. Kopfer, "Memetic algorithm timetabling for non-commercial sport leagues", *European Journal of Operational Research* **153**, pp. 102-116 (2004).

15. F. Della Croce and D. Oliveri, "Scheduling the Italian Football League: an ILP-based approach", *Computers and Operations Research* **33**, pp. 1963-1974 (2006).
16. T.F. Noronha, C.C. Ribeiro, G. Duran, S. Souyris and A. Weintraub, "A branch-and-cut algorithm for scheduling the highly-constrained Chilean soccer tournament", *Lecture Notes in Computer Science* **3867**, pp. 174-186 (2007).
17. R. Rasmussen and M. Trick, "A Benders approach for the constrained minimum break problem", *European Journal of Operational Research* **177**, pp. 198–213 (2007).
18. D. Briskorn and A. Drexl, "Scheduling sport leagues using branch-and-price", *Journal of the Operational Research Society* **60**, pp. 84-93 (2009).
19. C.C. Ribeiro and S. Urrutia, "Scheduling the Brazilian soccer tournament by integer programming maximising audience shares under fairness constraints", *Proc. of the 2nd International Conference on the Mathematics in Sport*, Groningen, Netherlands (2009).
20. C.C. Ribeiro and S. Urrutia, "Bicriteria integer programming approach for scheduling the Brazilian national soccer tournament", *Proc of the Third International Conference on Management Science and Engineering Management*, Bangkok, pp. 46-49 (2009).
21. E.K. Burke, D. Werra, J.D. de Landa Silva and C. Raess, "Applying heuristic methods to schedule sports competitions on multiple venues", *Proc. of the 5th International Conference on the Practice and Theory of Automated Timetabling*, Pittsburgh, USA, pp. 441-444 (2004).
22. K. Nurmi and J. Kyngäs, "Improving the Schedule of the Finnish Major Ice Hockey League", *Proc. of the 2nd International Conference on the Mathematics in Sport*, Groningen, Netherlands (2009).
23. M. Yavuz, U.H. Inan and A. Figlali, "Fair referee assignments for professional football leagues", *Computers and Operations Research* **35**, pp. 2937-2951 (2008).
24. A. Anagnostopoulos, L. Michel, P. Van Hentenryck and Y. Vergados, "A Simulated Annealing Approach to the Traveling Tournament Problem", *Journal of Scheduling* **9(2)**, pp. 177-193 (2006).
25. A. Lim, B. Rodrigues and X. Zhang, "Scheduling sports competitions at multiple venues – revisited", *European Journal of Operational Research* **175**, pp. 171-186 (2006).
26. M.B. Wright, "Scheduling fixtures for basketball New Zealand", *Computers & Operations Research* **33**, pp. 1875–1893 (2006).
27. J.P. Hamiez and J.K. Hao, "Solving the sports league scheduling problem with tabu search" *Lecture Notes in Artificial Intelligence* **2148**, pp 24–36 (2001).
28. J.H. Lee, Y.H., Lee and Y.H. Lee, "Mathematical modeling and tabu search heuristic for the traveling tournament problem", *Lecture Notes in Computer Science* **3982**, pp 875-884 (2006).
29. L. Di Gasper and A. Schaerf, "A composite-neighborhood tabu search approach to the traveling tournament problem", *Journal of Heuristics* **13**, pp. 189-207 (2007).
30. J.A.M. Schreuder, "Combinatorial aspects of construction of competition Dutch Professional Football Leagues", *Discrete Applied Mathematics* **35**, pp. 301–312 (1992).
31. G. Nemhauser and M. Trick, "Scheduling a major college basketball conference", *Operations Research* **46(1)**, pp. 1-8 (1998).
32. T. Bartsch, A. Drexl and S. Kröger, "Scheduling the professional soccer leagues of Austria and Germany", *Computers and Operations Research* **33(7)**, pp. 1907-1937 (2006).
33. G. Duran, M. Guajardo, J. Miranda, D. Saure, S. Souyris, A. Weintraub and R. Wolf, "Scheduling the Chilean soccer league by integer programming", *Interfaces* **37**, pp. 539-552 (2007).
34. D. Goossens and F.C.R. Spieksma, "Scheduling the Belgian soccer league", *Interfaces* **39(2)**, pp. 109-118 (2009).
35. R. Rasmussen, "Scheduling a triple round robin tournament for the best Danish soccer league", *European Journal of Operational Research* **185(2)**, pp. 795–810 (2008).
36. F. Bonomo, A. Burzyn, A. Cardemil, G. Durán and J. Marenc, "An application of the traveling tournament problem: the Argentine volleyball league", *Proc. of the 7th International Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada (2008).
37. J. Kyngäs and K. Nurmi, "Scheduling the Finnish Major Ice Hockey League", *Proc. of the IEEE Symposium on Computational Intelligence in Scheduling*, Nashville, USA (2009).

38. J. Kyngäs and K. Nurmi, “Scheduling the Finnish 1st Division Ice Hockey League”, *Proc. of the 22nd Florida Artificial Intelligence Research Society Conference*, Florida, USA (2009).
39. D. de Werra, “Scheduling in sports”, in *Studies on graphs and discrete programming,* edited by Amsterdam and Hansen, 1981, pp. 381-395.
40. K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Durán, J. Kyngäs, J. Marenco, C.C. Ribeiro, F. Spieksma, S. Urrutia and R. Wolf, “A Framework for a Highly Constrained Sports Scheduling Problem”, *Lecture Notes in Engineering and Computer Science: Proc of The International MultiConference of Engineers and Computer Scientists* 2010, IMECS 2010, 17-19 March, 2010, Hong Kong, pp. 1991-1997 (2010).
41. K. Nurmi et. al., “Sports Scheduling Problem” [Online], Available: http://www.samk.fi/ssp, (Last update 28.11.2009).
42. M. Henz, T. Müller, S. Thiel and M. van Brandenburg, “Benchmarks and results for round robin tournaments” [Online], Available: http://www.comp.nus.edu.sg/~henz/roundrobin_benchmarks/. (Created 2000).
43. M. Trick, “Challenge Traveling Tournament Instances” [Online], Available: http://mat.gsia.cmu.edu/TOURN, (Last update 4.8.2009).

# Publication 6

E.I. Ásgeirsson, J. Kyngäs, K. Nurmi and M. Stølevik, “A Framework for Implementation-Oriented Staff Scheduling”, in Proc of the 5th Multidisciplinary Int. Scheduling Conf.: Theory and Applications (MISTA), Phoenix, USA, 2011 (submitted for publication).

# A Framework for Implementation-Oriented Staff Scheduling

**Eyjólfur Ingi Ásgeirsson • Jari Kyngäs • Kimmo Nurmi • Martin Stølevik**

**Abstract** This paper presents a general framework for an implementation-oriented, highly constrained staff scheduling problem that is modeled from the requirements of various lines of business and industry. The presented model builds up a solid foundation for wide variety of real-world staff scheduling scenarios. We hope the presented modeling issues assist academics in getting their research results implemented into commercial systems. We define a staff scheduling problem, introduce the necessary terminology, discuss research guidelines and detail the constraints of the problem. A set of artificial and real-world instances derived from the actual problems solved for various companies are presented. We publish the best solutions we have found, and invite the staff scheduling community to challenge our results. We believe that the instances will help researchers to test the implementation value of their solution methods. The instances are available online.

## 1 Introduction

Staff scheduling is a difficult and time-consuming problem that every company or institution that has employees working on shifts or on irregular working days must solve. The staff scheduling problem has a fairly broad definition. Most of the studies focus on assigning employees to shifts, determining working days and rest days or constructing flexible shifts and their starting times. Different variations of the problem are NP-hard and NP-complete [1]-[7] and thus extremely hard to solve. The first mathematical formulation of the problem based on a generalized set covering model was proposed by Dantzig [8]. Good overviews of staff scheduling can be found in [9]-[11].

Nurse rostering [12] is by far the most studied application area in staff scheduling. Other successful application areas include airline crews [13], call centers [14], check-in counters [15], ground crews [16], nursing homes, call centers and airport ground services [17], postal services [18] and transport companies [19]. Recent successful algorithms for staff scheduling include ant colony optimization [20], dynamic programming [21], constraint programming [22], genetic algorithms [23], scatter search [24], hyperheuristics [25], integer programming

Eyjólfur Ingi Ásgeirsson
Reykjavik University, School of Science and Engineering, Iceland
E-mail: eyjo@hr.is

Jari Kyngäs
Satakunta University of Applied Sciences, Finland
E-mail: jari.kyngas@samk.fi

Kimmo Nurmi
Satakunta University of Applied Sciences, Finland
E-mail: cimmo.nurmi@samk.fi

Martin Stølevik
SINTEF ICT, Department of Applied Mathematics, Norway
E-mail: martin.stolevik@sintef.no

[26], metaheuristics [27], simulated annealing [28], tabu search [29] and variable neighborhood search [30].

There are basically four reasons for the current interest in staff scheduling. First, public institutions and private companies around the world have become more aware of the possibilities for decision support technologies, and they no longer want to handle the schedules manually. Second, human resources are one of the most critical and most expensive resources for these organizations. Careful planning can lead to significant improvements in productivity. Third, good schedules are very important for the welfare of the staff. Besides increasing employee satisfaction, effective labor scheduling can also improve customer satisfaction. Finally, new algorithms have been developed to tackle previously intractable problem instances, and, at the same time, computer power has increased to such a level that researchers are able to solve large real-world problems. One further significant benefit of automating the scheduling process is the considerable amount of time saved by the administrative staff involved.

The focus of this paper is to introduce a general framework for an implementation-oriented, highly constrained staff scheduling problem that is modeled from the requirements of various lines of business and industry. The presented model builds up a solid foundation for wide variety of real-world staff scheduling scenarios. In Section 2 we introduce the necessary staff scheduling terminology. Section 3 discusses research guidelines and presents a model for an implementation-oriented staff scheduling problem. The next two sections show that both existing problems and artificial ones can be expressed by the model. In Section 4 we present a set of artificial test instances and in Section 5 we present real-world instances derived from the actual problems solved for various lines of business. We publish the best solutions we have found and invite the staff scheduling community to challenge our results.

## 2 Staff Scheduling Terminology

Staff scheduling consists of assigning *employees* to tasks and shifts over a period of time according to a given timetable. The *planning horizon* is the time interval over which the employees have to be scheduled. Each employee has a *total working time* that he/she has to work during the planning horizon. Furthermore, each employee has *competences* (qualifications and skills) that enable him/her to carry out certain *tasks*. Days are divided into *working days* (days-on) and rest days (*days-off*). A sequence of working days with one shift each day is called a *work stretch*. Each day is divided into *periods* or *timeslots*. A timeslot is the smallest unit of time. The length of a timeslot determines the granularity of the schedule. A *shift* is a contiguous set of working hours and is defined by a day and a starting period on that day along with a *shift length* (the number of occupied periods). Shifts are usually grouped in *shift types* - for example morning day and night shifts. Each shift may be composed of a number of tasks. A shift or a task may require the employee assigned to it to possess one or more competences. A *combined shift* is a collection of non-overlapping shifts; an employee assigned to a combined shift is assigned to all the shifts in the combined shift. If the combined shift is composed of non-contiguous shifts it is called a *split-shift*. A specific sequence of shifts for an employee is called a *stint*. A work schedule for an employee over the planning horizon is called a *roster*. A roster is a combination of shifts and days-off assignments that covers a fixed period of time. Figure 1 clarifies the terminology.

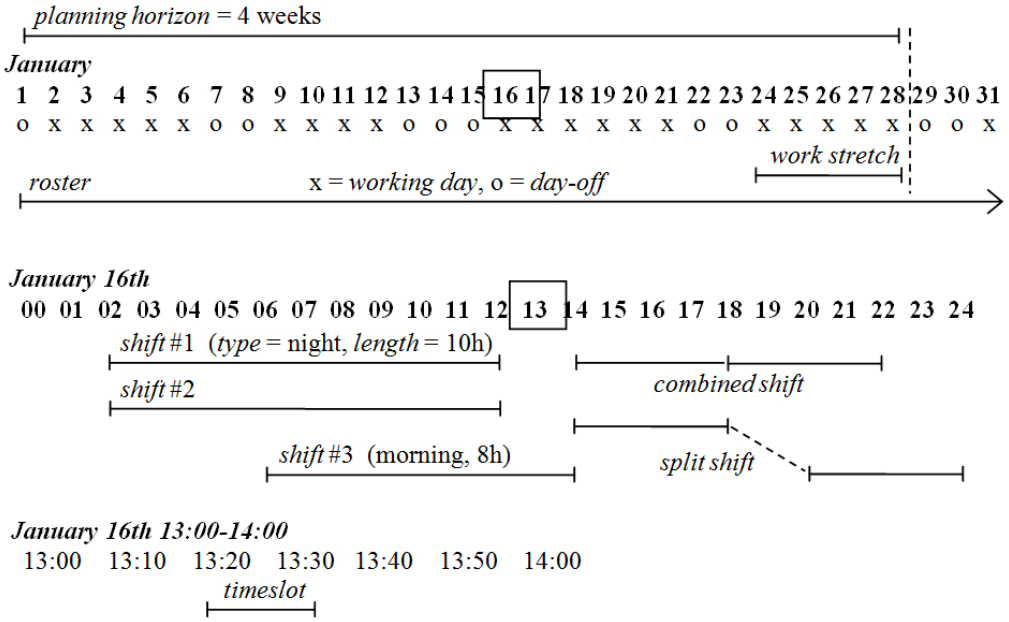


**Figure 1:** Staff scheduling terminology.

*Cyclic* schedules are such that all employees have the same basic schedule but start with a different day. In cyclic scheduling the goal is to find a schedule that is optimal for all employees. *Non-cyclic* schedules are individual schedules. In non-cyclic scheduling the goal is to find rosters that fulfill as many requests as possible. *Continuous* schedules arise in organizations that operate 24 hours a day and seven days a week - otherwise a schedule is called *discontinuous*. *Wrap-around scheduling* means that after the end of the planning horizon, the employees are expected to work the same roster all over again. If wrap-around scheduling is in use, some constraints must be checked outside the planning horizon by extrapolating the employees' rosters. If the process must take into consideration the shifts worked right before the beginning of the planning horizon, we call this *initial condition scheduling* and assume that the initial conditions are input to the problem. For example, decomposing a long time horizon into shorter time horizons which are solved sequentially can be a successful method in practical problems and would create initial conditions that must be handled.

| | Mon | | Tue | | Wed | | Thu | | Fri | | Sat | | Sun | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Early | Late | Early | Late | Early | Late | Early | Late | Early | Late | Early | Late | Early | Late |
| Aaron | 1 | | 1 | | 1 | | 1 | | | 1 | | | 1 | |
| Betty | 3 | | | 3 | | 3 | | 3 | | | 3 | | 3 | |
| Cecil | | 3 | | 1 | | | 2 | | 2 | | 1 | | 2 | |
| David | 2 | | 2 | | | 1 | | | 1 | | 2 | | | 2 |
| Ellie | | | 3 | | 3 | | 3 | | 3 | | | 3 | | 3 |
| Fred | | 2 | | | 2 | | | 2 | | 3 | | 2 | | 1 |
| Gary | | 1 | | 2 | | 2 | | 1 | | 2 | | 1 | | |

**Table 1:** An example and solution of staff scheduling problem.

Table 1 shows a solution for a simple one-week staff scheduling problem with seven employees, two shifts (early and late) in a working day and one of three tasks to be completed within a shift. Moreover, task 1 and 2 cannot be carried out by Betty and Ellie, a late shift

cannot be followed by an early shift on the next day, and each employee should have one day-off.

We classify the real-world staff scheduling process as given in Figure 2. *Demand modeling* is the process of determining the staffing levels, that is, how many employees are needed at different times over some planning horizon. In this presentation, demand modeling also includes determination of planning horizons, competence structures, regulatory requirements and other constraints. Demand modeling requires labor forecasting as well as hiring and budgeting decisions. *Shift design* is the process of determining the shift structure, tasks to be carried out in particular shifts and the competence needed in different shifts. The output of the demand modeling and the shift design is a mathematical model of the staff scheduling problem at hand.

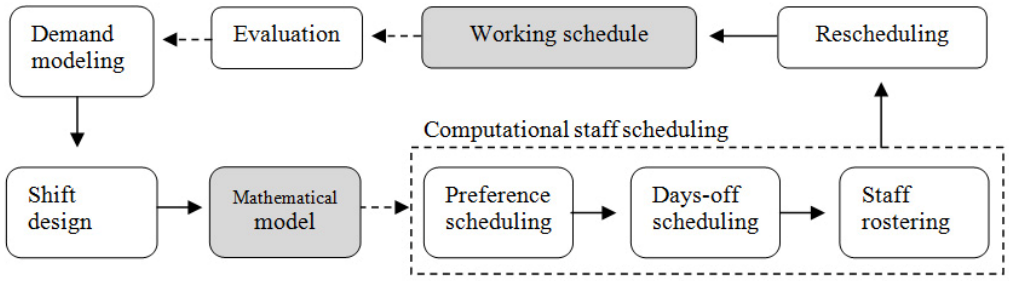


**Figure 2:** The real-world staff scheduling process.

In *preference scheduling*, each employee gives a list of preferences and attempts are made to fulfill them as well as possible. *Pure self-scheduling* refers to preference scheduling in which the employees are entirely responsible for the schedule generation. *Days-off scheduling* deals with the assignment of rest days between working days over a given planning horizon. *Staff rostering*, also referred as shift scheduling, deals with the assignment of employees to shifts. It can also specify the starting time and duration of shifts for a given day even though in most cases they are preassigned in shift design. In other words, days-off scheduling deals with working days and staff rostering deals with the working times of day. When days-off and shifts are scheduled simultaneously, the process is sometimes called *tour scheduling*. The name comes from the fact that we need to specify the hours of the day and days of the week through which each employee must travel. In this paper, the process of scheduling preferences, days-off and shifts is called *computational staff scheduling*. Computational staff scheduling is a key to increased productivity, quality of services, customer satisfaction and employee satisfaction. Other advantages include reduced planning time, reduced payroll expenses and ensured regulatory compliance.

*Rescheduling* deals with ad hoc changes that are necessary due to sick leaves or other no-shows. The changes are usually carried out manually. Finally, participation in *evaluation* ranges from the individual employee, through personnel managers, to executives. A reporting tool should provide performance measures in such a way that the personnel managers can easily evaluate both the realized staffing levels and the staff satisfaction. When necessary, the demand modeling and/or shift design can be reprocessed and focused, and the whole staff scheduling process restarted.

In the demand modeling we have to determine which phases in the computational staff scheduling are scheduled simultaneously. For example, scheduling days-off every tenth week and shifts every second week, enables the staff to plan their free time more conveniently. However, scheduling both at the same time every second week increases the probability of meeting the staffing levels without the need to hire part-time employees. The staff scheduling problems presented in this paper include at least two of the components of the computational staff scheduling. Some of them can be scheduled simultaneously. Alternatively, some of them can be solved first, and then the solution is given as a fixed input to the next component.

## 3 A Model For Implementation-Oriented Staff Scheduling Problems

There are hundreds of staff scheduling solutions commercially available and in widespread use. In that sense, the implementation-oriented staff scheduling approach is already standard practice in industry. However, we believe that there is still some gap between academic and commercial solutions. The commercial products may not include the best academic solutions. We define the implementation-oriented staff scheduling research as a research which raises

1) such modeling issues that have probably precluded academics from getting their research results implemented to commercial advantage, and
2) the trinity between an academic researcher, a problem owner and an industry software vendor.

We have studied a significant number of theoretical and practical staff scheduling papers to realize the most important elements in real-world systems. This, combined with our experience in implementing staff scheduling software, forms the basis to the model. We believe that a considerable number of real-world staff scheduling scenarios can be modeled using the constraints presented in this section.

Most of the staff scheduling cases in which academic researchers have announced that they have closed a contract with a customer concern nurse rostering, see e.g. [27] and [31]-[37]. Hospitals tend to be very open about their operational details, enabling easy cooperation with academics who wish to publish the results of their work. We have experienced that nurse rostering cooperation between a commercial software vendor and academics work. The implemented plug-ins are a great sales pitch for the software vendors when they talk to potential customers.

The interest of the academic staff scheduling community has somewhat shifted to other application areas. The overall interest to real-world staff scheduling has also increased: the number of presented staff scheduling papers has increased in two recognized scheduling conferences, PATAT and MISTA.

According to our experience, the best action plan for real-world staff scheduling research is to cooperate both with a problem owner and with a third-party vendor. In addition, an academic should consider not to work with user interfaces, financial management links, customer reports, help desks, etc. Instead, one should concentrate on modeling issues and algorithmic power.

It is apparent that a profound understanding of the relevant requests and requirements presented by customers is a prerequisite for implementing and solving real-world staff scheduling problems. The implementation should present a wide variety of real-world constraints and be tractable enough to enable the addition of new constraints. It is important to concentrate on the acceptance and satisfaction of both the staff and the personnel managers. Despite the fact that the algorithm should be as robust as possible, no parameter tuning should be expected from the end-users. On the other hand, it should be possible for the end-users to influence different aspects of the algorithm, like weighting between constraints or limit running times, if he/she wishes to.

It should be noted that it is difficult to incorporate the experience and expertise of the personnel managers into a staff scheduling system. Personnel managers often have extremely valuable knowledge, experience and detailed understanding of their specific staffing problem, which will vary from company to company. To formalize this knowledge into constraints is not an easy task. Still, we believe that the model given in this section builds up a solid foundation for staff scheduling scenarios.

The schedules in the model are non-cyclic and can be either continuous or discontinuous. The employees' total working time may vary depending on the employee. Each shift has a fixed start and end time (for example 08:00 – 16:00). Shifts may vary in length and can overlap. Shifts can be classified by shift types which can be used to balance the working times at different times of the day between the employees. A shift is composed of a number of tasks. Each employee has competences that enable her/him to carry out certain tasks. Furthermore,

each employee has preferences over certain tasks, certain shifts and shift types, and certain days-off.

In some cases the most important goal is to minimize understaffing and overstaffing. Low-quality rosters can lead either to an undersupply of employees with a need to hire part-time employees or an oversupply of employees with too much idle time both implicating a loss of business. Furthermore, it is very important to pay attention to employee requests. Kellog and Walczak [38] report that any academic model that does not include some opportunity for preference or self-scheduling will probably not be implemented. For example, nurses use complex decision-making skills when selecting their personal schedules. In addition, our experiences have shown that personal managers can generate remarkably good schedules without optimization tools. Therefore, academic research should explore versatile ways to support preference and self-scheduling.

The overall objective is to meet daily staffing requirements and personal preferences at minimum penalty without violating work contracts and government regulations. We make no strict distinction between hard and soft constraints; that will be given by the instances themselves. The goal is to find a feasible solution that is most acceptable for the problem owner. That is, a solution that has no hard constraint violations and that minimizes the weighted sum of the soft constraint violations. The weights will also be given by the instances themselves. Still, one should bear in mind that an instance is usually just an approximation of practice. In reality, hard constraints can turn out to be soft, if necessary, while giving weights to the soft constraints can be difficult.

We classify the constraints into coverage, regulatory and operational requirements, and operational and personal preferences as given in [39]. The coverage requirement ensures that there are a sufficient number of employees on duty at all times. The regulatory requirements ensure that the employee's work contract and government regulations are respected. The personnel's requests are very important and should be met as far as possible; this leads to greater staff satisfaction and commitment, and reduces staff turnover.

The following constraints can be used as a framework for modeling. An academic researcher can examine these issues together with an industry software vendor:

*Coverage requirement*

- (C1) An employee cannot be assigned to overlapping shifts.
- (C2) A minimum number of employees of particular competences must be guaranteed for each shift or each timeslot
- (C3) A maximum number of employees of particular competences cannot be exceeded for each shift or each timeslot
- (C4) A balanced number of surplus employees must be guaranteed in each working day

*Regulatory requirements*

- (R1) The required number of working days, working hours and days-off within a timeframe must be respected
- (R2) The required number of holidays within a timeframe must be respected
- (R3) The required number of free weekends (both Saturday and Sunday free) within a timeframe must be respected
- (R4) The minimum rest time within a timeframe must be respected
- (R5) The minimum time gap of rest time between two shifts must be respected
- (R6) The number of special shifts (such as union steward duties and training sessions) for particular employees within a timeframe must be respected
- (R7) Employees cannot work consecutively for more than $k$ days (the maximum length of a work stretch)
- (R8) Some employees cannot work on weekends or during specific hours of the day
- (R9) The maximum number of shifts in a single day must be respected

*Operational requirements*

- (O1) An employee can only be assigned to a shift he/she has competence for
- (O2) At least $k$ working days must be assigned between two separate days-off

(O3) An employee cannot be assigned to more than $k$ weekend days within a timeframe
(O4) An employee cannot be assigned to more/less than $k$ shifts of a given type within a timeframe
(O5) An employee assigned to a shift type $t_1$ must not be assigned to a shift type $t_2$ on the following day (certain stints are not allowed)
(O6) An employee must be assigned to a particular shift or on-duty or off-duty on a particular day or during a particular timeslot

*Operational preferences*

(E1) Single days-off should be avoided
(E2) Single working days should be avoided
(E3) The maximum length of consecutive days-off is $k$
(E4) A balanced assignment of single days-off and single working days must be guaranteed between the employees
(E5) A balanced assignment of different shift types must be guaranteed between the employees
(E6) A balanced assignment of different tasks must be guaranteed between the employees
(E7) A balanced assignment of weekdays must be guaranteed between employees
(E8) Assign or avoid a given shift type before or after a free period (days-off, vacation)
(E9) Assign as many wanted, and avoid as many unwanted, stints as possible

*Personal preferences*

(P1) Assign or avoid assigning given employees to the same shifts
(P2) Assign a requested day-on or avoid a requested day-off
(P3) Assign a requested shift or avoid an unwanted shift
(P4) Assign a shift (work) in a requested timeslot or assign no shift (free) to a requested timeslot.

Often, an employee cannot be assigned to more than one shift per day, although we have seen applications where more than one shift per day is allowed. The definition of constraint C1 allows two or more shifts to be assigned provided they do not overlap. Instead of using shifts in C2 and C3, the minimum and maximum number of on-duty staff can be assigned to timeslots. Quite often, a company has more employees working than are needed to cover the minimum number of employees each working day. The surplus employees are used to cover the expected sick leaves and other no-shows (see constraint C4).

Constraints R1-R9 can be different from one employee to another, based on the employees' contracts. The number of working days, working hours and days-off in constraint R1 need not be exact. We can allow minor deviations to working hours over the planning horizon because it might be impossible to meet the number of hours specified in the contract during the planning horizon using the predefined shifts. For example, the number of working hours could be given as a min-max or a $\pm p$ % range to fit this constraint. Shortage or excess is typically handled by the user in the next planning period.

As an example of a work contract and operational requirements and preferences, consider the following. An employee should have exactly nine days-off in every four-week timeframe starting from the beginning of the planning horizon (R1). He/she should have at least one weekend off (R3). The number of working days on weekends cannot exceed four (O3). An employee cannot work consecutively for more than six days (R7). At least two working days must be assigned between two separate days-off (O2). Single days-off and single working days should be avoided (E1 and E2). The maximum length of consecutive days-off is three (E3). Some employees have a work contract that does not involve weekend work (R8).

The working days and shifts in the sample case could be built up using the following rules. The number of working hours per week is 36 hours (R1). At least eleven hours of rest are required during any period of twenty-four consecutive hours (R4). The maximum number of night shifts is three in every four-week timeframe (O4). A night shift has to be followed by at least fourteen hours of rest (R5). An employee assigned to a late or night shift must not be assigned to an early shift the following day (O5). Furthermore, a night/early shift should be

avoided before/after a free period (E8). Note, that the special shifts (R6) can be treated as regular shifts.

Each employee has competences (qualifications and skills) that enable him/her to carry out certain tasks. A shift may require the employee assigned to it to possess one or more competences (O1).

Constraint E9 can be used to model wanted and unwanted shift patterns. The user can input a number of wanted and unwanted stints (patterns). Examples of unwanted stints are D-E-D (Day-Evening-Day) or E-D-E; frequently changing the time of day is stressful. The balanced assignments in constraints E4-E7 can be guaranteed either by using a given min-max range or the range can be expressed as the maximum percentage deviation from the mean.

Constraint P1 can be used in cases when certain employee groups need to work together on the same shifts. Another case could be when a married couple should have the same working hours.

Constraint O6 together with constraints P2, P3 and P4 build up the basis for preference scheduling. The O6 constraints cover any preferences that must be satisfied while the P2, P3 and P4 constraints represent the requests from or wishes of the employees. An efficient and effective preference scheduling relies on having a system whereby the employees can see the requirements, available shifts, an overview of the currently scheduled hours and an easy method of checking the regulations or possible rule violations caused by proposed changes to the schedule. If the schedule created by the employee preferences is close to satisfying the constraints and the coverage demand, it becomes easier to create a schedule that is both feasible with regard to the scheduling constraints and includes most of the requests and preferences. Therefore, it is beneficial for both the employer and the employee to have a good system for preference scheduling. A preference scheduling system that focuses on employee satisfaction should allow each employee a limited number of strict requests of type O6 to schedule days off for things such as doctor's appointments, birthdays and so on, while the number of requests of type P2, P3 and P4 should be enough for the employees to be able to completely schedule all their working hours.

In the case of staff scheduling over several entities (e.g. two departments in a hospital), constraints R1 and O6 may be used to model the days and working time that are "locked" by the other entity (department). For example, Annie is working 70 % for her home department and 30 % for another department. In her home department her 70 % position is modeled by R1 (reduced numbers compared to a 100 % position) and the shifts assigned by the other department is modeled by O6 as pre-assigned days off. It might be necessary to also use some additional constraints: assume for instance that Annie is working a night shift in the other department on Tuesday (starting Monday 23:00 and ending Tuesday at 7:00). Tuesday is then a day off at her home department (O6). She most probably cannot work an evening shift in her home department on Monday. This can be modeled using P4 or O6 by assigning a free period in her home department on Monday evening.

Finally, in cases when days-off are scheduled separately prior to staff rostering, the fixed days-off can be preassigned for the staff rostering phase using constraint O6. The number of holidays within a timeframe can also be preassigned (R2) or the holidays can be treated as days-off.

At the beginning of this section we stated that academics should consider allying with industry software vendors. The other way around also holds. However, Kellog and Walczak [38] give several reasons why developers of commercial products rarely consult academics. For academics, it is usually more important to make publications than to make business. This has a consequence that the scope of the models academics create seems to be relatively small. Although the staff scheduling models developed by academics solve the problem instance at hand, they may fall short of meeting the complex needs of the customers. Academic solutions are often not only computer and platform dependent, but can also use commercial mathematical programming solvers for linear programming, mixed integer programming and constraint programming. However, we believe that the recent advancements made in the staff scheduling research are rapidly closing the research-application gap.

We end this section by summarizing some other important factors that real-world staff scheduling research should consider. Based on our experience, we believe that in addition to being capable of presenting a wide variety of real-world constraints, a staff scheduling system should

- generate just a few, clearly different solutions to choose from,
- allow users to specify the importance of requests and requirements,
- minimize the scheduling time,
- run on any modern computer with any operating system,
- not use third-party mathematical software packages with expensive licensing policies and be integratable with existing industry software.

## 4 Artificial Benchmark Instances

Researchers quite often only solve some special artificial cases or one real-world case. The strength of random test instances is the ability to produce many problems with many different properties. Still, they should be sufficiently simple for each researcher to be able to use them in their test environment. The strength of practical cases is self-explanatory. However, an algorithm performing well on one practical problem may not perform well on another practical problem, which is why we present a collection of test instances for both artificial and real-world cases. We start with artificial cases.

| ID | *n* | *w* | *d* | *h* | $s_{max}$ | $c_{max}$ | E1+E2 | O5+E8 | *r* | E7 | *sol* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 50 | 1 | 0 | 56 | | | | | | | 64 |
| 2 | 100 | 2 | 0 | 112 | | | | | | | 0 |
| 3 | 56 | 4 | 10 | 144 | 6 | 4 | x | | | | 0 |
| 4 | 56 | 4 | 10 | 144 | 6 | 4 | x | x | | | 0 |
| 5 | 56 | 4 | 10 | 144 | 6 | 4 | x | x | 9 | | 0 |
| 6 | 15 | 3 | 7 | 96 | 5 | 3 | x | | | | 0 |
| 7 | 112 | 4 | 8 | 160 | 6 | 3 | x | | | x | 0 |
| 8 | 112 | 4 | 8 | 160 | 6 | 3 | x | x | | x | 0 |
| 9 | 112 | 4 | 8 | 160 | 6 | 3 | x | x | 9 | x | 1 |

**Table 2:** Nine staff scheduling test instances ($n$ = the total number of employees, $w$ = the length of a planning horizon in weeks, $d$ = the number of days-off in the planning horizon (R1), $h$ = the number of working hours per employee within the planning horizon (R1), $s_{max}$ = the maximum length of a work stretch (R7), $c_{max}$ = the maximum length of consecutive days-off (E3), E1+E2 = no single days-off and working days, O5+E8 = forbidden stints (N+M, N+D, N+DO), $r$ = minimum rest time between two working days (R5), E7 = the number of working days between employees on any weekday must not differ by more than 3). The list of the shifts for the test instances can be found in (Ásgeirsson et al. 2010). The solutions (*sol*) have been found using the algorithm described in (Nurmi and Kyngäs, 2011).

Table 2 shows nine staff scheduling test instances. The instances comply with the model presented in Section 3. The number of employees ($n$) varies between 50 and 112, and the length of the planning horizon ($w$) between 1 and 4 weeks. In each instance, the sum of the shift lengths is exactly the same as the total working hours of the employees (see constraint R1 in Section 3). The number of working hours per employee within the planning horizon ($h$) is the same for all employees.

The first two instances do not include days-off. For the other instances, the maximum length of a work stretch (R7) and the maximum length of consecutive days-off (E3) are given. Their minimum lengths are two (E1+E2).

In four instances, an employee assigned to a night shift must not be assigned to a morning or day shift (O5) the following day, or have a day-off (E8) in the following day. In two of these instances, the minimum rest time between two working days is given (R5).

Finally, in the last three instances, the number of working days between employees on any weekday must not differ by more than three (E7). The list of shifts for the test instances can be found in [40]. The challenge is to find a feasible (no hard constraint violations) solution that minimizes the number of soft constraint violations. All but two constraints are hard. The two soft constraints are:

- each employee should have exactly $h$ working hours (two violations for each minute above $h$ and one violation for each minute below $h$) (R1)
- certain stints are forbidden (one violation for each such case) (O5+E8).

We were able to find the optimum zero solution for seven of the test instances; we do not know the optimum value for the other instances. We hope these test instances will help researchers to test the implementation value of their solution methods.

## 5 Real-World Benchmark Instances

The three real-world instances introduced in this section are based on cases we have solved. The understanding of the two issues raised in the beginning of Section 3 has significantly contributed to the quality of the solutions. The instances have been simplified and modified in order to make it easier for researchers to use them in their test environment. The instances are derived from various lines of business and industry in Finland, Iceland and Norway. The instances and list of shifts can be found in [40].

A Finnish bus transit company has 77 bus drivers. The length of the planning horizon is two weeks. The days-off for the planning horizon are given - they have been scheduled separately prior to staff rostering. The fixed days-off can be preassigned for the staff rostering using constraint O6. The number of working days varies between 2 and 10. The average working day is 8 hours. The total number of shifts is 329 per week. The shift lengths vary between 4 hours 15 minutes and 11 hours 14 minutes. An average shift length is 8 hours and 1 minute.

The problem is highly compact. The total number of working days in the planning horizon is only two less than the total number of shifts. Only two Sundays of one employee or one Sunday of two employees will be left empty. Furthermore, the total sum of the working minutes in all the shifts is only sixteen less than the total number of working minutes to be scheduled to the employees.

The challenge is to find a feasible solution that minimizes the number of the following soft constraint violations:

- each employee should have exactly $8n$ working hours, where $n$ is the number of working days for the employee (two violations for each minute above $8n$ and one violation for each minute below $8n$) (R1)
- a night shift should not be assigned before a day-off (ten violations for each such case) (E8).

We believe that mathematical programming techniques on their own, are in many cases too rigid to deal with the multiple and often changing, objectives and goals of the real-world staff scheduling. This first problem was solved using the population-based cooperative local search method described in [41]. We were able to find a solution with 20 soft constraint violations. Remember that the total sum of the working minutes in all the shifts is 16 less than the total number of working minutes to be scheduled to the employees, that is, the theoretical optimum value is 16 soft constraint violations.

An Icelandic call center has 92 employees that must be scheduled in 30 minute timeslots over six weeks. Each day has around 210 possible shifts, ranging from 4 hours up to 11 hours, and the minimum and maximum number of on-duty staff is given for every timeslot

in the planning period. A full-time position requires 38 hours per week, but some employees are working half-time, 19 hours per week. The hard constraints of the problem are: one shift per day at most (R9); the employees must get at least one in every four weekends off (R3) and each employee must get at least 11 consecutive hours of rest in any 24 hour period (R4); the maximum number of consecutive working days is 6 (R7) and the maximum number of working hours in each 24-hour period is 9 hours (R1). Any requests for time off or vacations are treated as hard constraints (P2, P4), but each employee has only a limited number of such requests for each planning period. The employees can also have limitations on working hours included in their contracts, such as no night shifts or no work on weekends (R8). Furthermore, some employees have meetings or other duties that cannot be modified (O6) and some employees are included in the instance but are not part of the rostering process - i.e., their schedule must be kept unchanged (O6).

The soft constraints are: the coverage requirements (C2, C3) that are given for every timeslot should be satisfied; the total number of working hours over the planning period for each employee should be within a given range, usually ± 8 hours (R1). The problem input includes requests for shifts or working hours from the employees (P3, P4), such that most of the employees have requested enough hours to fulfill their working hour requirement. The goal is to find a schedule that fulfills all the hard constraints while minimizing a weighted sum of the soft constraint violations.

This case was solved using multiple-stage greedy local search, with different operations and objectives for each stage [see 17].

A Norwegian hospital ward has 20 employees that must be scheduled over four weeks. There is a fixed manpower plan consisting of nine different shift types, divided into three different categories (Day – six shift types; Evening – two shift types; Night – one shift type). In total, 78 shifts must be scheduled every week. The average weekly working time is 28.5 hours, but the contracts vary between 17.75 hrs/week (50 % part-time position) to 35.5 hrs/week (full time position). The hard constraints of the problem are: one and only one shift per day (R9) (shifts are assigned to the day they start); the employees must work one weekend each, and which one is given by the input (O6); the cover specified in the manpower plan must be covered exactly; the working hours over the planning horizon must not deviate by more than ± 2 % from the contracted amount (i.e., 142 hours. for a full-time position) (R1); some shifts cannot be followed by other shifts (O5) because there must be a minimum time between on-duties (R5). Also, a maximum weekly working time (R1) and a weekly continuous free period (R4) are hard constraints. The challenge is to minimize the weighted penalty associated with the soft constraints: a maximum number of consecutive day shifts (3), evening shifts (2) and night shifts (3), and a maximum five consecutive working days (R7). Furthermore, there should be no single day shifts or night shifts (E2), each employee should have a maximum of five evening shifts (O4) and a min/max of 2/6 night shifts (O4) over the planning horizon. Finally, single days-off should be avoided (E1).

This case has been solved using a hybrid approach between constraint programming and variable neigh-borhood descent in an iterated local search framework (Stølevik et al. 2010). The best solution value found was 4.48.

## 6 Conclusions and Future Work

We defined an implementation-oriented staff scheduling framework. We believe that a considerable number of real-world staff scheduling scenarios can be modeled using the constraints presented in this paper. In addition, we hope the presented modeling issues assist academics in getting their research results implemented into commercial systems. This research has contributed to better systems for our industry partners. A set of artificial and real-world instances derived from the actual problems solved for various companies were presented. We have published the best solutions we have found. We invite the staff scheduling community to challenge our results. We believe that the instances will help researchers to test the implementation value of their solution methods. The instances are available online. We are

currently modeling the instances presented in this paper using the xml-based modeling format introduced and managed by Tim Curtois (Curtois 2010). In addition, we will present the other real-world instances we have been recently working with

**References**


1. Garey M.R. and Johnson D.S., Computers and Intractability. A Guide to the Theory of NP Completeness. Freeman (1979)
2. Bartholdi, J.J., "A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering", *Operations Research* 29, pp 501–510 (1981).
3. Tien J. and Kamiyama A., "On Manpower Scheduling Algorithms", *SIAM Rev.* 24(3), pp. 275–287 (1982).
4. Lau, H. C., "On the Complexity of Manpower Shift Scheduling", *Computers and Operations Research* 23(1), pp. 93-102 (1996).
5. Kragelund L. and Kabel T., Employee Timetabling. An Empirical Study. Master's Thesis, Department of Computer Science, University of Aarhus, Denmark (1998).
6. Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O. and Nourbakhsh, I., "Staff scheduling for inbound call and customer contact centers", *AI Magazine* 23(4), pp 30-40 (2002).
7. Marx, D., "Graph coloring problems and their applications in scheduling", *Periodica Polytechnica Ser. El. Eng*. 48, pp. 5–10 (2004).
8. Dantzig, G.B., "A comment on Edie's traffic delays at toll booths", *Operations Research* 2, pp 339–341 (1954).
9. Alfares, H.K., "Survey, categorization and comparison of recent tour scheduling literature", *Annals of Operations Research* 127, pp. 145-175 (2004).
10. Ernst, A. T., Jiang H., Krishnamoorthy, M., and Sier, D., "Staff scheduling and rostering: A review of applications, methods and models", *European Journal of Operational Research* 153 (1), pp 3-27 (2004).
11. Meisels, A. and Schaerf, A., "Modelling and solving employee timetabling problems", *Annals of Mathematics and Artificial Intelligence* 39, pp. 41-59 (2003).
12. Burke, E.K., De Causmaecker P., Petrovic S. and Vanden Berghe G., "Variable neighborhood search for nurse rostering problems". In Resende and de Sousa, Editors, *Metaheuristics: Computer Decision-Making*, Kluwer, pp. 153–172 (2004).
13. Dowling, D., Krishnamoorthy, M., Mackenzie, H. and Sier, D., "Staff rostering at a large international airport", *Annals of Operations Research* 72, pp 125-147 (1997).
14. Beer, A., Gaertner, J., Musliu, N., Schafhauser, W. and Slany, W., "Scheduling breaks in shift plans for call centers". In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling*, Montréal, Canada (2008).
15. Stolletz, R., "Operational workforce planning for check-in counters at airports". *Transportation Research* Part E 46, pp. 414-425 (2010).
16. Lusby T., Dohn A., Range T. and Larsen J., "Ground Crew Rostering with Work Patterns at a Major European Airlines". In *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling* (PATAT), Belfast, Ireland (2010).
17. Ásgeirsson, E. I., "Bridging the gap between self schedules and feasible schedules in staff scheduling". In *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling* (PATAT), Belfast, Ireland (2010).
18. Bard, J. F., Binici, C. and Desilva, A. H., "Staff Scheduling at the United States Postal Service", *Computers & Operations Research* 30, pp 745-771 (2003).
19. Nurmi K. and Kyngäs J., "Days-off Scheduling for a Bus Transportation Staff", *International Journal of Innovative Computing and Applications*, Inderscience, UK, (to be published 2011).
20. Seçkiner, S.U. and Kurt, M., "Ant colony optimization for the job rotation scheduling problem", *Applied Mathematics and Computation* 201(1-2), pp. 149-160 (2008).

21. Elshafei M. and Alfares H., "A dynamic programming algorithm for days-off scheduling with sequence dependent labor costs", *Journal of Scheduling* 11(2), pp 85-93 (2008).
22. Qu R. and He F., "A hybrid constraint programming approach for nurse rostering problems". In Allen, Ellis, and Petridis, Editors, *Applications and Innovations in Intelligent Systems XVI*, Cambridge, UK, pp. 211-224 (2008).
23. Dean J., "Staff Scheduling by a Genetic Algorithm with a Two-Dimensional Chromosome Structure". In *Proc of the 7th Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada (2008).
24. Burke, E.K., Curtois, T., Qu, R. and Vanden Berghe, G., "A scatter search methodology for the nurse rostering problem", *Journal of the Operational Research Society* (to be published 2010).
25. Remde, S., Cowling, P. I., Dahal, K. P. and Colledge, N., "Exact/Heuristic Hybrids Using rVNS and Hyperheuristics for Workforce Scheduling", In *Proc. of the 7th Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science 4446, Springer, pp. 188-197 (2007).
26. Brunner, J.O., Bard, J.F., and Kolisch, R., "Flexible shift scheduling of physicians", *Health Care Management Science*. 12(3), pp 285-305 (2009).
27. Burke, E., De Causmaecker P., Petrovic S., and Vanden Berghe G., "Metaheuristics for Handling Time Interval Coverage Constraints in Nurse Scheduling", *Applied Artificial Intelligence*, pp 743-766 (2006).
28. Goodale, J. and Thompson, G., "A Comparison of Heuristics for Assigning Individual Employees to Labor Tour Schedules", *Annals of Operations Research* 128(1), pp 47-63 (2004).
29. Musliu, N., "Heuristic Methods for Automatic Rotating Workforce Scheduling", *International Journal of Computational Intelligence Research* 2(4), pp. 309-326 (2006).
30. Burke, E.K., Curtois, T., Post, G.F., Qu, R. and Veltman, B., "A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem", *European Journal of Operational Research* 188(2), pp 330-341 (2008).
31. Bard, J. and Purnomo H., "Hospital-wide reactive scheduling of nurses with preference considerations", *IIE Trans.* 37(7) 589–608 (2005).
32. Beddoe, G.R., Petrovic, S. and Li, J, "A Hybrid Metaheuristic Case-based Reasoning System for Nurse Rostering", *Journal of Scheduling* 12, pp 99–119 (2009).
33. Bilgin, B., De Causmaecker, P., Rossie, B. and Vanden Berghe G., "Local Search Neighbourhoods to Deal with a Novel Nurse Rostering Model". In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling*, Montréal, Canada (2008).
34. Diaz, T., Ferber, D., deSouza C. and Moura A., "Constructing nurse schedules at large hospitals", *Internat. Trans. Oper. Res.* 10(3), pp. 245–265, (2003).
35. Kawanaka, H., Yoshikawa T., Shinogi T. and Tsuruoka S., "Constraints and search efficiency in nurse scheduling problem". In *Proc. Internat. Sympos. Comput. Intelligence Robotics Automation* 1, pp. 312–317 (2003).
36. Meyer auf'm Hofe, H., "Solving rostering tasks by generic methods for constraint optimization". *Internat. J. Foundations Comput. Sci.* 12(5), pp. 671–693 (2001).
37. Van Wezel, W. and Jorna R., "Scheduling in a generic perspective: Knowledge-based decision support by domain analysis and cognitive task analysis", *Internat. J. Expert Systems* 9(3), pp. 357–381 (1996).
38. Kellogg D.L. and Walczak S., "Nurse Scheduling: From Academia to Implementation or Not", *Interfaces* 37(4), pp. 355-369 (2007).
39. Bellanti, F., Carello, G., Della Croce F., and Tadei R., "A greedy-based neighborhood search approach to a nurse rostering problem", *European Journal of Operational Research*, 153(1), pp. 28–40 (2004).
40. Ásgeirsson, E.I., Kyngäs J., Nurmi, K. and Stølevik, M. (Last update ?.?.2011). "Implementation-Oriented Staff Scheduling Problem" [Online], Available: http://www.samk.fi/iossp.

41. Nurmi K. and Kyngäs J., “Improving the Schedule of the Finnish Major Ice Hockey League”. In *Proc. of the 2nd International Conference on the Mathematics in Sport*, Groningen, Netherlands (2009).
42. Stølevik, M, Nordlander T. E., Riise, A. and Frøyseth, H., “An efficient hybrid approach that solves real world nurse rostering problems, using constraint programming and variable neighborhood descent” (preprint submitted to Engineering Applications of Artificial Intelligence 2010).
43. Curtois, T. (Last update August 2010). “Staff Rostering Benchmark Data Sets” [Online]. Available: http://www.cs.nott.ac.uk/~tec/NRP/

# Publication 7

K. Nurmi, J. Kyngäs and G.Post, “Staff Scheduling for Bus Transit Companies”, Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists, Hong Kong, 2011 (in press).

# Staff Scheduling for Bus Transit Companies

K. Nurmi, J. Kyngäs and G. Post

***Abstract*—Good rosters have many benefits for an organization, such as lower costs, more effective utilization of resources and fairer workloads and distribution of shifts. The process of constructing optimized work timetables for the personnel is an extremely demanding task, hence the use of decision support systems for staff scheduling has become increasingly important for both the public sector and private companies. Staff scheduling, preceded by vehicle scheduling and driver scheduling, is the last phase in the bus transit scheduling process. This paper presents a successful way to schedule days-off on a yearly basis and shifts on a monthly basis in one of the Finnish bus transportation companies. The days-off and shifts are scheduled using an algorithm that is a variation of the cooperative local search method. The generated software will be integrated with a third-party vendor product.**

***Index Terms*—Real-World Scheduling, Staff Scheduling, Driver Rostering, Bus Transit Scheduling.**

## I. Introduction

Staff scheduling is a difficult and time consuming problem that every company or institution that has employees working on shifts or on irregular working days must solve. The staff scheduling problem has a fairly broad definition. Most of the studies focus on assigning employees to shifts, determining working days and rest days or constructing flexible shifts and their starting times. Different variations of the problem are NP-hard and NP-complete [1]-[7] and thus extremely hard to solve. The first mathematical formulation of the problem based on a generalized set covering model was proposed by Dantzig [8]. Good overviews of staff scheduling are published by Alfares [9], Ernst et al. [10] and Meisels and Schaerf [11].

Many of the staff scheduling cases concern nurse rostering, see e.g. [12]-[15]. Other successful application areas include airline crews [16], call centers [17], check-in counters [18], ground crews [19], nursing homes [20] and postal services [21]. This paper presents a case in a transportation company.

There are basically four reasons for the increased interest in real-world staff scheduling. First, public institutions and private companies around the world have become more aware of the possibilities of decision support technologies, and they no longer want to handle the schedules manually. Second, human resources are one of the most critical and most expensive resources for these organizations. Careful planning can lead to significant improvements in productivity. Third, good schedules are very important for the welfare of the staff. Besides increasing employee satisfaction, effective labor scheduling can also improve customer satisfaction. Finally, new algorithms have been developed to tackle previously intractable problem instances, and, at the same time, computer power has increased to such a level that researchers are able to solve real-world problems. One further significant benefit of automating the scheduling process is the considerable abount of time saved by the administrative staff involved.

The purpose of this paper is to sequentially solve the days-off scheduling problem and the shift scheduling problem as it occurs in one of the Finnish bus transit companies. In Section II we define the staff scheduling problem and present the necessary terminology. Section III gives an outline of the overall scheduling process in bus transit companies and details the requirements and preferences of the staff scheduling problem. Our solution method is discussed in Section IV. Although there is a clear tendency to use integer and constrained programming models, our algorithm uses a mixture of evolutionary and local search methods. The algorithm is a variation of cooperative local search. In Section V we present and solve a scheduling problem in one of the Finnish bus transportation companies. It will be seen that our approach produces excellent results.

## II. Staff Scheduling

Staff scheduling consists of assigning *employees* to tasks and shifts over a period of time according to a given timetable. The *planning horizon* is the time interval over which the employees have to be scheduled. Each employee has a total working time that he/she has to work during the planning horizon. Furthermore, each employee has competences (qualifications and skills) that enable him/her to carry out certain *tasks*. Days are divided into working days (*days-on*) and *rest days* (days-off). A sequence of working days with one shift each day is called a *work stretch*. Each day is divided into periods or *timeslots*. A timeslot is the smallest unit of time and the length of a timeslot determines the granularity of the schedule. A *shift* is a contiguous set of working hours and is defined by a day and a starting period on that day along with a *shift length* (the number of occupied periods). Shifts are usually grouped in *shift types*, for example morning, day and night shifts. Each shift is composed of a number of tasks. A shift or a task may require the employee assigned to it to possess one or more *competences*. A specific sequence of shifts for

K. Nurmi is with the Satakunta University of Applied Sciences, Pori, Finland (phone: +358 44 710 3371; fax: +358 2 620 3030; e-mail: cimmo.nurmi@samk.fi).

J. Kyngäs is with the Satakunta University of Applied Sciences, Pori, Finland (e-mail: jari.kyngas@samk.fi).

G. Post is with the University of Twente, Department of Applied Mathematics, Faculty of EEMCS, Twente, The Netherlands (e-mail: g.f.post@ewi.utwente.nl).

an employee is called a *stint*. A work schedule for an employee over the planning horizon is called a *roster*. A roster is a combination of shifts and days-off assignments that covers a fixed period of time.

*Cyclic* schedules are such that all employees have the same basic schedule but start with a different day. In cyclic scheduling the goal is to find a schedule that is optimal for all employees. *Non-cyclic* schedules are individual schedules. In non-cyclic scheduling the goal is to find rosters that fulfill the requests of most employees. *Continuous* schedules arise in organizations that operate 24 hours a day and seven days a week, otherwise a schedule is called *discontinuous*.

Table 1 shows a solution for a simple one-week staff scheduling problem with seven employees, two shifts (early and late) in a working day and one of three tasks to be completed within a shift. Moreover, task 1 and 2 cannot be carried out by Bea and Ella, a late shift cannot be followed by an early shift on the next day, and each employee should have one day-off.

Table I
An example of a staff scheduling solution.

| | | Alan | Bea | Cass | Dirk | Ella | Fox | Gary |
|---|---|---|---|---|---|---|---|---|
| Mon | Early | 1 | 3 | | 2 | | | |
| | Late | | | 3 | | | 2 | 1 |
| Tue | Early | 1 | | | 2 | 3 | | |
| | Late | | 3 | 1 | | | | 2 |
| Wed | Early | 1 | | | | 3 | 2 | |
| | Late | | 3 | | 1 | | | 2 |
| Thu | Early | 1 | | 2 | | 3 | | |
| | Late | | 3 | | | | 2 | 1 |
| Fri | Early | | | 2 | 1 | 3 | | |
| | Late | 1 | | | | | 3 | 2 |
| Sat | Early | | 3 | 1 | 2 | | | |
| | Late | | | | | 3 | 2 | 1 |
| Sun | Early | 1 | 3 | 2 | | | | |
| | Late | | | | 2 | 3 | 1 | |

## III. Process and Model

We classify the scheduling process in bus transit companies in six phases, as given in Figure 1. In real-world scheduling scenarios, vehicle scheduling, driver scheduling, days-off scheduling and shift scheduling are all extremely hard combinatorial problems of their own.

*Bus routing* or *Line planning* is a preliminary phase in the development of bus service operations. In public transport the bus routes and their frequencies are defined by the city and the bus companies usually have little opportunity to influence them. Private transport operators create the bus routes based on the business opportunities. In both the public and private sectors, it is completely up to the companies to schedule their fleet of buses, roster the drivers and decide the days-off and working shifts of their drivers. An early reference on bus routing is [22].

*Vehicle scheduling* consists of scheduling a fleet of vehicles to cover the set of bus routes at minimum cost. The problem is solved for each day of the given time horizon separately, and the solution is a set of vehicle schedules. The vehicle scheduling problem was initially introduced by Dantzig and Ramser [23] as the truck dispatching problem. The problem has been proven to be NP-hard [22]. Good overviews of vehicle scheduling can be found in [24] and [25].

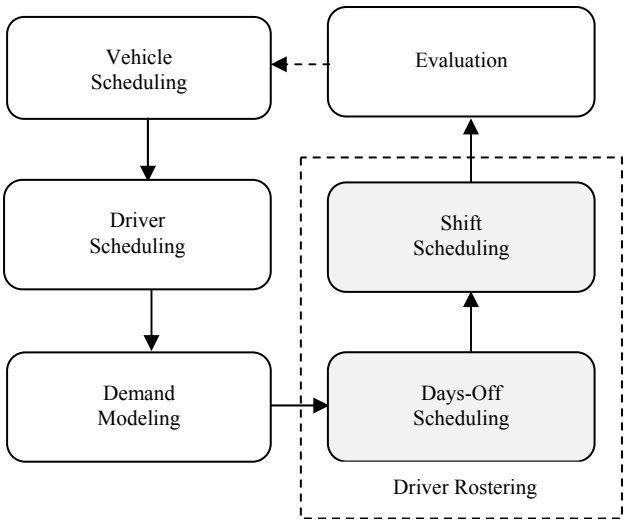


Fig. 1. The scheduling process in bus transit companies.

The goal in *driver scheduling* is to partition the vehicle schedules into operational tasks and to define the sequences of these tasks as shifts. Every task must be assigned to a shift while minimizing the cost in such a way that the daily rules are respected. A task is defined as a sequence of trips on one vehicle without a break that can be performed by a single driver without interruption. The construction of shifts is limited by a maximum total driving time, a maximum number of working hours, a maximum time period spent driving without a break, the number and length of lunch and short breaks in a scheduled time-window, etc. The measure of efficiency may be the total number of shifts used or the total cost in paid hours or a combination of both. Driver scheduling can be modeled as a set covering problem, which is NP-hard [1]. Among the few papers on driver scheduling we mention [26] and [27].

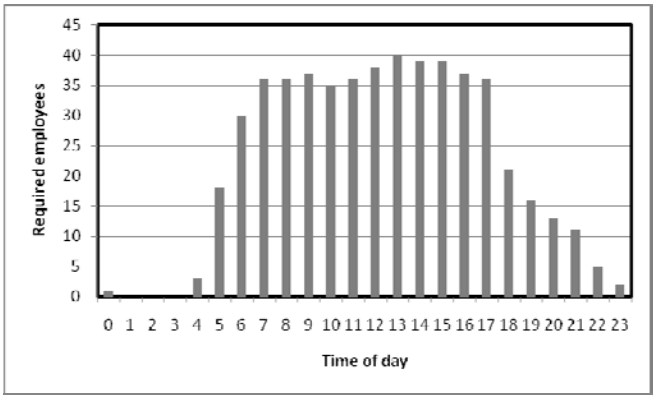


Fig. 2. An example of staffing levels in a bus transit company.

*Demand modeling* is the process of determining the staffing levels, that is, how many drivers are needed at different times over some planning horizon. Demand modeling includes determination of planning horizons, shift structure, competence structures, regulatory requirements and other constraints. Demand modeling requires labor

forecasting as well as hiring and budgeting decisions. The output of the demand modeling is a mathematical model of the staff scheduling problem at hand. In bus transit scheduling, the driver scheduling phase determines the staffing levels. Figure 2 shows an example of staffing levels in a bus transit company.

The staff scheduling phase of the transit scheduling process is called *driver rostering.* The goal is to assign drivers to the constructed shifts over a planning horizon. Driver rostering consists of *days-off scheduling* and *shift scheduling*. Days-off scheduling deals with the assignment of rest days between working days to drivers over a given planning horizon. Shift scheduling deals with the assignment of drivers to shifts. It can also specify the starting time and duration of shifts for a given day. In other words, days-off scheduling deals with working days and shift scheduling deals with the working times of day. When days-off and shifts are scheduled simultaneously, the process is sometimes called *tour scheduling*. For example, scheduling days-off every tenth week and shifts every second week, enables the staff to plan their free time more conveniently. However, scheduling both at the same time every second week increases the probability of meeting the staffing levels without the need to hire part-time drivers.

Finally, participation in *evaluation* ranges from the individual driver through personnel managers to executives. A reporting tool should provide performance measures in such a way that the personnel managers can easily evaluate both the realized staffing levels and the staff satisfaction. When necessary, the vehicle scheduling, driver scheduling and demand modeling can be reprocessed and focused, and the driver rostering process restarted.

It is apparent that a profound understanding of the relevant requests and requirements presented by customers is a prerequisite for implementing a real-world driver rostering problem. The implementation should present a wide variety of real-world constraints and be tractable enough to enable addition of new constraints. We are aware that it is difficult to incorporate the experience and expertise of the personnel managers into a driver rostering system. Personnel managers often have extremely valuable knowledge, experience, and detailed understanding of their specific staffing problem, which will vary from company to company. To formalize this knowledge into constraints is not an easy task. Still, we believe that the model given in this section is applicable in bus transit scheduling scenarios. The model is based on the framework for implementation-oriented staff scheduling given in [28].

The schedules in the model are non-cyclic and can be either continuous or discontinuous. Drivers' total working time may vary depending on the driver. Each shift has a fixed start and end time (for example 07:15 – 16:30). Shifts may vary in length and they overlap. Shifts can be classified by shift types, which can be used to balance the working times at different times of the day between the drivers. Each driver has competences that enable him/her to carry out certain shifts.

In most cases the most important goal is to minimize understaffing and overstaffing. Low-quality rosters can lead either to an undersupply of drivers with a need to hire part-time drivers, or an oversupply of drivers with too much idle time, both implicating a loss of business. The overall objective is to meet daily staffing requirements at minimum penalty without violating work contracts and government regulations.

We next give an outline of the typical constraints of the driver rostering problem. The hard and soft constraints of the problem vary somewhat depending on the problem instance at hand. However, in most cases the hard constraints consist of coverage, regulatory and operational requirements and the soft constraints consist of operational and personal preferences. The coverage requirements ensure that there are a sufficient number of drivers on duty at all times. The regulatory requirements ensure that the driver's work contract and government regulations are respected. The personnel's requests are very important and should be met as far as possible; this leads to greater staff satisfaction and commitment, and reduces staff turnover. A bus transit company can use a mixture of the following requirements and preferences as a framework for its driver rostering generation:

*Coverage requirement*

- (C1) A driver cannot be assigned to overlapping shifts.
- (C2) A minimum number of drivers of particular competences must be guaranteed for each shift or each timeslot
- (C3) A balanced number of surplus drivers must be guaranteed in each working day

*Regulatory requirements*

- (R1) The required number of working days, working hours and days-off within a timeframe must be respected
- (R2) The required number of holidays within a timeframe must be respected
- (R3) The required number of free weekends (both Saturday and Sunday free) within a timeframe must be respected
- (R4) The required number of special shifts (such as union steward duties) for particular drivers within a timeframe must be respected
- (R5) Drivers cannot work consecutively for more than $k$ days (the maximum length of a work stretch)
- (R6) Some drivers cannot work on weekends or during specific hours of the day

*Operational requirements*

- (O1) A driver can only be assigned to a shift he/she has competence for
- (O2) At least $k$ working days must be assigned between two separate days-off
- (O3) A driver cannot be assigned to more than $k$ weekend days within a timeframe
- (O4) A driver assigned to a shift type $t_1$ must not be assigned to a shift type $t_2$ on the following day (certain stints are not allowed)
- (O5) An employee must be assigned to a particular shift or off-duty on a particular day or during a particular timeslot

*Operational preferences*

- (E1) Single days-off should be avoided
- (E2) Single working days should be avoided
- (E3) The maximum length of consecutive days-off is $k$

(E4) A balanced assignment of single days-off and single working days must be guaranteed between the drivers
(E5) A balanced assignment of different shift types must be guaranteed between the drivers
(E6) A balanced assignment of weekdays must be guaranteed between drivers

*Personal preferences*

(P1) Assign or avoid assigning given drivers to the same shifts
(P2) Assign a requested day-on or avoid a requested day-off
(P3) Assign or avoid a given shift type before or after a free period (days-off, vacation).

Generally, a driver cannot be assigned to more than one shift per day. The definition of constraint C1 allows two or more shifts to be assigned provided they do not overlap. Instead of using shifts in C2, the minimum number of on-duty staff can be assigned to timeslots. Quite often a company has more drivers working than is needed to cover the minimum number of drivers each working day. The surplus drivers are used to cover the expected sick leaves and other no-shows (see constraint C3). Constraints R1-R6 can be different from one driver to another, based on the employee's contract.

## IV. The Solution Method

Our driver rostering algorithm is a population-based local search method. As we know, the main difficulty for a local search is

1) To explore promising areas in the search space to a sufficient extent, while at the same time,
2) to avoid staying stuck in these areas too long,
3) to escape from these local optima in a systematic way.

The heart of the algorithm is based on ideas similar to the Lin-Kernighan procedures [29] and ejection chains [30]. The basic hill-climbing step is extended to generate a sequence of moves in one step, leading from one solution candidate to another. Our main heuristic operator is the greedy hill-climbing mutation (GHCM). A recent description of GHCM can be found in [31] and a very detailed description in [32].

The GHCM operator moves an object, $o_1$, from its old position, $p_1$, to a new position, $p_2$, and then moves another object, $o_2$, from position $p_2$ to a new position, $p_3$, and so on, ending up with a sequence of moves. In shift scheduling, each position corresponds to a day, and an object is a shift.

The initial solution is created by setting the shifts to random days. The starting shift (shift 1) for the GHCM operator is selected randomly. The new day to which shift 1 is moved is selected considering all possible days on the time horizon and selecting the day that causes the least increase in the objective function when considering just the relocation cost. Then, shift 2 in the new day is selected such that the removal cost of shift 2 (after adding shift 1) causes the highest decrease in the objective function. Next, a new day is selected for shift 2, and so on. The sequence of moves stops if the last move causes an increase in the objective function value and the value is larger than that of the previous non-improving move, or if the maximum length (set to 10) is reached. In those cases a new starting shift for the GHCM operator is selected.

The population-based method uses a population of solutions in each iteration. Population-based methods are good to escape from local optima. Our algorithm is similar to the cooperative local search introduced by Preux and Talbi [33]. In the cooperative local search scheme, each individual carries out its own local search, in our case the GHCM operator. When the operator gets stuck it asks for the cooperation of the population in order to find a direction to move in and continues the search from another point in the solution space. The results in each individual may be different at different times and this encourages diversity within the population. We use a population size of 20.

Our algorithm introduces a mechanism to avoid staying stuck in a not-so-promising search area for too long. After being stuck for a given number of iterations, we shuffle the current solution, that is, we allow worse solutions to replace better ones in the current population. We use the five simple shuffling operators described in [34].

The reproduction operation of the algorithm is, to a certain extent, based on the steady-state reproduction: the new schedule replaces the old one if it has a better or equal objective function value. Furthermore, the least fit is replaced with the best one when $n$ better schedules have been found, where $n$ is the size of the population. The pseudo code of the algorithm is given in Figure 3.

```
Set the time limit t, no_change limit m and the population size n
Generate initial population of schedules by randomly
assigning shifts to days
Set no_change = 0 and better_found = 0
WHILE elapsed-time < t
   REPEAT n times
      Select a schedule S by using a marriage selection
      Apply GHCM to S to get a new schedule S'
      Calculate the change Δ in fitness value
      IF Δ < = 0 THEN
         Replace S with S'
         IF Δ < 0 THEN
            better_found = better_found + 1
            no_change = 0
         ENDIF
      ELSE
         no_change = no_change + 1
      ENDIF
   ENDREPEAT
   IF better_found > n THEN
      Replace the worst schedule with the best schedule
      Set better_found = 0
   ENDIF
   IF no_change > m THEN
      Apply shuffling operators
      Set no_change = 0
   ENDIF
   Update the dynamic weights of the hard constraints (ADAGEN)
ENDWHILE
Choose the best schedule from the population
```

Fig. 3. The pseudo code of the driver rostering algorithm.

The traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation

scores to the hard constraint values to get a single value to be optimized. In our research we use the ADAGEN method [32] which assigns dynamic weights to the hard constraints.

A very detailed description of the algorithm is given in [32]. The parameters of the algorithm (maximum length of the mutation sequence and population size) are those that were found to work best in [35].

Many researchers have argued that the quality of the solution (strongly) depends on the initial solution. However, the initial solution does not have to be good. On the contrary, it has been argued that very good initial solutions are hard to improve on with the method at hand. We note here, that our test runs have shown that our algorithm works best with a random initial population.

## V. The Problem in a Finnish Transit Company

In our previous studies we have successfully scheduled the Finnish major ice hockey league [36] and the Finnish 1st division ice hockey league [34]. When the CEO of Turku Transport Services Ltd. heard that we had scheduled these leagues, he contacted us. Turku Transport Services Ltd. is a bus transportation company in the City of Turku. They currently have 58 full-time, eight part-time and four retired-but-still-active bus drivers. Two part-time drivers count as one full-time driver.

Prior to the year 2010, rosters for bus drivers were produced by a cyclic shift scheduling software that was quite out-of-date. The current number of drivers and the current way of doing business had outgrown the limits of the current system. The old system had led to an oversupply of bus drivers with too much idle time. Furthermore, the old system also required far too much manual work. For example, the days-off scheduling was done completely manually. The CEO was interested in a more effective and sophisticated way of constructing days-off and shifts and avoiding overstaffing.

The driver rostering problem in the company is non-cyclic and discontinuous and must be divided into two separate phases: days-off scheduling and shift scheduling. The drivers must know their days-off a year beforehand while the working times in a day must be known only four weeks beforehand.

### *The days-off scheduling problem*

The company has $n$ (62) full-time-equivalent drivers. Over the planning horizon of one year (13 timeframes with a timeframe of four weeks totaling 364 days), we must find $n$ such sequences of days-on and days-off that satisfy the following hard constraints (see Section IV):

- A minimum number of drivers (see Table II) must be guaranteed for each working day (C2).
- A balanced number of surplus drivers must be guaranteed in each working day (C3).
- Each driver should have nine days-off in every four-week timeframe (R1).
- Drivers cannot work consecutively for more than six days (R5).
- Six drivers cannot work on weekends (R6).
- At least two working days must be assigned between two separate days-off (O2).
- The number of days-off per weekday between drivers should not differ by more than 10% (E6).
- The same sequence within each of the three driver groups with three drivers must be guaranteed (P1).

Moreover, the following soft constraints are considered:

- Single days-off should be minimized (two violations for each single day-off) (E1).
- Single working days should be avoided (one violation for each single working day) (E2).
- The maximum length of consecutive days-off is three (ten violations for each day more than three) (E3).
- The number of single days-off and single working days between drivers should not differ by more than 25% (five violations for each unit of percentage over 25) (E4).

Table II
The minimum number of drivers needed per day of week.

| | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|---|---|---|---|---|---|---|---|
| Drivers | 47 | 47 | 47 | 47 | 50 | 27 | 10 |

The company has more drivers working than is needed to cover the minimum number of drivers each working day. The surplus drivers are used to cover the expected sick days.

In addition, retired-but-still-active drivers can be used if necessary. The average number of surplus drivers is calculated as follows. Per week we need 275 drivers (see Table II), hence the number of man-days needed over the planning horizon is $52 \times 275 = 14{,}300$. The drivers work $28 - 9 = 19$ days in four weeks, hence the man-days available is $62 \times 19 \times 13 = 15{,}314$. Hence we have an absolute surplus of 1014, and a surplus of 7.1% per man-day needed.

The average surplus of drivers on the different days of the week is calculated proportionally, giving 3.3, 3.3, 3.3, 3.3, 3.6, 1.9 and 0.7. The second hard constraint (C3) can now be rephrased as "On Mondays, Tuesdays, Wednesdays, Thursdays and Fridays either three or four, on Saturdays either one or two, and on Sundays either zero or one surplus drivers must be guaranteed".

Table III
The constant weights for the soft constraints and the maximum values for the hard constraints (minimum is 1).

| E1 | E2 | E3 | E4 | C3 | R1 | R5 | R6 | O2 | E6 | P1 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 10 | 5 | 50 | 50 | 50 | 50 | 20 | 30 | 70 |

The objective is to find a solution that has no hard constraint violations and minimizes the weighted sum of the soft constraint violations. We use the adaptive penalty method for multi-objective optimization (see Section IV). The importance of the soft constraints is handled by giving them different constant weights. Hard constraint weights are

dynamically calculated according to the ADAGEN method. The values of the weights, given in Table III, were decided based on the information from the company. Note that hard constraint C1 is not listed in the table because the algorithm uses the exact number of employees as given in Table II.

We generated ten days-off schedules and selected the best one. This schedule has no hard constraint violations and 443 single days-off and 513 single working days. Thus, an average driver has a single working day in every seventh week, and a single working day in every sixth week. The number of single days-off and single working days between drivers does not differ by more than 25%. Every days-off sequence is at most three. As a result, the weighted sum of the soft constraint violations is $2 \times 443 + 1 \times 513 = 1399$. The algorithm was run on an Intel Core 2 Extreme QX9775 PC with a 3.2GHz processor and 4GB of random access memory running 64bit Windows Vista Business Edition. The best solution was found in 16 hours of computer time. The time may appear to be long. However, the point here is not to find a solution fast enough and with sufficient quality, but to find a solution of the best quality. The planning horizon is one year, so it is worth running the algorithm overnight.

*The shift scheduling problem*

A driver roster is a combination of shifts and days-off assignments that covers a fixed period of time. In our case, the days-off are scheduled separately prior to shift scheduling.

In Section III we stated that the driver scheduling phase partitions the vehicle schedules into pieces of work and defines the sequences of these pieces of work as shifts. A piece of work was defined as a sequence of trips on one vehicle without a break that can be performed by a single driver without interruption. A shift includes several different bus routes. The shift length is determined by the time needed to complete all the routes. The length varies between 4 hours 55 minutes and 9 hours 23 minutes. The company uses six different shift types: early, late, night, school, peak and service. Mondays, Tuesdays, Wednesdays and Thursdays have an equal shift structure. Fridays, Saturdays and Sundays each have unique shift structures. The total number of shifts is 275 per week (see Table II).

The solution of the days-off scheduling problem is the input to the shift scheduling problem. Over the planning horizon of four weeks, given a days-off schedule and a set of predetermined shifts, the problem is to find a roster for each driver that satisfies the following hard constraints:

- A driver can only be assigned to a shift he/she has competence for (O1).
- A driver assigned to a late or night shift must not be assigned to an early shift on the following day (O4).
- An employee must be assigned to an off-duty shift (day-off) on a particular day (O5).

Moreover, the following soft constraints are considered:

- The number of working hours for each driver should be 153 (one violation for each partial hour below or above 153) (R1)
- The number of different shift types between drivers should not differ by more than 25% (five violations for each unit of percentage over 25) (E5)
- Assign an early shift before a day-off or a vacation and a late or night shift after a day-off or a vacation (one violation for each such assignment) (P3).

The objective again is to find a solution that has no hard constraint violations and minimizes the weighted sum of the soft constraint violations. The rosters with less than 153 hours are considered as bad as the rosters with more than 153 hours. For some companies the most important goal could be to reduce idle time for employees. The values of the three hard constraint weights are all between one and five.

We solved the problem using exactly the same algorithm and the same computer as for the days-off scheduling problem. We generated ten shift schedules for the first four-week planning horizon and selected the best one. The best solution was found in four hours of computer time. The time is in line with the fact that the planning horizon is four weeks. The best schedule has no hard constraint violations. Sixteen drivers have exactly 153 working hours, 31 drivers have a maximum of 152 working hours and 15 drivers have a maximum of 151 working hours. Thus, an average driver has about one hour idle time, less than 1%. Note that the total sum of the working hours in all the shifts was 61 less than the total number of working hours to be scheduled to the drivers in the days-off scheduling.

The number of different shift types between drivers did not differ by more than 25%. Two other than early shifts were assigned before a day-off or a vacation and no late or night shifts after a day-off or a vacation. As a result, the weighted sum of the soft constraint violations is $1 \times 31 + 2 \times 15 + 1 \times 2 = 63$.

The company is very satisfied with our results. In their opinion the days-off scheduling algorithm could run for days because it is generated only once a year. As explained before, they are interested in the best possible value of the objective function, not in how fast this is reached. The shift scheduling algorithm can be run overnight. It is perfectly reasonable to run it for up to 15 hours, again because it is generated only once per month.

The company listed a number of advantages and savings as a result of switching to the developed system: the reduced time for developing rosters, the fairer and more balanced days-off and shifts, and the reduced idle time for bus drivers. The system also produces rosters that are more stable with regard to small changes, both in the operational environment and in the employees' work contracts. One further significant benefit is that the system can be used as a planning tool for future scenarios. The company actually demonstrated the effect of using different planning horizons and different vacation patterns.

The company wants to integrate our algorithms into their operational systems. However, our system does not include an adequate user interface nor financial management links and customer reports. For this reason, we contacted the

major bus transit software vendor in Finland. This vendor had no optimization in their product and was very interested in cooperating with us. After four months of further coding and fine-tuning, the generated driver rostering software was ready to be integrated into the third-party vendor product. The software

- Allows users to specify the importance of requests and requirements.
- Minimizes the scheduling time required by users.
- Runs on any modern desktop computer.
- Does not use third-party mathematical software packages with expensive licensing policies.
- Generates just a few solutions to choose from. Generates clearly different solutions to choose from.

The company, the third-party vendor and we were all very pleased with how the project ended. We do not have to work on user interfaces, financial management links, customer reports, help desks etc. Instead, we can concentrate on our core competence: development of algorithms that are useful in real-world applications.

## VI. Conclusions and Future Work

We scheduled the staff in a Finnish bus transit company. Our algorithm found feasible and acceptable solutions to their days-off scheduling and shift scheduling problems. The generated software will be integrated into a third-party vendor product.

Our direction for future research will be to solve the vehicle scheduling and driver scheduling problems that precede the driver rostering problem solved in this paper.

## References


[1] Garey M.R. and Johnson D.S., Computers and Intractability. A Guide to the Theory of NP-Completeness, Freeman, 1979.

[2] Bartholdi, J.J., A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering, Operations Research 29, 501–510, 1981.

[3] Tien J. and Kamiyama A., On Manpower Scheduling Algorithms. In SIAM Rev. 24 (3), 275–287, 1982.

[4] Lau, H. C., On the Complexity of Manpower Shift Scheduling, Computers and Operations Research 23(1), 93-102, 1996.

[5] Kragelund L. and Kabel T., Employee Timetabling. An Empirical Study, Master's Thesis, Department of Computer Science, University of Aarhus, Denmark, 1998.

[6] Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O. and Nourbakhsh, I, Staff scheduling for inbound call and customer contact centers, AI Magazine 23(4), 30-40, 2002.

[7] Marx, D., Graph coloring problems and their applications in scheduling, Periodica Polytechnica Ser. El. Eng. 48, 5–10, 2004.

[8] Dantzig, G.B., A comment on Edie's traffic delays at toll booths, Operations Research 2, 339–341, 1954.

[9] Alfares, H.K., Survey, categorization and comparison of recent tour scheduling literature, Annals of Operations Research 127, 145-175, 2004.

[10] Ernst, A. T., Jiang H., Krishnamoorthy, M., and Sier, D., Staff scheduling and rostering: A review of applications, methods and models, European Journal of Operational Research 153 (1), 3-27, 2004.

[11] Meisels, A. and Schaerf, A. 2003, Modelling and solving employee timetabling problems, Annals of Mathematics and Artificial Intelligence 39, 41-59, 2003.

[12] Bard, J. and H. Purnomo, Hospital-wide reactive scheduling of nurses with preference considerations, IIE Trans. 37(7), 589–608, 2005.

[13] Beddoe, G.R., Petrovic, S. and Li, J., A Hybrid Metaheuristic Case-based Reasoning System for Nurse Rostering, Journal of Scheduling 12, 99–119, 2009.

[14] Bilgin, B., De Causmaecker, P., Rossie, B. and Vanden Berghe G., Local Search Neighbourhoods to Deal with a Novel Nurse Rostering Model. In Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling, Montréal, Canada, 2008.

[15] Burke, E., P. De Causmaecker, S. Petrovic, and G.Vanden Berghe, Metaheuristics for Handling Time Interval Coverage Constraints in Nurse Scheduling, Applied Artificial Intelligence, 743-766, 2006.

[16] Dowling, D., Krishnamoorthy, M., Mackenzie, H. and Sier, D., Staff rostering at a large international airport", Annals of Operations Research 72, 125-147, 1997.

[17] Beer, A., Gaertner, J., Musliu, N., Schafhauser, W. and Slany, W., Scheduling breaks in shift plans for call centers. In Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling, Montréal, Canada, 2008.

[18] Stolletz, R., Operational workforce planning for check-in counters at airports. Transportation Research Part E 46, 414-425, 2010.

[19] Lusby, R., Dohn, A., Range, T. and Larsen, J., Ground Crew Rostering with Work Patterns at a Major European Airlines. In Proc of the 8th Conference on the Practice and Theory of Automated Timetabling (PATAT), Belfast, Ireland, 2010.

[20] Ásgeirsson, E.I., Bridging the gap between self schedules and feasible schedules in staff scheduling. In Proc of the 8th Conference on the Practice and Theory of Automated Timetabling (PATAT), Belfast, Ireland, 2010.

[21] Bard, J. F., Binici, C. and Desilva, A. H., Staff Scheduling at the United States Postal Service, Computers & Operations Research 30, 745-771. 2003.

[22] Bertossi, A.A., Carraresi, P., Gallo, G., On some matching problems arising in vehicle scheduling models, Networks 17, 271–281, 1987.

[23] Dantzig, G.B., and Ramser, J.H., The truck dispatching problem, Management Science 6, 81-91, 1959.

[24] Desaulniers, G and Hickman, M., Public transit. Handbooks in Operations Research and Management Science, Transportation, Vol. 14, G. Laporte and C. Barnhart (eds), Elsevier, Amsterdam, 69-127, 2007.

[25] Bunte, S., & Kliewer, N., An overview on vehicle scheduling models. Journal of Public Transport 1(4), 299-317, 2009.

[26] Wren, A; Fores, S; Kwan, A; Kwan, R; Parker, M; Proll, L., A flexible system for scheduling drivers, Journal of Scheduling 6, 437-455, 2003.

[27] Li J. and Kwan R.S.K., A Self-Adjusting Algorithm for Driver Scheduling, Journal of Heuristics 11 (4), 351-367, 2005.

[28] Ásgeirsson, E.I., Kyngäs, J., Nurmi, K. and Stølevik, M., A Framework for Implementation-Oriented Staff Scheduling, In Proc of the 21st International Conference on Automated Planning and Scheduling (ICAPS), Freiburg, Germany, 2011. Submitted to publication.

[29] Lin, S. and Kernighan B. W., An effective heuristic for the traveling salesman problem, Operations Research 21, 498–516, 1973.

[30] Glover, F., New ejection chain and alternating path methods for traveling salesman problems. In Computer Science and Operations Research: New Developments in Their Interfaces, edited by Sharda, Balci and Zenios, Elsevier, 449–509, 1992.

[31] Nurmi, K. and Kyngäs, J., Days-off Scheduling for a Bus Transportation Staff. In Proc of the 4th International Conference on Bioinspired Optimization Methods and their Applications, Ljubljana, Slovenia, 2010.

[32] Nurmi, K., Genetic Algorithms for Timetabling and Traveling Salesman Problems, Ph.D. dissertation, Dept. of Applied Math., University of Turku, Finland, 1998. Available: http://www.bit.spt.fi/cimmo.nurmi/

[33] Preux, P. and Talbi, E-G., Towards Hybrid Evolutionary Algorithms International Transactions in Operational Research 6, 557-570, 1999.

[34] Kyngäs, J. and Nurmi, K., Scheduling the Finnish 1st Division Ice Hockey League. In Proc. of the 22nd Florida Artificial Intelligence Research Society Conference, Florida, USA, 2009.

[35] Nurmi, K. and Kyngäs, J., A Framework for School Timetabling Problem. In Proc. of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications, Paris, France, 2007.

[36] Kyngäs, J. and Nurmi, K., Scheduling the Finnish Major Ice Hockey League. In Proc. of the IEEE Symposium on Computational Intelligence in Scheduling, Nashville, USA, 2009.

# Turku Centre for Computer Science
# TUCS Dissertations

**108.** **Tero Säntt**i, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
**109.** **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
**110.** **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
**111.** **Camilla J. Hollanti**, Order-Theoretic Methods for Space-Time Coding: Symmetric and Asymmetric Designs
**112.** **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
**113.** **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
**114.** **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
**115.** **Petri Salmela**, On Communication and Conjugacy of Rational Languages and the Fixed Point Method
**116.** **Siamak Taati**, Conservation Laws in Cellular Automata
**117.** **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
**118.** **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
**119.** **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
**120.** **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
**121.** **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
**122.** **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
**123.** **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
**124.** **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
**125.** **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
**126.** **Tuomo Saarni**, Segmental Durations of Speech
**127.** **Johannes Eriksson**, Tool-Supported Invariant-Based Programming
**128.** **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
**129.** **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
**130.** **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
**131.** **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
**132.** **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
**133.** **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
**134.** **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
**135.** **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
**136.** **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
**137.** **Ville Junnila**, On Identifying and Locating-Dominating Codes
**138.** **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
**139.** **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
**140.** **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

**University of Turku**
*Faculty of Mathematics and Natural Sciences*
- Department of Information Technology
- Department of Mathematics

*Turku School of Economics*
- Institute of Information Systems Science

**Åbo Akademi University**
*Division for Natural Sciences and Technology*
- Department of Information Technologies

ISBN 978-952-12-2634-2
ISSN 1239-1883

Jari Kyngäs

# Solving Challenging Real-World Scheduling Problems

Jari Kyngäs

Solving Challenging Real-World Scheduling Problems