Proceedings of the Twenty-Fourth Innovative Appications of Artificial Intelligence Conference

Using AI Local Search to Improve an OR Optimizer

F. Morgado, **R. L. Saldanha**, **J. Roussado**, **L. Albino**, **E. Morgado**, and **J. P. Martins**

SISCOG - Sistemas Cognitivos, SA
Campo Grande 378 - 3
1700-097 Lisboa, Portugal
www.siscog.pt

Abstract

One of the key issues for transportation companies is to produce an optimal plan for the work of crew members. Crew planning consists of a sequence of phases, the first two corresponding to planning *duties* (sequences of trips to be done by crew members from their home base to their home base) and planning *rosters* (sequences of duties and rest days to be followed by crew members during a certain number of weeks).

Both duty and roster planning are subject to a large number of constraints. Duty planning is constrained by intra-duty constraints and roster planning by inter-duty constraints. Since inter-duty constraints relate how duties can be combined into a roster, it is desirable that some of these constraints be transposed into the duty planning phase, as *additional constraints*, to guarantee that the duties produced in the first phase are "rosterable" in the second phase.

Both Artificial Intelligence (AI) and Operations Research (OR) have addressed duty planning, but for very large scale problems, OR has been far more successful due to its global vision of the problem. This paper discusses the use of AI local search to improve an OR-based duty planning optimizer that uses additional constraints.

Introduction

Railroads are going through a golden expansion period. The pressure for environmental-friendly transportation associated to the growing needs for mass transport and the possibility of competing with airlines in medium-range routes are generating a railroad expansion that had not been seen for almost one century (UNIFE 2010). Productivity concerns are leading railroad companies to introduce new management styles to improve the results of the business. This trend is similar to what was felt in other areas of transportation where the benefits of IT solutions have been extensively explored. In particular, tools are being searched to improve the use of resources. Railroads deal with three types of resources: track, rolling stock, and crew.

Crew planning is fundamental for the smooth operation and profitability of a railroad company. Crew planning is subject to global and individual constraints and preferences that mutually interact, making it a very complex process. In order to control its complexity, most railroad companies divide crew planning in different phases, resulting in a set of smaller, and therefore less complex, subproblems. A full description of crew planning phases can be found in (Martins and Morgado 2010).

This paper addresses issues related with the first two phases: *duty planning*, where crew duties are generated by sequencing trips, and *roster planning*, where rosters are generated by combining duties into weekly rosters. These phases are critical for producing efficient global solutions. Although done separately, it is desired that they work in a cooperative fashion, i.e., that the duties provided by the first can be fit in the rosters generated by the second, to reduce the number of iterations between the two phases. One way to achieve this is by using in duty planning a transposition of some of the constraints used in roster planning. In this way, duties produced are constrained by some roster constraints and, thus, are more likely to fit in the rosters. Since roster constraints are not part of duty planning, they are called *additional constraints*. This paper addresses duty planning with additional constraints.

Problem Description

Duty planning consists of distributing a set of trip tasks by a set of duties, at the lowest possible cost. A *duty* (Figure 1) is a sequence of tasks that can be performed by a crew member during a working period, beginning and ending at the home base, and during which a set of operational and labor constraints (*intra-duty constraints*) are satisfied. A *base* is an administrative center to which the crew members belong and, usually, is located close to where they live. A *labor constraint* results from the labor rules, agreed by negotiation between the railroad company and the unions. An *operational constraint* results from requirements inherent to the transportation operation itself.

Once duties are built, they are distributed by a set of rosters, also at the lowest possible cost. A *roster* (Figure 2) is a week-sequencing pattern that allows crew work variation over a period of several weeks for a group of crew members of a base. Planning duties in a roster, *roster planning*, is subjected to *inter-duty constraints*.

Optimizing crew rosters is important for the global optimization of crew, since rosters are repeated along the whole calendar period. In the traditional way, the duties produced

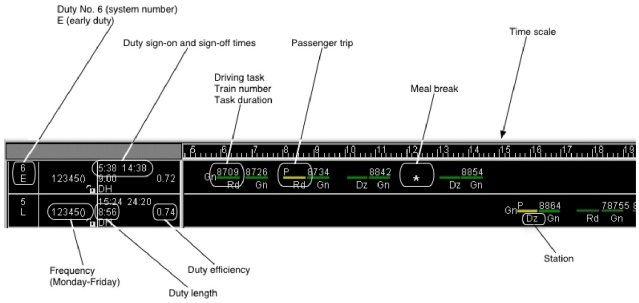

Figure 1: Example of a duty.

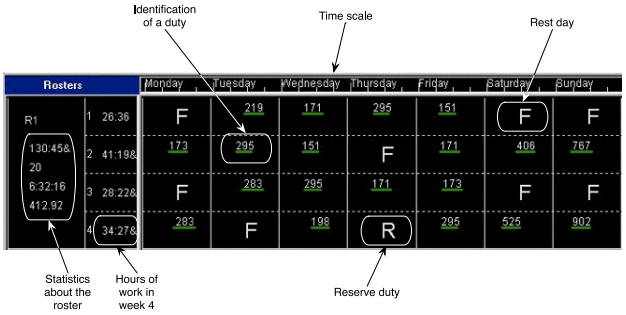

Figure 2: Example of a 4-week roster.

in the first phase are planned only taking into account intra-duty constraints. Therefore, there is no guarantee that they will fit the roster, due to inter-duty constraints, requiring iterations between the two phases. To reduce this limitation, in duty planning we can use a set of additional constraints that will increase the probability of the resulting set of duties to fit in the rosters. These constraints result from the non-exact transposition of some inter-duty constraints into the crew duty planning phase. By being validated during the duty planning phase, they reduce, or even avoid, the amount of iteration required. It is obvious that roster planning still has to deal with all inter-duty constraints, but, once most of the additional constraints are satisfied, it is now easier to find a solution. Additional constraints thus take a central role in duty planning. Solving them is a real need of transportation companies and an important contribution to their economy. There are three types of additional constraints:

- *Capacity constraints*. The capacity of a base is the number of crew members available on that base. We cannot allocate more duties to the base than its capacity. A capacity constraint is formulated as:

$$\sum_{j:\ t_j \in T_b} x_j \leq K_b \qquad (\forall b \in B) \tag{1}$$

  where B is the set of bases, T_b is the set of potential duties of base b, t_j is a duty of T_b, x_j is 1 or 0 whether t_j belongs or not to the solution, and K_b, a scalar, is the capacity of base b.

- *Average constraints*. Average constraints limit the average duration of duties of a given base. This controls the maximum amount of working hours assigned to each base and enables flexibility in creating long and short duties, provided that the average is respected. An average constraint is formulated as:

$$\frac{\sum_{j:\ t_j \in T_b} d_j x_j}{\sum_{j:\ t_j \in T_b} x_j} \leq d \qquad (\forall b \in B) \tag{2}$$

  where B, T_b, x_j, and t_j are as above, d_j is the duration of duty t_j, and d is a duration scalar.

- *Percentage constraints*. Percentage constraints allow a fair distribution of duties among bases, according to certain desirable standards. For example, it is undesirable that all dangerous or night duties are only assigned to a base, while other bases have only day duties and safe duties. Similarly, there may be highly desirable duties, which will also have to be evenly distributed among bases. Defining percentage constraints with a lower and an upper bound enables a fair distribution of the *sweet and sour* (Abbink et al. 2005) among bases. A percentage constraint, for a certain duty type p, is formulated as:

$$p_l \leq \frac{\sum_{j:\ t_j \in T_b} p_j x_j}{\sum_{j:\ t_j \in T_b} x_j} \leq p_u \qquad (\forall b \in B) \tag{3}$$

  where B, T_b, x_j, and t_j are as above, p_j is a binary function that tells whether t_j is of type p, and $p_l, p_u \in [0,1]$, are the lower and upper bounds, respectively.

The starting point of the work reported in this paper was an OR-based duty optimizer, CREWS IP-solver (Abbink et al. 2011), that although producing very good results, due to the existence of additional constraints, sometimes was "trapped" in local optima. CREWS IP-solver is part of a software product, CREWS, that addresses all phases of crew planning and management. CREWS is being used by several european railroad companies to plan and manage their crew members. The main focus of CREWS IP-solver is the global optimization of the solution and this overall goal sometimes leads the optimizer to fail in handling the additional constraints.

CREWS IP-solver was extended with the use of an AI-based algorithm, called AC-solver, whose main focus is the satisfaction of the additional constraints. CREWS IP-solver successively invokes the AC-solver, whenever trapped in a local optimum.

Application Description

The problem being addressed is NP-hard (Karp 1972) and the check of additional constraints can only be done in a complete solution. For these reasons, a constructive search method cannot be used. We decided to use local search as a complement to CREWS IP-solver. The local search algorithm receives a complete solution whenever IP-solver gets

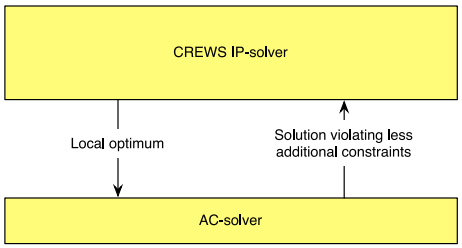

Figure 3: CREWS IP-solver architecture with AC-solver.

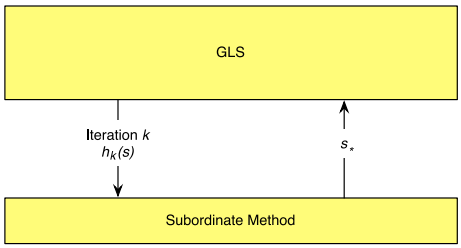

Figure 4: AC-solver architecture with subordinate method.

stuck in a local optimum and provides IP-solver with a modified version of the solution that satisfies more additional constraints than the solution received (Figure 3). It was decided that a limited amount of time should be allocated for the AC-solver to deal with the additional constraints. The reason for this decision is associated to the fact that the main goal of CREWS IP-solver is to optimize duties, by itself a computationally heavy task, and by the fact that it is admissible that some of the additional constraints be violated.

We used Guided Local Search (GLS) (Voudouris 1997) (Voudouris and Tsang 2003) for the following reasons: (1) it can use problem-specific knowledge, focusing on what really matters; (2) it can escape local optima, directing the search for promising regions of the solution space; (3) it traverses the solution space very intelligently, which is important when the allocated time is short. GLS sits on top of another local search, the *subordinate method*. GLS defines a set of features for candidate solutions, penalizing some of them whenever the subordinate method, which uses an objective function augmented through these penalties, gets stuck in a local optimum. The novelty of GLS comes from the way it selects features to penalize, allowing to guide the search effort for more promising regions of the solution space. CREWS IP-solver already included a local search strategy, based on first-choice, but it did not satisfactory handle additional constraints.

To apply GLS, we have to define a set of features of a candidate solution, each one with an associated cost and penalty. A *feature* is a property of a solution that is relevant for its evaluation, allowing to distinguish the solution from other solutions with different features. A feature has a direct impact on the objective function. The *cost* of a feature should be proportional to its contribution for the objective function, more important features must have higher cost than less important ones. The *penalty* of a feature starts at zero and can be increased by one unit when a local optimum is achieved.

Instead of using the usual objective function g, GLS invokes successively the subordinate method with an augmented objective function h (Figure 4), which considers the addition of dynamic penalties to function g while iterating. The augmented function h is defined as:

$$h_k(s) = g(s) + \lambda \sum_{i \in F} p_{i,k} l_i(s) \tag{4}$$

where k is the GLS iteration, s is a candidate solution, λ is a parameter of GLS, i ranges over features F, $p_{i,k}$ is the dynamic penalty for feature i (initially 0) and l_i is a binary function that tells whether s exhibits feature i.

When the subordinate method is trapped in a local optimum, some features are penalized, thus changing the objective function. The features selected are those with the highest value of the following function:

$$\mathrm{u}_{i,k}(s_*) = l_i(s_*) \frac{c_i}{(1 + p_{i,k})} \tag{5}$$

where s_* is a candidate solution that corresponds to a local optimum, c_i is the cost of the feature i, and $l_i(s_*)$ and $p_{i,k}$ are as above. The higher the cost of a feature and the less often it has been penalized, the higher the utility of penalizing it. The intention is to penalize "unfavorable features" when the subordinate method settles in a local optimum. Features with higher cost affect more the overall solution cost. In turn, features that have been penalized fewer times are those in which the search has not focused as much as with others and should gain importance. When a feature is penalized, its penalty value is increased by one.

We now describe (1) the original objective function g, (2) the features that enable the objective function g to be transformed in the augmented objective function h, and (3) the subordinate method.

Objective Function The objective function considers as a good solution one that minimizes, as much as possible, the number and cost of duties, the quantity of violation, and the intensity of violations present in the solution (Morgado 2011).

This function defines a set of violation intensity levels — qualitative measures of violation — where, the higher the level, the higher the violation intensity. The objective function g, for a candidate solution s, is defined as:

$$g(s) = \sum_j c_j + \sum_r q_r(s) + \sum_i \sum_{n=1}^{n(i)} p_{n,r(i)} l_i(s) \tag{6}$$

The objective function g is composed of three terms:

- *Efficiency* ($\sum_j c_j$, where j varies over the duties in s), focus on *efficiency of the solution*. The computed cost of duty j, c_j, consists of a fixed cost (which is identical for all duties), to minimize the number of duties, and a variable cost (corresponding to the actual cost of duty j), to minimize the overall cost of the duties.

- *QuantViol* ($\sum_r q_r(s)$, where r ranges over the set of additional constraints), focus on *satisfaction of the solution*. Each $q_r(s)$ linearly reflects the quantity of violation of constraint r present in s. The aim is to minimize the global quantity of violation of solution s.

- *FixedPen* ($\sum_i \sum_{n=1}^{n(i)} p_{n,r(i)} l_i(s)$, where i ranges over the set of additional constraints and, for each additional constraint, over the violation intensity levels), also focus on *satisfaction of the solution*. In order to know which constraint and violation intensity level corresponds to i, we define the functions $r(i)$ and $n(i)$ that, respectively, provide this information. In this term, $p_{n,r(i)}$ is the fixed penalty corresponding to the intensity level n of violation of the constraint $r(i)$ and $l_i(s)$ indicates whether the solution s violates constraint $r(i)$ with violation intensity $n(i)$. Each constraint violation has associated an intensity level, e.g., from 1 (low) to 5 (high). For each type of constraint, we must specify the limits of violation for all intensity levels defined. For example, in case of an average constraint, we may define a violation until 30 seconds as one of intensity level 1, from 30 to 60 seconds as one of intensity level 2, and so on. For each constraint violation, this term adds to $g(s)$ the penalty corresponding to the intensity level of the violation, as well as the penalties corresponding to all intensity levels below it. The goal is to guide the search process for solutions with violations with lower and lower intensity levels.

The objective function aims at successively reducing the intensity of violations. The idea is to guide the search process in reaching solutions with violations that are easier and easier to solve, until eventually will be solved.

Features Definition Since a GLS feature corresponds to a property that influences the evaluation of a solution, it makes sense that an additional constraint be associated with as many features as the number of intensity levels that are defined. Thus, in Equation 4, i ranges over the set of additional constraints and, for each additional constraint, over the violation intensity levels defined. The variables $r(i)$, $n(i)$, and $l_i(s)$ are defined in the same way as in Equation 6, and c_i and $p_{i,k}$ are, respectively, the cost and the dynamic penalty of the violation intensity level $n(i)$ of the constraint $r(i)$.

In order to highlight the importance of reducing the intensity level of the violations, the costs c_i of the features corresponding to higher violation intensity levels are higher than those of lower violation intensity levels. Thus, features corresponding to higher violation intensity levels are more penalized by GLS and, therefore, GLS puts a bigger effort in solving these.

GLS augmented objective function h used in this application is slightly different from what is defined in Equation 4[1]:

$$h_k(s) = g(s) + \lambda \sum_{i \in F} \sum_{n=1}^{n(i)} p_{n,r(i),k} l_i(s) \quad (7)$$

where $p_{n,r(i),k}$ is the dynamic penalty corresponding to the violation intensity level n of the constraint $r(i)$.

Subordinate Method The subordinate method is hill-climbing. To move from solution to solution it is necessary to define a way of generating the neighborhood of a solution s. In the context of this problem, it is a set of potential alternatives, each one obtained through a change of duties in the solution s.

Since in this problem the neighborhoods contain a large number of solutions, it is essential to cut the number of neighbors, thus not to consider all possible combinations of exchanges. So, we adopted a two-step generation method:

1. We remove a duty from s that (1) maximizes the value of the sum of all terms in the objective function h except g's *Efficiency* term and (2) has not been selected for removal yet;
2. We insert in s as many duties as needed to cover the tasks that turned out to be uncovered. To limit the combinatorial explosion, for each uncovered task, a duty is chosen that both covers it and minimizes the value of the same terms of h. These duties are selected from a set of potential duties (see next section).

In order to minimize the number of duties in the solution, after this change of duties has taken place, redundant duties, i.e., duties that cover only tasks that are already covered by other duties, are removed.

Since, for large problems, the time required to generate all solutions of the neighborhood is still too long, solutions are generated one by one, until a given acceptance criterion is satisfied. In this version of hill-climbing, the acceptance criterion accepts, for the next current solution, the solution s', from the first $p\%$ solutions in the neighborhood[2] of the current solution s, that verifies $h(s') < h(s)$ but also lowers the value of $h(s')$. If such solution does not exist, the algorithm continues to search beyond the first $p\%$ solutions and accepts the first solution that verifies the condition $h(s') < h(s)$. When no solution in the neighborhood of s verifies this condition, the algorithm terminates.

Application Development and Deployment

In order to validate GLS, we compared its results with those of the IP-solver with its traditional local search (here called *Classic*) and other well-known local strategies, namely, simulated annealing (SA) (Kirkpatrick, C. D. Gelatt, and Vecchi 1983), tabu search (TS) (Glover and McMillan 1986), and hill-climbing (HC). HC is the same as defined as GLS

[1] While GLS iterates, in order to prevent penalties corresponding to features of lower violation intensity levels from exceeding the value of higher violation intensity levels, instead of adding to $h(s)$ only the penalty corresponding to the violation intensity level of a given constraint, we also add the penalties corresponding to all violation intensity levels of the same constraint below it. This reasoning is also used in function g, so that it is consistent with h.

[2] Since $p\%$ is a percentage and we must guarantee a number of neighbor solutions greater than zero, then $p \in]0, 100]$. In fact, since the number of neighbors of a solution must be an integer number, we may not generate exactly $p\%$ of solutions in the neighborhood, but a number of solutions that correspond to the immediately lower integer number (i.e., $\lfloor p\% \text{ of the neighborhood} \rfloor$).

	Tasks	Duties	Constraints	Potential duties
Average	10,926	1,067	720	73,359
Minimum	9,602	935	487	3,500
Maximum	10,957	1,074	1,249	81,379
Std. dev.	202	20	351	14,226

Table 1: Global analysis of the tested problems features.

subordinate method, except that it uses the objective function g (Equation 6). SA and TS also use the objective function g.

A set of 45 problems provided by NS (the Dutch Railways), was chosen as benchmark. Table 1 shows a global analysis of their features, number of tasks to be covered (Tasks), number of duties in the initial solution (Duties), number of additional constraints (Constraints), and number of potential duties (Potential duties)[3]. The potential duties are a subset of the universe of duties that may be built from the given tasks and that are considered suitable for building a solution, out of which a minimal subset that covers the tasks must be chosen.

The average number of violations in the initial solution is around 10, reaching a maximum value of 17. These are the violations that must be solved by GLS. These are also the violations that could not be solved by IP-solver (before calling GLS), and so they are the most difficult to solve. It is important to stress that along the search process GLS has to deal with hundreds of different constraints and that it is guaranteed that a valid solution exists for each of the 45 problems.

GLS is just one of the many computations performed by IP-solver and it must take up very little time from the total time IP-solver has to solve a problem. Each strategy has a time limit of one hour, although it may terminate sooner.

To compare the strategies, we use the values of their objective function g and its term *Efficiency*, as well as the value of an attribute, called *Satisfaction*, which corresponds to the sum of the terms *QuantViol* and *FixedPen* of function g. In addition, we use attribute *NumViol* that corresponds to the number of violations in the solution. Since IP-solver only accepts local search solutions with no violations, we also use attribute *ValidSol* that indicates whether the strategy has produced a valid solution.

The strategy that produces the best results is GLS (Figure 5) by the average of the following attributes of the solutions obtained for the 45 problems:

- g. GLS is the strategy that produces the best results, followed by HC, TS, Classic, and SA. While all strategies seem to improve satisfaction relatively to the initial solution, we must not forget that the latest violations are the most difficult to solve.

  A surprise was the fact that HC was better than SA, since unlike HC that is attracted to the closest local optimum, SA has the ability to escape from it and find better local optima, eventually, the global optimum. One possible reason has to do with the scale of the problems tested and the short time that SA has to do its job. It is known that SA is better than HC, when given enough time. Since HC, in this benchmark, takes an average of 22 minutes to get to the closest local optimum, SA would require much longer than one hour to perform better.

  Another surprise was the fact that HC was better than TS. The justification is also the limited time allocated to the search. In just one hour, TS does not have time to properly use its intensification and diversification mechanisms.

- *Efficiency* and *Satisfaction*. GLS is the strategy that achieves better results for *Satisfaction*, although at the expense of a loss in *Efficiency* (in relation to the two best strategies on this attribute, HC and TS). This loss is normal, since it is the objective function g that lets to change efficiency for satisfaction. By penalizing the violation intensity levels, GLS focus better on the resolution of violations than other strategies and, thus, it is the one that, in the short time available, can have greater impact on satisfaction. The search is guided to regions of the solution space where the cost of *Satisfaction* is lower.

  HC and TS achieve results that are closer to those of GLS.

[3]The number of potential duties shown in Table 1 are those considered in the benchmark. The number of potential duties considered by IP-solver, during the whole search process, is in the order of millions.

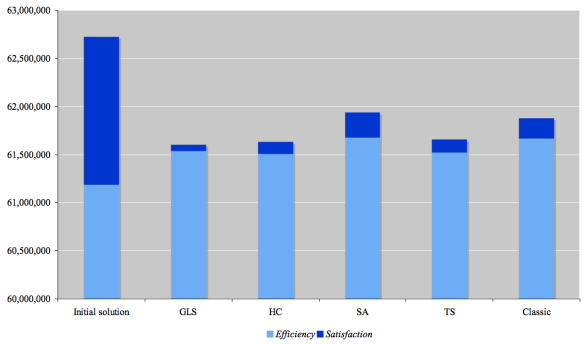

Figure 5: Comparison of the average of g, divided into its attributes *Efficiency* and *Satisfaction*, for the initial solution and the solutions obtained by each strategy, in the 45 problems.

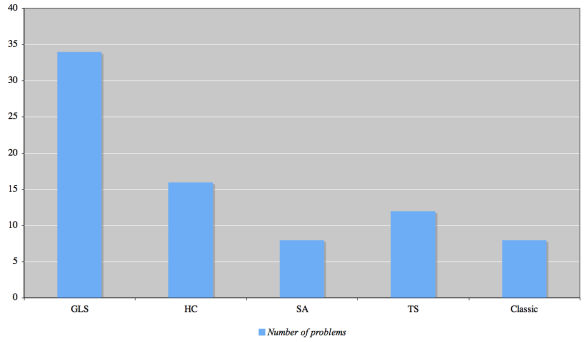

Figure 6: Comparison of strategies for the ability to obtain valid solutions.

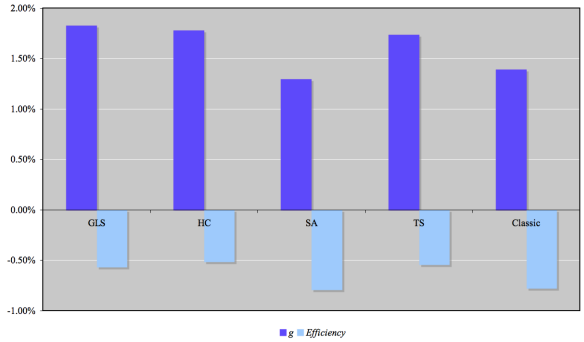

Figure 7: Comparison of gains in g and *Efficiency* of strategies in relation to the initial solution.

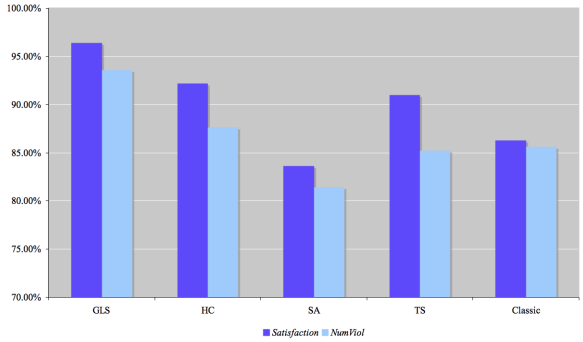

Figure 8: Comparison of gains in *Satisfaction* and *NumViol* of strategies in relation to the initial solution.

They get efficiency results slightly better, although the satisfaction results are clearly worse. In the case of HC, it is obvious that GLS is better since, until the first local optimum is reached, it behaves as HC, and, then, has mechanisms to improve the solution. In the case of TS, although it has intensification and diversification mechanisms to effectively exploit other regions of the solution space based on the elite (best solutions found so far), it is not guided to search regions with solutions with better satisfaction that have not yet been visited. The lack of focus on what is relevant, does not allow, in the short time given, to reach these regions. On the contrary, the results of SA and Classic were clearly worse, both in satisfaction and efficiency. SA has no knowledge of the problem, and is only guided to escape local optima. For this reason, the time limit is even more critical than in the case of TS. Classic has an objective function that does not consider violation intensity levels, but just the number of violations in the solution. So, during the process of solving violations, the remaining ones are more and more difficult to solve. Besides, it is a variant of hill-climbing less capable than HC.

By analyzing *ValidSol* (Figure 6), we conclude that GLS is the strategy more capable of obtaining valid solutions. It does so in 34 out of 45 problems, while the second best strategy, HC, achieves this only in 16 (less than half) and Classic only in 8. Since IP-solver only accepts valid solutions, GLS is the best strategy to use, obtaining valid solutions in about 75% of the problems.

Figures 7 and 8 allow to compare the average percentage gains of different strategies concerning the objective function g, as well as the attributes *Efficiency*, *Satisfaction*, and *NumViol*. In relation to Classic, GLS has gains of 0.44% in the objective function and 10.15% in satisfaction, and yet still manages to gain 0.21% in efficiency, i.e., loses less efficiency. Concerning *NumViol*, GLS has gains of 8%.

Besides comparing the strategies regarding the quality of the solutions produced, it is also important to compare them regarding the efficiency of the search process. Concerning average execution time, Classic and HC are the fastest, since they are both variants of hill-climbing. They used 15 (with a maximum of 48) and 22 (with a maximum of 41) minutes, respectively. Next, GLS uses 41 minutes (with a maximum of one hour). Finally, the SA and TS strategies used all the allocated time (one hour).

In face of the results obtained, in 2011, GLS was incorporated in CREWS IP-solver, replacing the existing local search strategy.

Application Use and Payoff

This section addresses the gains obtained by the use of GLS by the Dutch Railways (NS), so far, the only company where GLS was used. It should be kept in mind that since GLS was introduced in a core product used by many other companies, Norwegian Railways, Danish State Railways, Suburban Trains of Copenhagen, Finnish State Railways, London Underground, and Lisbon Metro, the gains here described (regarding the use of about 5,000 crew members) can be extrapolated to over 25,000 crew members that are daily planned with the use of CREWS.

NS, Dutch Railways, consists of several divisions, one of them, NS Reizigers (NSR), responsible for passenger trains, operates 4,700 trains per day on a network with 1,500 miles, using 123 electric locomotives, 1,883 EMU cars, 86 DMU cars, and 833 coaches, together with a rail staff of 5,200 (2,500 engineers and 2,700 conductors). Every day, more than one million people travel by train in the Netherlands over the busiest railway transportation network in the world.

CREWS_NS, the customization of CREWS to NSR, has been in continuous use by NSR since 1998. Over the years several improvements have been introduced, as summarized by the Head of Logistics Department of NSR (Fabries 2008): "*End users (planners) really appreciate the system. We reduced the amount of planners significantly. Next to that we were able to save millions of Euros by making more efficient schedules. This is not only the result of the system, but without such a system this would not have been possible*".

The first gains were obtained in the early years of deployment, only using manual mode, i.e., without using an optimizer (see next section): all tedious work of checking which trains were covered by duties was handled by the system; all labor rules were automatically checked whenever

duties were constructed or when there was a change in the timetable; furthermore, since labor rules were in the system, detailed knowledge of these rules was not required of planners. Differences in rules and regulations for engineers and conductors started to be seen as insignificant. As a consequence, the same persons could plan duties for both personnel groups and planning units could be integrated.

The initial use of the system originated 10% decrease in the number of planners. Another interesting aspect of the start of deployment with the manual mode was that there was no need for change in the working methods because the system provided the freedom to use the old working methods. This was a direct consequence of using AI technology (see next section). Thus, such a change, that would demand a lot of managerial attention, was avoided.

In 2009, after ten years of full production, a benchmark was performed using the 1999 data and rules to measure the overall improvement that the several versions of the optimizers had introduced. The results of the tests have shown that the new optimizer gives about 6% efficiency improvement with respect to the manually created plan. In the mean time, the several changes to rules and constraints that have been introduced (and were not considered in the benchmark) give additional efficiency. It is also important to say that the generation of duties subject to the new rules and constraints would have been impossible without the use of an optimizer. The number of planners was reduced by 60% from the initial count in 1999.

After the development of the AC-solver and of the testing presented in previous section, IP-solver using GLS was tested against IP Solver using Classic with full NSR data, starting from an empty initial solution. Both were capable of producing a solution with no violations, but IP Solver using GLS produced 0.75% fewer duties (920 against 927). At first, a gain of 0.75% may seem too low. However, it should be kept in mind that this gain was obtained over the results of an OR optimizer that had been fine-tuned over several years.

According to information supplied by NSR for previous gains achieved by IP-solver, gains of 1% in the solution correspond to 3 million euros savings per year. So, with these results, IP-solver with GLS produces savings in the crew plan of NSR in the order of 2.25 million euros (about US$ 2.89 million) per year. In face of these results, the 0.75% gains are quite significant.

We witnessed that local search is called many times by IP Solver in order to solve violations. The resulting solutions are then used as "seeds" to find new promising solutions. By solving the violations slowly, successively reducing the intensity levels of the violations until they eventually disappear, while keeping the number of duties low, GLS guides IP Solver to better regions of the solution space, allowing it to produce better solutions than Classic.

Use of AI Technology

This paper reports recent results of a long-term development project in crew planning and management involving AI technology. It is associated with the development of a product, called CREWS, that plans and manages crew members in a transportation company. The initial ideas for developing CREWS started in 1986 with a prototype for TAP/Air Portugal. With this prototype, SISCOG started the development of CREWS, initially using Explorer LISP Machines and KEE as the underlying tool. In the late 1990s CREWS was ported to Windows / Intel, abandoning KEE but keeping LISP as the main programming language.

Since its inception, CREWS incorporated several AI techniques. The most visible part was state-space search, using a modified version of beam search (Bisiani 1992). The search tree was used as the unifying media of different modes of operation: in *manual mode*, using drag-and-drop, the user would tell the system the tasks to be placed in a duty, producing a successor of the current node, and the system would verify all constraints imposed upon the resulting duty, telling the user the constrains that were violated by the operation; in *semi-automatic mode*, the successors of the current node were generated and shown to the user in a intuitive way, letting the user to choose which one to use; in *automatic mode*, the system would carry a state-space search providing the user with a selected solution. The search tree and its states could be inspected at any moment during the search process. Since the user could perceive what was going on, could interact with the system by proposing alternatives or querying decisions, the term "white box" was coined to characterize the behavior of the system.

By combining clever heuristics with adequate cost functions, the system could be fine-tuned to optimize the relevant criteria chosen by the user. Knowledge was represented using a frame-based formalism. Labor rules were represented in a mixed declarative and procedural language with a specific interpreted developed by SISCOG. This enabled the separation of the rules from the code and the modification of the labor rules by the customer. Another aspect of AI that was omnipresent is the use of constraints. These were used by the automatic and semi-automatic modes to select the most constrained tasks as preferential tasks to be used in node expansion. Data dependencies were used in a component of CREWS called *Data Manager* to make sure that the data in the system, usually coming from different sources, was both consistent and complete. These dependencies were influenced by TMS-based systems (Doyle 1979) (Martins and Shapiro 1988).

The first deployment of CREWS was by the Dutch Railways, NS (Nederlandse Spoorwegen), the main passenger railroad operating company in the Netherlands, in a system called CREWS_NS, as reported in (Morgado and Martins 1998). In the version originally deployed, only the manual mode was used because the solutions provided by the state-space search were not satisfactory. In the years that followed, CREWS grew with the development of new modules and was successfully deployed by Norwegian Railways (Martins, Morgado, and Haugen 2003), Danish State Railways, Suburban Trains of Copenhagen, Finnish State Railways, London Underground, and Lisbon Metro (Martins and Morgado 2010).

Although CREWS_NS was appreciated by planners and the management of NSR could appreciate the flexibility in the change of labor rules, the state-space search algorithm remained a problem due to its very local view of the planning

process, producing sub-optimal solutions. SISCOG tried without success to incorporate other AI techniques to improve the quality of the solutions produced by the automatic mode (Almeida 2006) (Saldanha and Morgado 2003). NSR started to look for an external optimizer that could be linked to the system, ending up with TURNI (Abbink et al. 2005). So, for a while, CREWS_NS was used by NSR together with an external optimizer.

In 2003, SISCOG completed the development of an OR-based optimizer, IP-solver, followed an approach inspired by (Caprara, Fischetti, and Toth 1995). From 2003 to 2008, this version was successively improved, in order to try to solve larger problems and increasingly complex additional constraints, through various mechanisms, including a simple local search algorithm. Although IP-solver takes into account additional constraints, solving them in many cases, it often failed in problems with constraints that were very difficult to solve. This problem was finally solved with the AC-solver presented in this paper. In 2009, IP-solver started operating in the NSR, replacing TURNI. Between 2009 and 2010 a new version of IP-solver (Abbink et al. 2011) was introduced which, according to its authors, solves the largest crew duty planning problem in the world.

So, from 2003 to 2011, although still using many AI techniques, CREWS lost the AI-based component in its automatic mode. Since the IP-solver produces solutions without explaining how they were obtained, it was named the "black box". Solutions produced by the black box can be imported into the white box and be manipulated by the user.

The work reported in this paper brought again an important AI component to CREWS IP-solver optimizer that further improves the quality of the solutions produced.

Maintenance

The maintenance of a CREWS-based system is considered under two perspectives:

- *Maintenance of the product*. This corresponds to big fixes and the supply of new versions of CREWS. Under this perspective, about once a year, SISCOG announces a new version of the product and the customers may decide to migrate to this version. The latest release in CREWS includes the addition of AC-solver to CREWS optimizer, IP-solver. In this way all customers of SISCOG will improve the duty planning task with the use of the AI technology presented in this paper.
- *Maintenance of the customization*. This corresponds to changes in rules, in scheduling strategies, and the addition of small functionality adjustments. Part of this maintenance can be performed by the customer, resorting to the technical person in charge of the system, part of the changes are done by SISCOG as part of a maintenance contract.

Acknowledgements

SISCOG thanks all its staff for their work and dedication. Special thanks go to Carlos Iglésias and Rita Portugal of SISCOG's Innovation Department, helped creating the excellent scientific environment where this work was carried out. We also thank Dennis Huisman for his comments on a draft version of this paper.

References

Abbink, E. J. W.; Fischetti, M.; Kroon, L. G.; Timmer, G.; and Vromans, M. 2005. Reinventing crew scheduling at netherlands railways. *Interfaces* 35(5):393–401.

Abbink, E. J. W.; Albino, L.; Dollevoet, T.; Huisman, D.; Roussado, J.; and Saldanha, R. L. 2011. Solving large scale crew scheduling problems in practice. *Journal Public Transport* 3(2):149–164.

Almeida, F. 2006. *Planeamento de Recursos Móveis por Melhoramento Iterativo*. Ph.D. Dissertation, Instituto Superior Técnico.

Bisiani, R. 1992. Beam search. In *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, Inc. 56–58.

Caprara, A.; Fischetti, M.; and Toth, P. 1995. A heuristic method for the set covering problem. *Operations Research* 47(5):730–743.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence* 12(3):231–272.

Fabries, W. 2008. Personal communication.

Glover, F., and McMillan, C. 1986. The general employee scheduling problem. An integration of MS and AI. *Computers & Operations Research* 13(5):563–573.

Karp, R. M. 1972. Reducibility among combinatorial problems. In Miller, and Thatcher., eds., *Complexity of Computer Computations*. Plenum Press. 85–103.

Kirkpatrick, S.; C. D. Gelatt, J.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220(4598):671–680.

Martins, J. P., and Morgado, E. 2010. Case Studies in Planning Railroad Crew members. In *Proceedings of 2010 Joint Rail conference*, volume 2, 535–544. ASME.

Martins, J. P., and Shapiro, S. C. 1988. A model for belief revision. *Artificial Intelligence* 35(1):25–79.

Martins, J. P.; Morgado, E.; and Haugen, R. 2003. TPO: A system for scheduling and managing train crew in Norway. In *Proc. of the Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-03)*, 25–32. AAAI.

Morgado, E., and Martins, J. P. 1998. CREWS_NS: Scheduling train crew in The Netherlands. *AI Magazine* 19(1):25–38.

Morgado, F. 2011. Melhoramento iterativo de turnos para tripulações. Master's thesis, Instituto Superior Técnico.

Saldanha, R. L., and Morgado, E. 2003. Solving Set Partitioning problems with Global Constraint Programming. In Moura-Pires, and Abreu., eds., *Proc. 11th Portuguese Conference on Artificial Intelligence (EPIA 2003)*. Springer-Verlag. 101–115.

UNIFE. 2010. World Rail Market Study – status quo and outlook 2020. Technical report, www.unife.org.

Voudouris, C., and Tsang, E. P. K. 2003. Guided Local Search. In *Handbook of Metaheuristics*, volume 57. Kluwer Academic Publishers. 185–218.

Voudouris, C. 1997. *Guided Local Search for Combinatorial Optimisation Problems*. Ph.D. Dissertation, University of Essex.