Hybrid Approaches for Rostering: a Case Study in the Integration of Constraint Programming and Local Search

Raffaele Cipriano[1], Luca Di Gaspero[2], and Agostino Dovier[1]

[1] Dip. di Matematica e Informatica
raffaele.cipriano@gmail.com|dovier@dimi.uniud.it
[2] Dip. di Ingegneria Elettrica, Gestionale e Meccanica
l.digaspero@uniud.it
Università di Udine, via delle Scienze 208, I-33100, Udine, Italy

Abstract Different approaches in the hybridization of constraint programming and local search techniques have been recently proposed in the literature. In this paper we investigate two of them, namely the employment of local search to improve a solution found by constraint programming and the exploitation of a constraint model to perform the exploration of the local neighborhood. We apply the two approaches to a real-world personnel rostering problem arising at the department of neurology of the Udine University hospital and we report on computational studies on both real-world and randomly generated structured instances. The results highlight the benefits of the hybridization approach w.r.t. their component algorithms.

1 Introduction

Constraint Satisfaction Problems (CSPs) are a useful formalism for modeling many real problems, either discrete or continuous. Remarkable examples are planning, scheduling, timetabling, and so on. A CSP is generally defined as the problem of associating values (taken from a set of domains) to variables subject to a set of constraints. A solution of a CSP is an assignment of values to all the variables so that the constraints are satisfied. In some cases not all solutions are equally preferable, but we can associate a cost function to the variable assignments. In these cases we talk about Constrained Optimization Problems (COPs), and we are looking for a solution that (without loss of generality) minimizes the cost value. The solution methods for CSPs and COPs can be split into two categories:

- *Complete methods*, which systematically explore the whole solution space in search of a feasible (for CSPs) or an optimal (for COPs) solution.
- *Incomplete methods*, which rely on heuristics to focus on interesting areas of the solution space with the aim of finding a feasible solution (for CSPs) or a "good" one (COPs).

Constraint programming (CP) languages [2] are usually based on complete methods that analyze the search space alternating deterministic phases (constraint propagation) and non-deterministic phases (variable assignment), exploring implicitly or explicitly the whole search space. Local search (LS) methods [1], instead, rely on the definition of "proximity" (or neighborhood) and they explore only specific areas of the search space. Local search method, concentrating on some parts of the search space, can approximate optimal solutions in shorter time.

Two major types of approaches for combining the abilities of constraint programming and local search are presented in the literature [6,7] (in [8] constraint programming and local search are hybridized in a more liberal context):

1. a systematic-search algorithm based on constraint programming can be improved by inserting a local search algorithm at some point of the search procedure, e.g.:
   (a) at a leaf (i.e., on complete assignments or on an internal node (i.e., on a partial assignment) of the search tree explored by the constraint programming procedure, in order to improve the solution found;
   (b) at a node of the search tree, to restrict the list of child-nodes to explore;
   (c) to generate in a greedy way a path in the search tree;
2. a local search algorithm can benefit of the support of constraint programming, e.g.:
   (a) to analyze the neighborhood and discarding the neighboring solution that do not satisfy the constraints;
   (b) to explore a fragment of the neighborhood of the current solution;
   (c) to define the search of the best neighboring solution as a problem of constrained optimization (COP).

In this work we adopt the two hybrid techniques 1(a) and 2(b) and we apply them to a hospital personnel rostering problem. In the first approach we employ constraint programming for searching an initial feasible solution and subsequently we improve it by means of local search, using classical algorithms like hill climbing, steepest descent and tabu search. In the second approach we devise a local search algorithm (called *hybrid steepest descent*) that exploits a constraint programming model for the exploration of neighborhood fragments.

The employment of constraint programming for finding an initial feasible solution exhibits several advantages w.r.t. generating an initial random solution. Indeed, constraint programming allows us to find in short times (about 5-10 seconds) a feasible solution, providing a good starting point for local search.

The local search methods that exploits the constraint programming neighborhood model has evidenced how constraint programming can be used in the exhaustive exploration of neighborhood fragments, obtaining competitive local search procedures; furthermore, this approach is favorable only with huge search spaces. since for small problems the high computational overhead of the constraint programming is not payed back by remarkable improvements of the cost function.

The remainder of the paper is organized as follows: in Section 2 we present the formalization of the rostering problem analyzed. In Section 3 we provide some details about the implementation of the hybrid methods and in Section 4 we report the comparisons among the different solution techniques employed. Possible future directions of research are discussed in Section 5.

2 Hospital Personnel Rostering

The personnel *rostering* problem consists in the assignment of personnel requirements (usually expressed in terms of shift types) to qualified personnel over a certain planning period. The goal is to find a feasible assignment which minimizes a suitable objective function. Finding high-quality solutions to this problem is of extreme importance in knowledge-intensive labors, where the very specialized skills of personnel make impossible to exchange the duties among persons thus hardening the problem. This situation especially arises in hospital departments, which need an optimal schedule of the workforce that balances the trade-off between internal and external requirements such as the fair distribution of the workload among the different doctors and the assurance of a constant and efficient medical service to citizens.

Although in the last 30 years several studies proposes different solutions to this problem (e.g., by means of mathematical programming, multi-objective programming, constraint programming, expert systems, heuristic and meta-heuristic methods, see [3] for a comprehensive review), at present in Italian hospitals the problem is usually solved with pencil-and-paper by a doctor (*self-scheduling*). In this work we studied a particular family of rostering problems we called Neu-Rostering (NR for short), which model the personnel assignment problem of the department of Neurology of the University Hospital of Udine.

The problem is described in details in the following. Given m doctors, n days (the *temporal horizon*) and k possible shifts, the NR problem consists in assigning to each doctor the shifts to cover during the temporal horizon considered. In particular, for each day some shifts must be covered, based on the type of day (a workday or in the weekend) and of the weekday (some shifts are required only on Mondays, other on Tuesdays, and so on). Each shift must be assigned to one (and only one) doctor. Some shifts (e.g., the most general ones like urgent calls shifts) can be covered by any doctor, while others can be covered only by a restricted number of doctors on the basis of their competence. Every doctor can specify a list of days of the *temporal horizon* in which he/she is not available (e.g., because of days off, conferences, courses, teaching, ...). Moreover, there are a set of "temporal" restrictions that regulate the coverage of the shifts: every shift has fixed working times, every doctor can be assigned to consecutive shifts but, in this case, there is an upper bound on the number of consecutive working hours that cannot be exceeded, a doctor should have a rest between two assigned shifts.

2.1 Formulation of NR as CSP

Let $E = \{e_1, \ldots, e_k\}$ be the set of all possible shifts; we consider the variables $G_1, ..., G_n$, one for each of the n days, where $G_i \subseteq E$ is defined as follows:

$$G_i = \{e_k \in E \mid e_k \text{ is a shift to be covered on the day } i\}$$

The skills of each doctor $j \in 1..m$ are encoded in a set $M_j \subseteq E$, defined as follows:

$$M_j = \{e_k \in E \mid e_k \text{ is a shift that doctor } j \text{ is allowed to cover}\}$$

Furthermore, let consider m sets $F_j \subseteq \{1, \ldots, n\}$ that represent the days of the temporal horizon in which doctor j is not available.

Let $C \subseteq E \times E$ be the set of pairs (e_i, e_j) of shifts that *can* be covered in the same day by the same doctor. Then, we define as T the set of shift sets (singletons or doubletons) that can be covered by a doctor in the same day:

$$T = \{\{e_i\} \mid e_i \in E\} \cup \{\{e_i, e_j\} \mid (e_i, e_j) \in C\}$$

Let $S \subseteq E \times E$ be the set of pairs of shifts (e_i, e_j) that can be covered by the same doctor in consecutive days.

The problem input is defined by the sets generated above. In order to coding the problem constraints we consider for all $i \in 1..n, j \in 1..m$ the variables $O_{i,j}$ whose domains are the sets $T \cup \{\emptyset\}$. $O_{i,j} = \textsf{set of shifts}$ means that in day i doctor j will cover the set of shifts indicated. An assignment to these variables is an (admissible) *solution* if and only if:

1. $\forall i \in 1..n \forall j \in 1..m \quad O_{i,j} \subseteq M_j$ (*competence*: each doctor can only cover shifts he/she is qualified for);
2. $\forall i \in 1..n \forall j \in 1..m \quad i \in F_j \Rightarrow O_{i,j} = \emptyset$ (*availability*: each requirements for days off is satisfied);
3. $\forall i \in 1..n \forall j \in 1..m \quad O_{i,j} \in T$ (*max hours*: each assignment is coherent with the maximum number of hours per day);
4. $\forall i \in 1..n-1 \forall j \in 1..m \quad \forall t \in O_{i,j} \forall t' \in O_{i+1,j} \quad (t, t') \in S$ (*legal rules*: there is the suitable distance between consecutive shifts for the same doctor);
5. $\forall i \in 1..n \quad \cup_{j=1}^{m} O_{i,j} = G_i$ (*coverage*: all required shifts are covered);
6. $\forall i \in 1..n \forall j_1 \in 1..m \forall j_2 \in 1..m (j_1 \neq j_2 \rightarrow O_{i,j_1} \cap O_{i,j_2} = \emptyset)$ (*mutual exclusion*: each shift is covered by at most one doctor).

Let us observe that our encoding of the NR problem uses $m \cdot n \cdot k$ Boolean variables with the following intuitive meaning: $\forall i \in 1..m \forall j \in 1..n \forall z \in 1..k \, X_{i,j,z} = 1$ if and only if doctor i covers shift z in the day j. Let us observe that establish the existence of a solution to an instance of NR is NP-complete (by means of a straightforward encoding of 3-GRAPHCOLORING).

2.2 COP model for NR

Our NR implementation is endowed with an objective function used to model some soft constraints associated to the problem and to choose one solution w.r.t. others. This function has been obtained by eliciting information from the manager of the Neurology Dept. and it is based on five parameters.

Weekend, Nights, and Guards. One of the objectives is to balance the work of the doctors in the week-ends. In the week-ends there are only two types of shifts, denoted by Urgent calls Morning and Afternoon (UMUP) and Urgent calls Night (UN). We looked for an expression that assumes high values when shifts are badly distributed among the doctors. Given a doctor i we propose to sum all the values (actually Boolean values are seen as integer values here) related to shifts UMUP and UN in the weekends and in the holidays:

$$\mathsf{Wk} = \textstyle\sum_{i=1}^{m} (\sum_{j \in GWE, z \in TWE} X_{i,j,z})^2$$

In the above formula, i ranges over doctors, j over days to be selected in the set GWE of days in weekends and holidays (e.g., if December 1st is on Monday, then $n = 31$ and $GWE = \{6, 7, 13, 14, 20, 21, 25, 27, 28\}$), and z ranges over shifts in the set TWE (in this case $TWE = \{\mathsf{UMUP}, \mathsf{UN}\}$). In order to balance the amounts of night shifts UN and of Urgent calls morning UM (save those in the weekends and holidays, already considered in Wk—shifts UMUP) we define the two following formulas in analogous way:

$$\mathsf{Nt} = \textstyle\sum_{i=1}^{m} (\sum_{j \in 1..n \setminus GWE} X_{i,j,\mathsf{UN}})^2, \; \mathsf{Gu} = \sum_{i=1}^{m} (\sum_{j \in 1..n \setminus GWE} X_{i,j,\mathsf{UM}})^2$$

Undesired Pairs. Here we take care of the number of days where a doctor works either in a morning and in a afternoon shift. This is highly undesirable for doctors that work both in the public hospital and as private professionals. We define the following formula for the variable Dp:

$$\mathsf{Dp} = \sum_{i \in 1..m, j \in 1..n} \left(\sum_{z \in \mathsf{AM}} X_{i,j,z} \cdot \sum_{z \in \mathsf{PM}} X_{i,j,z} \right)$$

where AM (resp., PM) is the set of all morning (resp., afternoon) shifts. Let us observe that the product of the two inner sums assume value 1 only when a doctor is employed either in the morning or in the afternoon and 0 elsewhere (values greater than 1 are forbidden by max hours constraints).

Consecutive shifts. Here we consider situations where a doctor is employed in the same type of shifts for three consecutive days. Some of these sequences are penalize (we count as 1), other (e.g. RMu and RMg) are encouraged (we count as -1). Therefore, we assign to Cn the following formula, where E is the set of all possible shifts:

$$\mathsf{Cn} = \sum_{\substack{i \in 1..m, j \in 1..n-2, \\ z \in E \setminus \{\mathsf{RMu}, \mathsf{RMg}\}}} (X_{i,j,z} X_{i,j+1,z} X_{i,j+2,z}) - \sum_{\substack{i \in 1..m, j \in 1..n-2, \\ z \in \{\mathsf{RMu}, \mathsf{RMg}\}}} (X_{i,j,z} X_{i,j+1,z} X_{i,j+2,z})$$

Objective functions. Finally, we assign a weight to each one of the just defined five variables in order to balance solutions to NR. In the case studied, where $m = 20$, $k = 28$, and $n \in 28..31$ (for a temporal horizon of one month there are more or less 16000 variables) is the following:

$$\text{FObj} = 50\mathsf{Wk} + 40\mathsf{Nt} + 30\mathsf{Dp} + 20\mathsf{Gu} + 10\mathsf{Cn}$$

Weights have been chosen in the following way. Starting from the by hand computed solutions for the year 2005, we deduced the holidays and the various constraints required by the doctors. Then we generated some sequences of solutions and showed them to the responsible of the Neurology. We modified the weights using his feedback in such a way that minimal values of the function are associated to more preferable solutions.

3 Implementation

In this work, for the solution of the NR problem we adopt two hybrid techniques, which integrate constraint programming and local search. In the first approach we employ constraint programming for searching an initial feasible solution and subsequently we improve it by means of local search. In the second approach we devise a local search algorithm that exploits a constraint programming model for the exploration of the neighborhood.

3.1 Application architecture

In order to solve the NR we design a software tool made up of two main modules:

1. the *FirstSolution* module, a program implemented by means of the `clpfd` SICStus Prolog package [4] which models the NR problem. The module let the user specify a problem instance and it starts processing it as soon as a feasible solution is found. If a feasible solution does not exists this module raises an error and stops the execution or, whenever possible, it relaxes some parts of the model leading to an approximate solution.
2. the *LocalSearch* module, which implements a set of local search algorithms for the NR problem. This module has been developed using the JEasyLocal framework, a Java version of the C++ framework EasyLocal++ [5]. The module takes as input a feasible solution obtained by the FirstSolution module and improves it by means of a local search algorithm that can be chosen by the user. The final solution found by this module can be further improved applying a different local search algorithm in an iterative process. The local search techniques implemented in this module are hill climbing, steepest descent and tabu search. Moreover, this module features a local search solver that uses the steepest descent technique for driving a constraint programming formulation of the exploration of the neighborhood.

Dr.	1	2	...
Jones	...	DH	...
...	...	...	...
Freud		UM	...
...	...	...	...

$\Longrightarrow$

Dr.	1	2	...
Jones	...	UM	...
...	...	...	...
Freud		DH	...
...	...	...	...

Figure 1: An example of an exchange move

Local Search Among other entities (i.e., the definition of the search space and the cost function that in this case are borrowed from the constraint programming formulation), to specify a local search algorithm it is necessary to define the *move*; that is, the local perturbation to be applied to a solution in order to obtain a neighboring one. To this aim we define the following move, called *exchange*:

"Given a specific day and working time, exchange the shifts of two doctors"

For example, if Dr. Freud covers the shift UM in the morning of day 2 and Dr. Jones covers the shift DH (Day Hospital) in the morning of the same day, a possible exchange move consists in swapping the shifts UM and DH between the two doctors, so that in the morning of day 2 Dr. Jones will be assigned to the shift UM and Dr. Freud will cover the shift DH as shown in Figure 1.

The shifts involved in the exchange move can be working shifts or rest periods. When we exchange a working shift with a rest one, the doctor currently in rest will get the working shift and the doctor currently working will get a rest shift. Exchange moves that involve doctors that are both currently in a rest period do not affect the solution and therefore are *idle* moves; conversely, all other types of exchange moves are meaningful and modify the solution. An exchange move is therefore identified by: the two doctors participating in the exchange, the day of the time horizon and the working time (that can be "Morning" or "Afternoon").

Notice that, given a solution, the size of the neighborhood (i.e., the number of neighboring solutions of the current state) is equal to $\frac{m(m-1)}{2}2n = O(m^2n)$. It is possible to generalize the concept of exchange move by introducing the *compound* exchange move defined as follows:

"A compound exchange move is a sequence of one or more exchange moves"

The compound moves are very useful to handle consecutive shifts that must (or it is preferable to) be moved together. It is worth to notice that the definition of the exchange moves always lead to states where the covering constraints are satisfied: indeed, if solution A satisfy the covering constraints so will solution B obtained by an exchange move since no shift is added or removed from a day column, but simply the assignment of two doctors are swapped. As a consequence, making sure that the local search procedure will start from a feasible solution (like the one obtained by the FirstSolution module) and applying only exchange moves there is no need to make the covering constraint explicit. However, we observe that an exchange move could lead to a state where other types of constraints are violated: for example a doctor could be assigned a shift for which he/she is not qualified, or he/she could be not available on that day. These violations are taken into account by an objective function $FObj$ which penalizes such situations.

In the following sections we briefly outline the local search algorithms we have developed.

Hill climbing. The hill climbing (HC) strategy adopted in this work is the so-called randomized hill climbing: an exchange move is randomly drawn and applied to the current solution. If the solution obtained improves or has an equal value of the objective function, then it is accepted and it becomes the new incumbent solution; conversely, if the new solution worsens the cost function it is discarded and a new random move is drawn. This procedure is iterated and the whole process stops when a user specified time-out has expired.

The crucial aspect in the implementation of this method concerns the random generation of the moves, which could lead to the generation and testing of a lot of idle moves. The random procedure has been therefore biased toward meaningful moves by enforcing that the first doctor of the exchange move must not be on a rest period, thus avoiding the generation of idle moves.

Steepest descent. The steepest descent (SD) strategy consists in the full exploration of the neighborhood of the current solution, looking at the solution that gives the biggest improvement of the objective function. This move is then applied to the current state to obtain the new incumbent solution. The procedure is iterated and it stops as soon as no improving move can be found.

Compared to HC, this procedure is more time-consuming but it generally leads toward bigger improvements of the objective function. The key aspect of this method is the procedure employed for the enumeration of the moves. Since the evaluation of the objective function is a costly operation it is advisable to avoid unnecessary computations, especially on moves that lead to states where the constraints are violated. The enumeration procedure we have implemented makes use of a basic knowledge about the constraints and it skips such moves, allowing us to save computation time.

Tabu search. This method (TS) explores a subset of the neighborhood of the current solution and applies the move that gives the minimum value of the objective function, regardless the fact that this value is better or worse than the one of the previous solution. This allows the method to escape from local minima, but at the risk of cycling among a set of solution. To avoid the latter phenomenon the method employs the so-called "Tabu List", a memory of recently applied moves, and it forbids the application of moves that are inverses of the moves in the list (which would lead to an already visited state).

Among different variants of the memory mechanism presented in the literature we employ the so-called dynamic tabu list. The list contains a number of moves comprised between two values k_{min} e k_{max}, which are parameters of the method. Once a move enters the list it is assigned a random integer value in the range $k_{min}..k_{max}$ that corresponds to the number of iterations the move is kept in the tabu list. For this problem we find out experimentally that the best setting of these parameters is $k_{min} = 5$ and $k_{max} = 10$.

As for the aspiration criterion, which overrides the prohibition status of the moves, we choose to accept also moves in the tabu list when they lead to a state that is better than the current best solution.

Hybrid steepest descent. Finally, we implemented an hybrid local search algorithm driven by the steepest descent strategy (HSD), which employs a neighborhood model encoded in SICStus Prolog. The idea behind this algorithm is to explore fragments of the neighborhood of the current solution by letting the constraint programming solver to find a representative solution (the best neighbor) of the current fragment. The neighborhood fragments have to be chosen so that they form a partition of the whole neighborhood of the current solution. The algorithm then accepts the best among the representative solutions that improve the objective function, inspired by the steepest descent strategy. The search of the best representative solution is performed by the `labeling` predicate of SICStus Prolog.

The concept of "neighborhood fragment" we had taken into account is based on a single day: given a solution S and a day of the temporal horizon x, the neighborhood of S w.r.t. the day x consists in the set of all solutions S' that are identical to S for all days $d \neq x$ and they differ from S for the shifts of day x. Hence, every solution will have n possible neighborhood fragments. The exploration of a neighborhood fragment w.r.t. the day x means the evaluation of all possible permutations of shifts on that day: the best permutation will be the representative of that neighborhood fragment.

This approach shares some similarities with the work of Pesant and Gendreau [9], however it differs in the type of move employed and in the granularity of the neighborhood exploration. In our case, indeed, we explore a full set of exchange moves (i.e., duty exchanges between doctors) whereas in [9] the authors use a insertion move (i.e., a duty is assigned to a doctor and not removed elsewhere). Furthermore, our neighborhood model involves a portion of shifts that insist on a single day only, while [9] neighborhood is restricted to a single shift.

4 Experiments

To the aim of comparing the four hybrid algorithms described in the previous section, we carried out an experimental evaluation of the solvers. Three types of experiments have been performed: (i) on randomly built structured instances of variable size; (ii) on real-world instances in long-runs; (iii) the best solver is compared with self-scheduling solutions.

First test — Methodology. The goal of the first test is to analyze the behavior of the algorithms on sets of structured instances that are similar to real-world

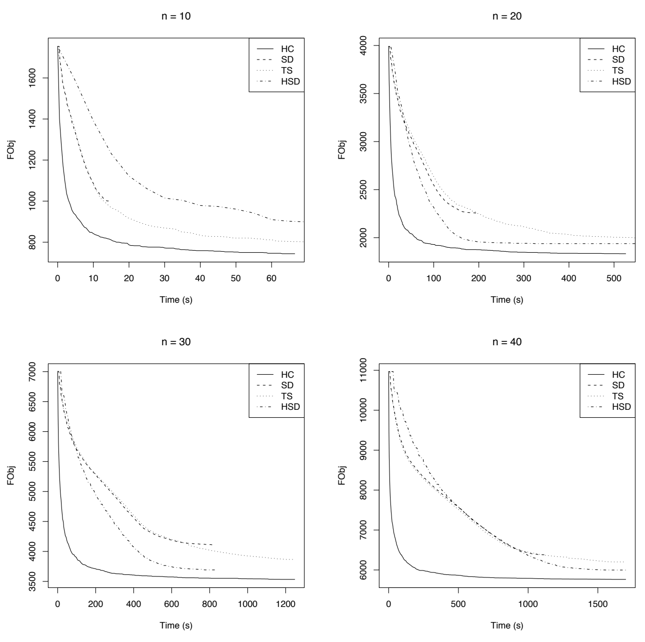

Figure 2: Average evolution of the objective function on instances with temporal horizon n equals to 10, 20, 30 and 40 days

ones. We randomly generate 4 series of 10 instances whose temporal horizon n consists of 10, 20, 30 and 40 days, respectively.

All the instances have been solved by the four algorithms, accounting for a total of 160 runs. Each algorithm was granted a running time proportional to the instance size, namely of $80 \cdot n$ seconds. During each run, we record the values of the objective function that corresponds to improvements of the current solution together with the running time spent. These data have been aggregated in order to analyze the average behavior of the different algorithms on each group of 10 instances. To this aim we perform a discretization of the data on regular time intervals; subsequently, for each discrete interval we compute the average value of the objective function on the 10 instances.

First test – Results. Figure 2 reports the evolution of the objective function for the different methods on the instances with temporal horizon n equals to 10, 20, 30 and 40 days.

From the picture it is apparent that HC outperforms all the other methods on all groups of instances. Indeed, thanks to the non-exhaustive sampling of the neighborhood this method is able to find reasonably good improvements quickly, leading to a fast decrease of the objective function from the very beginning of the search process. Furthermore, HC does not get stuck in local minima (as other methods do) but it keeps perturbing the current solution with sideways moves that could possibly lead to explore new regions of the search space.

The classical SD strategy, instead, is the worst method among the ones tested. Because of the full exploration of the neighborhood, the method is slower than the hill climbing. Furthermore, the thoroughness of the exploration at each search step is not rewarded by a substantial decrease of the objective function. Finally, the method shows the intrinsic shortcoming of getting stuck in local minima and the search stagnates as soon as the first local minimum is found.

The behavior of TS lies between the two previous methods: in early stages of the search the method behaves exactly as the steepest descent (indeed, initially in the graph the two lines overlap), while the prohibition mechanisms start playing its role in diversifying the solution as soon as the first local minimum is found. Unfortunately, due to the high computational cost of the neighborhood exploration (especially on mid- and big-sized instances) the method performs worse than hill climbing.

HSD deserves a more thorough analysis since its behavior varies on the basis of the size of the instances. On smaller instances (10 days) its behavior is the worst in terms of decrease speed of the objective function. This can be explained by the high computational overhead needed to setting up the exploration of the neighborhood fragments. In fact, for each fragment a new constraint model should be posted to the constraint store giving rise to a significant computation effort. This overhead, on smaller instances, is not rewarded by a high decrease of the objective function at each step of the search. Conversely, on the mid-sized instances (20 and 30 days) the increased computational effort is repaid by a greater decrease of the objective function and this method result more competitive than tabu search just after the first tens of seconds. However, the method does not scale well on big-sized instances (40 days): although the tendency of having a better behavior than tabu search is confirmed, this behavior is apparent only after some hundred of seconds.

Second test — Methodology. The aim of this experiment is to analyze the behavior of the four methods on two real-world instances in a deployment situation, i.e., the methods are granted a running time of 12 hours in order to evaluate more precisely their behavior on longer runs. Both the instances have a temporal horizon of 30 days. The recorded data are the same of the previous experiment. However, differently from the previous test we need not to process the data since the results are relative to singular instances.

Second test — Results. The behavior of the objective function on the two instances is comparable to the outcomes of the previous test on the instances of size 30, therefore we do not show it here for brevity. Here we report in Table 1a the values of the objective function reached by the four methods after 12 hours.

In the last line we report also the best value reached after 12 hours by the exhaustive search performed by the constraint solver employing the constraint programming model alone. Those values are about two times higher than the results of the hybrid methods and they fully justify the employment of the hybrid

Method	FObj	
	Instance 1	Instance 2
HC	**3500**	**3240**
SD	4120	3760
TS	3520	3350
HSD	3630	3460
CLP(FD)	7590	5990

(a) Final results of the four methods and of the constraint solver

	Instance 1			Instance 2		
Method	+5%	+2%	+1%	+5%	+2%	+1%
HC	111	1381	4815	329	752	2758
SD	–	–	–	–	–	–
TS	983	1343	1471	1590	–	–
HSD	664	–	–	–	–	–

(b) Time (in seconds) to reach an approximation within x% of the best value found

Table 1: Results in 12 hours of computation

approaches for this problem. For both instances the hill climbing method is able to find the best result and it is not outperformed by any other method. However, it is worth to notice that, when granted with sufficient time, tabu search shows a good behavior and its result is not that far from the one of hill climbing.

Finally, in Table 1b we report the time needed to reach an approximation within a given percentage of the best solution value found by the methods after 12 hours.

From the table it is possible to notice that, for hill climbing and tabu search on instance 1, the convergence to good values (2% from the best value known) is obtained in less than 25 minutes. However, only HC could reach values close to the best known for instance 2. The reasons of this modest performance of tabu search will be matter of further investigation.

Third test — Methodology. From the previous tests, hill climbing results the best method (especially when provided with a short time limit), so we decided to compare the solutions obtained from this method with solutions manually obtained by the doctor who is in charge of self-scheduling. The available data ranges from the monthly requirements of September to December 2005. We compare the values obtained for the various components of the objective functions.

Third test — Results. For the four months considered in the experiment we show in Figure 3 the outcomes of the self-schedule (the left-hand column of each pair) and the result of a short hill climbing run (5 minute of CPU time). In Table 2 we report the the percentage of improvement over the self-schedule. The data is disaggregated for the various components of the objective function and is 1 minus the ratio of the cost value found by hill climbing over the cost found by the human (so that positive values indicate improvements).

You can notice that, even though the self-schedules are really high-quality ones, the hill climbing method is able to achieve some further improvements, especially on critical components like Guards and Undesired Pairs. For Consecutive Shifts (which is a negative component, since they are preferred), instead, we can improve the values only in two cases out of the four instances.

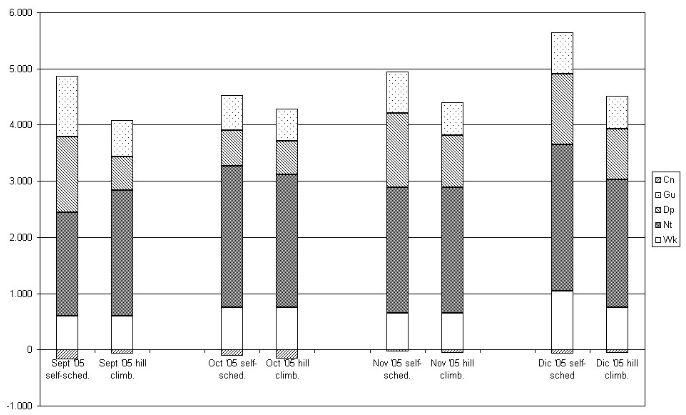

Figure 3: Comparison of HC solver runs of 5 minutes with self-scheduling

Component	Sep	Oct	Nov	Dec
Wk	0%	0%	0%	29%
Nt	-22%	6%	0%	12%
Dp	56%	5%	30%	29%
Gu	41%	6%	22%	22%
Cn	-143%	31%	50%	-17%
FObj	15%	6%	12%	20%

Table 2: Percentage of improvement of HC with respect to self-scheduling

5 Future work and Conclusions

This work is still ongoing and the presented results are still preliminary. We wish to extend the research pursued in this paper along the following two lines:

1. integrating the tool with new local search methods, and
2. improving the resolution technique implemented.

We have experimentally verified that HC outperforms other tested methods: it employs less time than the others to find a good move. We plan to test other local search methods, such as Tabu Search with First Improvement or Elite Strategy that, visiting a small part of the neighbors, should be comparable with HC w.r.t. the time for finding a good move. We also plan to implement a Simulated Annealing method. However, we believe that a long stage for tuning parameters is needed in order to effectively use this method.

Moreover, we would like to develop a constraint solver on finite domains ad hoc for this problem in order to speed-up this stage, to ease the integration with

JEasylocal, and to be independent from commercial languages. From a methodological point of view, we would like to analyze more carefully the behavior of our algorithms, using statistical methods and tuning more precisely the various parameters. In particular, further insight is needed to explain the modest behavior of tabu search.

Our system is currently in use in the Neurology Dept. of the Udine University Hospital. Acceptable solutions (with 20 doctors and a temporal horizons of one month) are obtained in a couple of minutes on a (average) PC. Future works will also include the generalization of the system in order to be usable by other hospital departments.

Acknowledgments

This work is partially supported by MIUR project PRIN 2005015491 and FIRB project RBNE03B8KK.

References

1. Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester (UK), 1997.
2. Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge (UK), 2003.
3. Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
4. Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In H. Glaser, Hartel P., and Kucken H., editors, *Programming Languages: Implementations, Logics, and Programming*, number 1292 in Lecture Notes in Computer Science, pages 191–206. Springer-Verlag, Berlin (Germany), 1997.
5. Luca Di Gaspero and Andrea Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software — Practice & Experience*, 33(8):733–765, July 2003.
6. Filippo Focacci, François Laburthe, and Andrea Lodi. Local search and constraint programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, chapter Local Search and Constraint Programming, pages 369–403. Kluwer Academic Publishers, 2003.
7. Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristic. *Artificial Intelligence*, 139(1):21–45, 2002.
8. Eric Monfroy, Frédéric Saubion, and Tony Lambert. On hybridization of local search and constraint propagation. In Bart Demoen and Vladimir Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in Lecture Notes in Computer Science, pages 299–313. Springer-Verlag, Berlin (Germany), 2004.
9. Gilles Pesant and Michel Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.