High-performance local search for task scheduling with human resource allocation

Bertrand Estellon[1], Frédéric Gardi[2], Karim Nouioua[1]

[1] Laboratoire d'Informatique Fondamentale – CNRS UMR 6166, Université Aix-Marseille II – Faculté des Sciences de Luminy, Marseille, France

[2] Bouygues e-lab, Paris, France

bertrand.estellon@lif.univ-mrs.fr, fgardi@bouygues.com, karim.nouioua@lif.univ-mrs.fr

Abstract. In this paper, a real-life problem of task scheduling with human resource allocation is addressed. This problem was approached by the authors in the context of the ROADEF 2007 Challenge, which is an international competition organized by the French Operations Research Society. The subject of the contest, proposed by the telecommunications company France Télécom, consists in planning maintenance interventions and teams of technicians needed for their achievements. The addressed combinatorial optimization problem is very hard: it contains several NP-hard subproblems and its scale (hundreds of interventions and technicians) induces a huge combinatorics. An effective and efficient local-search heuristic is described to solve this problem. This algorithm was ranked 2nd of the competition (over the 35 teams who have submitted a solution). Moreover, a methodology is revealed to design and engineer high-performance local-search heuristics for solving practically discrete optimization problems.

1 Presentation of the problem

The problem proposed by the telecommunications company France Télécom as subject of ROADEF 2007 Challenge [25] (an international competition organized every two years by the French Operations Research Society) can be viewed as a task scheduling problem with resource allocation. Here the tasks to plan are maintenance interventions and their achievement requires human resources, some technicians, each one having a skill level in different domains. The interventions are more or less priority; on the whole, 4 levels of priority are defined. Then, the objective is to minimize a linear function which depends on ending times of latest interventions for each priority.

Formally, the input of the problem is composed of n interventions I_i and of m technicians T_t. To each technician T_t is associated its skill level $C(t, d)$ in the domain d and its availability $P(t, j)$ on day j (1 for available, 0 otherwise). Each intervention has several characteristics too: $D(i)$ its execution time, $R(i, d, l)$ the number of technicians of level l in domain d required for its completion, $Z(i)$ its priority level.

Concerning skills, we precise that the different domains of skill are disjoint, but that the levels of each domain are hierarchically organized. Then, a technician of level l in domain d is able to perform any intervention requiring a smaller skill level ($l' < l$) in the same domain. Consequently, the constants $R(i, d, l)$ are cumulative, in the sense that they specify the number of technicians needed at level at least l in domain d. For example, for an intervention I_i which requires two technicians of level 1 and one technician of level 3 in domain d, we have $R(i, d, 0) = 3$, $R(i, d, 1) = 3$, $R(i, d, 2) = 1$, $R(i, d, 3) = 1$ and $R(i, d, l) = 0$ for all $l \geq 4$. Such a definition implies that the $R(i, d, l)$ are non-increasing according to the index l: $R(i, d, l) \geq R(i, d, l')$ for all $l \leq l'$.

Then, the notion of team arises. Daily, the (available) technicians must be grouped into teams (even if a team may be composed of only one technician). We insist on the fact that a team is formed for the entire day (for practical reasons). Then, the problem is to partition daily the technicians into teams and to assign them a set of interventions, in order to minimize an objective function depending on the ending dates of the interventions. Two constraints lie on this assignment: the sum of the lengths of interventions (which are completed sequentially) can not exceed the length of a working day fixed to $H = 120$ and the skills of the team must cover the skills required by the set of tasks in each domain. Finally, a solution of the problem is given as follows: for each day j, the team $E_{j,e}$ to which belongs the technician T_t (the team $E_{j,0}$ contains all the technicians not available on this day); for each intervention I_i, the day j_i and the starting time h_i of its execution as well as the team $E_{j,e}$ in charge of its execution.

The objective of the planning is to minimize the following cost function: $28t_1 + 14t_2 + 4t_3 + f$, where t_k denotes the ending date among those of the latest interventions of priority k and f denotes the ending date of all interventions. The starting date d_i (resp. ending date f_i) of an intervention I_i is obtained as $j_i \cdot H + h_i$ (resp. $j_i \cdot H + h_i + D(i)$), the days being numbered from 0. Initially, this objective function was supposed to imply the minimization of the t_k's in lexicographic order ($t_1 \succ t_2 \succ t_3 \succ f$). However, compensations between the four terms of the objective function were allowed during the competition (impacting gravely our approach as it will be seen later).

Finally, the scope of the problem may be extended in two ways. The first is to introduce precedence relations between interventions: for all intervention I_i, one can define a set $P(i)$ of interventions which must be completed before starting I_i (that is to say, any intervention $I_{i'} \in P(i)$ must satisfy the inequality $f_{i'} \leq d_i$). Note that the natural lapses of time between interventions (travel, breaks, etc.) are here considered as null. The second extension is to define a budget B allowing to subcontract a number of interventions. Then, a cost $S(i)$ is given for any intervention I_i and the sum of the cost $S(i)$ of all abandoned interventions must not exceed the budget B. In order to ensure the respect of precedences in this case, any abandoned intervention I_i leads to recursively abandon any intervention $I_{i'}$ such that $I_i \in P(i')$.

2 Contributions

To the best of our knowledge, this problem was never addressed in these terms in the literature, both from fundamental and experimental points of view. Because of its large definition, the problem contains several NP-hard subproblems. For any partition of technicians into teams a given day, determining if a set of interventions is assignable to these teams while respecting the working duration H, the precedence constraints and the skill constraints is NP-complete, even if the execution time of all interventions is unit (all interventions have equal execution time), the number of teams is fixed to two and the precedence graph is isomorphic to a set of vertex-disjoint paths [15]. In the case of arbitrary precedence constraints, the problem remains NP-complete, even if the execution time of all interventions is unit and the skill constraints are omitted (any intervention can be performed by any team) [8]. Minimizing the number of days to plan all interventions is NP-hard in the strong sense, even if the interventions are performed by one sole team each day (containing all available technicians), without precedence and skill constraints. Indeed, this subproblem corresponds to a bin-packing problem [8] when the length H is given as an input of the problem. Finally, maximizing the sum of lengths of the set of abandoned interventions is equivalent to a knapsack problem (with precedence constraints) [8].

Because of its hardness and large scale (hundreds of interventions and technicians), such a problem is typical of real-life discrete optimisation problems encountered in business and industry. In this paper, an effective and efficient local-search heuristic is described to solve this problem. Our algorithm was ranked 2nd of the ROADEF 2007 Challenge (over the 35 teams who have submitted a solution). The victorious algorithm, due to Hurkens [14], can be viewed as a local-search heuristic where large neighborhoods [1] are explored by integer linear programming (using ILOG CPLEX 10.0 solver); the team Cordeau-Laporte-Pasin-Ropke [4], ranked 2nd ex æquo, have also developed a large neighborhood search approach, but based on destroy and repair moves. Before describing our algorithm, we outline the methodology followed to design and implement it. This methodology, already used at our winning participation to the ROADEF 2005 Challenge [5, 6, 24], is a simple and clear recipe to engineer high-performance local-search heuristics for solving practically discrete optimization problems. Another successful application of this methodology for solving real-life inventory routing problems is presented in a companion paper [3].

For more details on high-performance algorithm engineering, the reader is referred to the papers by Moret et al. [20, 21] and, as an example, to the outstanding works of Helsgaun [11–13] on the traveling salesman problem.

3 Methodology: three-layers design

Several papers have been published describing methodologies for engineering local-search heuristics (see for example the survey edited by Aarts and Lenstra [2]). But many of these methodological papers are essentially concentrating on

search strategies and more particularly metaheuristics (see for example [10, 17, 22]). In this paper, we suggest to approach the engineering of local-search heuristics according to the following abc framework: a) search strategy, b) moves, c) algorithmics & implementation. We claim that the performance of local-search heuristics depends *equally* on the good treatment of each of these three layers. In fact, each one covers a fondamental point of the local-search paradigm: the definition of the search graph and the exploration of this graph. Figure 1 summarizes the key points of the methodology; note that only a few simple concepts are introduced for describing this one.

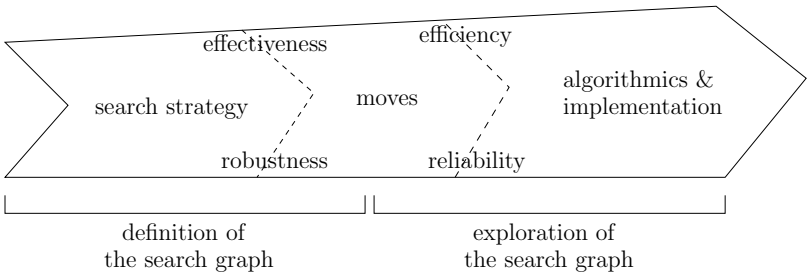

Fig. 1. The three layers of the methodology.

The *search space* $\mathcal{S} = (S, f)$, with S the set of solutions of the problem and $f : S \mapsto \mathcal{R}$ the objective function to minimize over this set, is defined as the discrete space into which local search walks. The search strategy, dedicated to the problem (or even to instances of the problem), allows to redefine the search space $\mathcal{S}$ if necessary. Indeed, the design of the search strategy may lead to redefine the original couple (S, f) into a surrogate one, denoted by (S_g, f_g), which supports the convergence of local search towards high-quality local optima. The idea is to increase the density of the search space $\mathcal{S}$ (more solutions in S_g). A way to do that is to relax some constraints of the problem by switching them into the objective function (similarly to Lagrangian relaxations in mathematical programming [19, pp. 349–368]). The idea is to relax only business constraints, and not physical constraints inducing the intrinsic combinatorial structure of the problem (matching, partial ordering, etc.). Indeed, relaxing constraints which strongly structure the solutions of the problem enables a wider diversification, but makes more difficult the convergence toward a new admissible solution.

Then, the *search graph* $\mathcal{G} = (S_g, f_g, A)$, associated to a local-search algorithm, is defined as the directed graph obtained by adding an arc $a \in A$ from $s \in S_g$ to $s' \in S_g$ if a move allows to reach the solution s' from s. Vertices of $S \subseteq S_g$ are green, whereas vertices in $S_g \setminus S$ are red. In the same way, the set A of arcs is partitioned such that $(s, s') \in A$ is green if $f_g(s') \leq f_g(s)$, or red otherwise. Then, the iterations of a local-search algorithm (that is, all the solutions visited during its walk) draw a subgraph in its associated search graph $\mathcal{G}$, inducing a

green-arc path. Thus, the red points of the space serve as bridging points to reach better admissible solutions, that is, green points having a better cost in the sense of f.

The *moves* (also called *transformations*) play a central role because they induce the connectivity of the search graph, which is decisive for convergence. Then the idea is to increase the density of the search graph $\mathcal{G}$ (more arcs in A) by defining a lot of moves, more or less orthogonal, more or less large, more or less specialized. This latter notion consists in increasing the success probability of a move (the number of red arcs visited before finding a green one) by using structural properties specific to the problem or even to the instances (see for example the work of Helsgaun [11–13] on travelling salesman problems or the works of the authors [5, 6] on car sequencing problems). Note that the idea which consists in using systematically a large pool of moves (i.e., of neighborhoods) appears at the root of well-known metaheuristics like Iterated Local Search or Variable Neighborhood Search (see [9] for more details).

This is at these levels – search strategy and moves – that some fragments of metaheuristics can be incorporated (thresholds, tabu lists). However, from our point of view, the diversification of the search must be firstly attained through the (re)definition of the search space (*density*) and the definition of moves (*connectivity*), and not only through a meta-strategy. The main reason is that such a diversification is guided and controlled via the surrogate objective function, unlike traditional metaheuristics. This is why we prefer, at least for starting, implementing a basic first-improvement descent strategy [2] with stochastic choice of moves. In this case, the diversification is realized by accepting to move to solutions with equal cost. Note that the introduction of stochastic elements in every choice made during the search is shown to improve the diversification, in particular by naturally avoiding cycling phenomena (nevertheless, stochastic does not mean uniform).

Finally, *algorithmics*, in particular those related to the evaluation of moves (that is, the exploration of neighborhoods), is crucial for *efficiency*. Since local search is an incomplete search technique, its effectiveness is closely linked to the number of solutions visited before the time limit. In this way, algorithmics forms the engine of the search. Incremental algorithms, exploiting invariants in discrete structures, help to speed up the convergence of local search by several orders of magnitude (see for example the works of Katriel et al. [16] in the context of the Comet software [18]). Then, careful implementations, aware of the locality principle ruling the cache memory allocation and optimized by code profiling, still helps to accelerate local search (see for example the works done on SAT solvers [27]). From experience, it is not surprising to observe an order of 10 between the times of convergence of two local-search heuristics, apparently based on the same principles.

Linked to algorithmics, software and implementation aspects like *reliability* are no less crucial than efficiency. Because relying on complex incremental algorithmics and data structures, engineering local search requires larger efforts for verifying and testing than in traditional (business) software engineering. Hence,

the verification process of local-search softwares must be systematic. The first step is to program with assertions [26] (by verifying preconditions, postconditions, invariants all along the program); in particular, one must check at each iteration of the local search (in debugging mode) that the current solution satisfies the constraints of the problem and that its objective value is correct. But one step beyond, the consistency of all dynamic data structures must be checked (in debugging mode) after each iteration of the local search by recomputing them from scratch (with naive algorithms independent from the local-search code). Consequently, a large part of the source code (and of the time spent to implement) in local-search engineering projects must be dedicated to verification and testing: from experience, code checkers represent from 10 to 20 % of the whole source code. Reliability aspects (as well as maintainability and portability issues) must be imperatively taken into account for costing tightly local-search engineering projects.

Once these three levels have been completed, the resulting algorithm can be evaluated by computing statistics on target instances: success rate (number of acceptations over the number of attempts) and improvement rate (number of improvements in the sense of f over the number of attempts) for each move, number of iterations and time to reach best solutions. From experience, the quest for high performance requires many stepwise refinements, following the 80-20 rule (the last 20 % of improvement takes 80 % of the engineering time).

4 Description of the algorithm

4.1 The overall heuristic

The general heuristic is divided into four successive phases, each phase k consisting in planning interventions of priority k. The objective of one phase k is to minimize the ending date t_k of interventions with priority k, without degrading ending dates of interventions with priority $k' < k$. For this, a greedy algorithm completes the feasible solution inherited from the previous phase with interventions of priority k. Then, this solution is modified by local search in order to decrease t_k while maintaining ending dates $t_{k'}$ for each priority $k' < k$. Local search, which is used to pack a set of interventions of a given priority, is the critical routine of the overall heuristic.

More precisely, the local-search step for minimizing the ending date t_k is done as follows. Given one feasible solution with ending dates $t_1, t_2, \ldots, t_k$, a new feasible solution with ending dates $t_1, t_2, \ldots, t_k - 1$ is searched. During the search, an intervention is called infeasible if it is not completed before the ending date $t_k - 1$ or if the team of technicians to which this one is assigned does not own enough skills to complete it. In this way, the surrogate objective of local search is to minimize the number of infeasible interventions. When all are feasible, a new feasible solution is obtained and the process is iterated. This is an example of search strategy increasing density of the search space through constraint relaxation, as described in the methodology section.

A preprocessing phase was added (during the last days of the competition) to the overall heuristic in order to deal with compensations between the four terms of the original objective function (originally assumed to be unlikely according to FRANCE TÉLÉCOM organizers). Indeed, scheduling interventions of priority k before interventions of priority $k' < k$ may be advantageous according to the global objective function, due to the weakly discriminating coefficients (28, 14, 4, 1). This paradox induces an additional difficulty for which our heuristic was not prepared: to determine in which order the four priorities must be scheduled. Thus, a preprocessing phase was designed to "guess" this order. For this, interventions of each priority k are scheduled separately to determine an upper bound of their completion time c_k; concretely, this is done by the local-search routine in a short execution time (15 seconds for each priority, 1 minute on the whole). Then, the order kept to schedule priorities is the one which minimizes the original objective function, with ending dates t_k obtained by summing durations c_k. For example, assume that one have $c_1 = 1200$, $c_2 = 120$, $c_3 = 600$. The natural ordering of priorities induces the ending dates $t_1 = 1200$, $t_2 = 1320$, $t_3 = 1920$, which implies a cost of 59760. But, by inverting priorities 1 and 2, we obtain $t_1 = 1320$, $t_2 = 120$, $t_3 = 1920$, which implies a cost of 46320. Consequently, the order kept for the application of the overall heuristic will be: 2, 1, 3, 4. The experimental study presented at the end of the paper shows that the optimal ordering of priorities differs from the natural ordering $1, 2, 3, 4$ for more than the half of all instances. However, we do not linger more on this aspect of the problem, since our work was focused on the local-search routine.

4.2 The transformations

The local-search routine, employed to pack interventions of each priority, consists in applying stochastically some transformations to modify the current solution. We have defined two kinds of transformations, namely moves and swaps, applied on two kinds of objects, namely technicians or interventions. A transformation is accepted if the new solution respects the precedence constraints between interventions, the maximal working duration H in a day, and if the number of infeasible interventions is not increased.

Eight core transformations have been defined, which forms the engine of the local search:

- `MoveTechnician`, `SwapTechnicians`
- `MoveInterventionInterDays`, `SwapInterventionsInterDays`
- `MoveInterventionIntraDay`, `SwapInterventionsIntraDay`
- `MoveInterventionIntraTeam`, `SwapInterventionsIntraTeam`

The transformations applied to technicians consist in moving or swapping some technicians into a given day of the planning. The transformations applied to interventions consist in moving or swapping interventions of the planning; the suffixes `InterDays` (resp. `IntraDay`, `IntraTeam`) mean that interventions are moved or swapped between different days (resp. into a same day, into a same

team). Then, these 8 transformations have been specialized in order to increase their probability of success (it can be viewed as a refinement of neighborhoods which are explored). For each transformation, the three following declinations are defined:

- `Generic`: choose technicians (resp. interventions) randomly;
- `InfeasibleDay`: choose randomly a day among the ones containing an infeasible intervention and pick technicians working this day (resp. interventions performed this day) randomly;
- `InfeasibleTeam`: choose randomly a team among the ones containing an infeasible intervention and pick technicians working in this team (resp. interventions performed by this team) randomly.

Finally, additional transformations have been introduced to tackle the two possible extensions of the problem; namely, adding precedences between interventions, and allowing to abandon interventions within the limit of a budget.

- `AbandonInterventionBudget`: abandon an intervention of the planning (declined into `Generic` and `InfeasibleDay`);
- `SwapInterventionsBudget`: swap an abandoned intervention with a planned one (declined into `Generic` and `InfeasibleDay`);
- `ReinsertInterventionBudget`: reinsert an abandoned intervention into the planning;
- `SwapInterventionsPrecedences`: swap two interventions $I_i, I_{i'}$ such that $d_i \leq d_{i'}$ and the number of descendants of $I_{i'}$ in the precedence graph is greater than or equal to the one of I_i (declined into `InterDays` and `IntraDay`).

On the whole, a pool of 31 transformations is used. At each iteration of the heuristic, a transformation is picked randomly following a certain distribution. Here the convergence speed of the local search depends strongly of the utilization rate of each transformation. These rates have been fixed by hand after experimentations done with the first 20 benchmarks provided by France Télécom. Here is the outline of the distribution: (i) 25 % of `MoveInterventionInterDays` declined into `InfeasibleDay`, (ii) 25 % of `MoveTechnician` declined into `InfeasibleTeam`, (iii) 15 % of `SwapTechnicians` declined into `InfeasibleTeam`, and from 5 % to 1 % for the 28 other moves (if no budget is available, no budget-specific transformation is used; idem for precedences). The prominence of transformations (i), (ii), (iii) in the distribution is sensible: (i) is in charge of reinserting interventions making a day infeasible into another ones, whereas (ii) and (iii) are supposed to solve the infeasibility generated by lack of skills in teams. Note that, despite their low utilization rate, the 28 other moves participate to the diversification of the search.

4.3 Algorithmics & implementation

Applying a transformation follows this scheme: if the evaluation of the move is positive (evaluate), then the move is performed and all the incremental data

structures are updated (commit), else the incremental data structures are initialized (rollback). Since the number of attempted moves is generally much higher than the number of accepted moves, evaluate and rollback procedures are critical for the efficiency of the local search. The evaluation procedure is staged in order to stop early in case of rejection of the move; the different tests which are part of it are ordered according to their time complexity and their propensity to fail. For example, since the precedence constraints are considered as inviolable, all tests related to precedences in the evaluation process of moves `MoveIntervention` and `SwapInterventions` are done first. Since the evaluation process cannot be detailed for each of the 8 core transformations, we will only insist on two main points: the evaluation related to skills and the evaluation related to precedences.

Evaluation of skills. Any move which impacts the technicians or the interventions of a team calls for an evaluation of the adequation between skills provided by the technicians and skills required by the interventions of this team. To realize this evaluation, to each team of technicians is associated a matrix C_e of skills giving for each domain d and level l, the number of technicians of level at least l in the domain d. Then, an intervention I_i assigned to the team E_e is infeasible (according to skills) if a pair (d, l) exists such that $C_i(d, l) > C_e(d, l)$. Since the number of domains and levels is not bounded (for example, the instance B4 of benchmarks provided by France Télécom includes 40 domains), it is difficult to design a data structure more efficient than this matrix domain/level to evaluate skills. Consequently, evaluating the impact of a move on skills becomes time expensive in the worst case, because in $O(dl)$ time.

Fortunately, the number of cells of this matrix which are necessary to scan can be drastically reduced in practice. For example, the scan can be restricted to the useful domains of the matrix of skills required by the intervention, that is, the domains for which at least one technician is required. Then, for each useful domain d, the scan can be reduced to an interval of levels. Remind that our skill matrices are built cumulatively: for each domain, the number of technicians is non-increasing according to levels. Thus, the evaluation can start at the higher level l_{inf} such that $C_i(d, l_{\text{inf}}) = C_i(d, l)$ for all $l \leq l_{\text{inf}}$ and stop at the lower level l_{sup} such that $C_i(d, l_{\text{sup}}) = 0$.

Finally, a heuristic test with a lower time-complexity can be done before the scan of the matrix, in order to stop earlier in case of negative evaluation. For each domain d, define $C_e(d) = \sum_l C_e(d, l)$ and symmetrically $C_i(d) = \sum_l C_i(d, l)$. Then, the following necessary condition holds: if one domain d exists such that $C_i(d) > C_e(d)$, then I_i is infeasible (note that the reciprocal is trivially false). Such a test located upstream enables to determine in only $O(d)$ time the infeasible status of the intervention. In the same way, it is appropriate to place even before another test verifying if $C_i = \sum_d C_i(d)$ is strictly greater than $C_e = \sum_d C_e(d)$. Finally, the evaluation of skills is composed of three successive tests, respectively in $O(1)$ time, in $O(d)$ time, and in $O(dl)$ time, each one allowing to conclude in case of failure. Of course, all the structures involved in these tests must be maintained incrementally during the search.

Maintaining precedences. The second point concerns the evaluation of the ending dates $t_1, t_2, \ldots, t_k - 1$, and more generally the evaluation of the completion dates of the set of interventions assigned to each team. The computation of these values are complicated by precedences between interventions, because requiring to compute longest paths in a directed acyclic graph (DAG). For this, a DAG is attached to each day of the planning. Each DAG contains a source node representing the start of the day and a destination node representing its end. Then, to each intervention planned into the day is associated one node in the DAG. These nodes are linked by two kinds of precedences: blue arcs which induce the order of the interventions assigned to each team of technicians into the day, and red arcs which represent the precedences given in input. The length $l(i, i')$ of the arc connecting the nodes corresponding two interventions $I_i \prec I_{i'}$ is given by the duration $D(i)$ of the intervention I_i. In this way, the earliest starting date of one intervention is determined by the length of a longest path from the source node to its node into the DAG. This date, stored at each node, allows to verify if the maximal working duration H is respected for all teams, and to compute the ending dates $t_1, t_2, \ldots, t_k - 1$.

Thus, any transformation `MoveIntervention` or `SwapIntervention` implies a cascade of insertion/suppression of arcs into the DAG of impacted days, needing a (temporary) update of the longest paths in order to evaluate the impact of the transformation. Since the interventions of each team are completed sequentially, each node has only one blue predecessor and only one blue successor. The red predecessors and successors are stored as unordered lists into the data structure of the node. These lists, implemented as arrays, are designed to support basic routines (find, insert, delete, clear) in $O(1)$ time. Such a representation was motivated by the sparsity of the precedence graph on benchmarks A and B (where the number of red arcs is lower than the number n of interventions).

The temporary update of longest paths is done by a recursive bread-first propagation from the inserted/suppressed node. The new longest path at a node is computed by scanning its predecessors: if the new longest path is different from the old one, then the successors of the node are placed into a queue in order to be examined recursively. This propagation also enables to detect the creation of cycles, which makes the transformation rejected. When the maximum degree of the DAG remains in $O(1)$, which is the case here, our incremental algorithm (evaluate, commit and rollback procedures) runs in optimal time and space $O(a)$ with a the number of affected nodes (that is, having a modified longest path). The interested reader can consult the works of Katriel et al. [16] on the subject, which give an incremental algorithm whose complexity becomes advantageous when the maximum in-degree of a node is large.

An implementation detail. As claimed in introduction, every choice made during the search follows stochastic rules, in order to avoid bias and to enforce diversification. Then, a number of choices are made before applying each single move. On average, the function `MyRand(n)`, which returns a pseudo-random integer value between 0 and $n-1$, is called 5 times per attempted move. For example,

the transformation `MoveInterventionInterDays` declined into `InfeasibleDay` (which represents 25 % of attempted moves) uses it 6 times. `MyRand` is in fact the portion of code which is the most called into our program (more than 10 billion of calls over 20 minutes of running time).

A direct implementation (in ISO C programming language) of `MyRand(n)` is `n * rand() / (RAND_MAX + 1.0)` [23, p. 277], where `rand()` is a function of the `stdlib` library returning a pseudo-random integer between 0 and the largest positive `int`-type number. Although providing pseudo-random integer sequences of sufficient quality for our application, a profiling of our program with *gprof* [7] pointed `MyRand` as the main bottleneck for running time. Inspired by the Knuth-Lewis generator [23, pp. 283–286], we have engineered a quick `MyRand(n)` function dedicated to our needs: `(n * ((seed = 1664525 * seed + 1013904223) >> 16)) >> 16`, which is correct if n is between 0 and $2^{16} - 1 = 65535$ and if the `int` type is encoded on 32 bits (the traditional `seed` of the generator is initialized at the beginning of the program).

Experimentations on different computing platforms have shown that this concise implementation is at least 3 times faster than the direct implementation. The period of the generator is of length $2^{32} > 4 \times 10^9$, which is comparable to the one of `rand()` and remains sufficient in this context (from experience, the quality of the pseudo-random number generation is not highly critical for simulating randomness in local search). This enables us to reduce the part of running time spent in `MyRand` from 17 % to 7 %, lowering it to the levels of the other time-consuming functions of the program (the 3 functions appearing just after `MyRand`, which are parts of the evaluation process, consume each one nearly 5 % of the total running time).

5 Experimental results

The whole algorithm was implemented in C programming language (ISO C99). The resulting program, which includes nearly 12000 lines of code, was compiled and tested on several computing platforms with comparable performance (Red Hat Linux/AMD Athlon 64, Windows XP/Intel Pentium 4, Windows XP/Intel Xeon, Windows Vista/Intel Xeon 64) using the free compiler `GCC 3.4.4` with options `-O3 -pedantic -Wall -W -std=c99`}. Note that nearly 10 % of the source code is dedicated to the verification of the program.

The benchmarks A, B, X provided by France Télécom and used for tests can be downloaded on the web page of the Challenge [25] (the set X, used to rank the competitors, was unveiled once the final results proclaimed). On each tested platform, our local-search algorithm *attempts more than 1 million moves per second*, even for large-scale instances (for example instance B8: 800 interventions, 150 technicians, 10 domains and 4 levels for skills, 440 precedences. Over 20 minutes of running time (which is the maximum allowed for the competition), the heuristic *visits more than 1 billion solutions* into the surrogate search space. The average success rate of transformations (that is, the number of accepted transformations divided by the number of attempted ones) varies between 10

and 60 % according to the instances. The memory allocated by the program does not exceed 10 Mo for any instance of the benchmarks (for example, 8 Mo of memory are allocated for B8 instance), allowing a full exploitation of the cache memory. Table 1 reports the results obtained on a computer equipped with a Windows XP operating system and a chipset Intel Xeon 3075 (CPU 2.67 GHz, L1 cache 64 Kio, L2 cache 4 Mio, RAM 2 Go). An executable binary file (compiled for the desired computing architecture) is available on request from the authors.

data	n	m	d	l	P	B	FT	EGN	BEST	gap	priority	attempt	accept	improve
A1	5	5	3	2	0	0	2490	2340	2340	0.0 %	1234	8696 M	1260 M	2
A2	5	5	3	2	2	0	4755	4755	4755	0.0 %	1234	4626 M	1530 M	2
A3	20	7	3	2	0	0	15840	11880	11880	0.0 %	2134	4262 M	1178 M	3
A4	20	7	4	3	7	0	14880	14040	13452	4.4 %	1234	4558 M	1047 M	80
A5	50	10	3	2	13	0	41220	29400	28845	2.0 %	2134	5203 M	951 M	273
A6	50	10	5	4	11	0	30090	18795	18795	0.0 %	2134	4861 M	1163 M	225
A7	100	20	5	4	31	0	38580	30540	29690	2.9 %	1234	4968 M	892 M	669
A8	100	20	5	4	21	0	26820	20100	16920	18.8 %	1234	4958 M	1176 M	1014
A9	100	20	5	4	22	0	35600	27440	27440	0.0 %	2134	5081 M	877 M	1166
A10	100	15	5	4	31	0	51720	38460	38296	0.5 %	1234	5689 M	707 M	577
B1	200	20	4	4	47	300	69960	33900	33675	0.7 %	1234	4453 M	1012 M	833
B2	300	30	5	3	143	300	34065	16260	15510	4.9 %	1234	4259 M	945 M	1195
B3	400	40	4	4	57	500	34095	16005	15870	0.9 %	1234	3722 M	825 M	1830
B4	400	30	40	3	112	300	50340	24330	23700	2.7 %	2134	2485 M	604 M	604
B5	500	50	7	4	427	900	150360	88680	87300	1.6 %	1234	3344 M	1520 M	612
B6	500	30	8	3	457	300	47595	27675	27210	1.8 %	2134	4437 M	616 M	1534
B7	500	100	10	5	387	500	56940	36900	33060	11.7 %	1234	2867 M	1544 M	643
B8	800	150	10	4	440	500	51720	36840	32160	14.6 %	1234	2927 M	1513 M	1036
B9	120	60	5	5	55	100	44640	32700	28080	16.5 %	1234	3853 M	1470 M	697
B10	120	40	5	5	55	500	61560	41280	34440	19.9 %	1234	3704 M	1499 M	565
X1	600	60	15	4	195	50	n/a	180240	151140	19.3 %	1234	2622 M	1136 M	546
X2	800	100	6	6	536	500	n/a	8370	7260	15.3 %	1234	2764 M	962 M	2712
X3	300	50	20	3	224	1000	n/a	50760	50040	1.5 %	1234	2458 M	1464 M	888
X4	800	70	15	7	321	150	n/a	68960	65400	5.5 %	2134	3383 M	623 M	2015
X5	600	60	15	4	201	50	n/a	178560	147000	21.5 %	1234	2551 M	1222 M	599
X6	200	20	6	6	128	500	n/a	10440	9480	10.2 %	1234	3573 M	1051 M	487
X7	300	50	20	3	235	1000	n/a	38400	33240	15.6 %	1234	2533 M	1405 M	527
X8	100	30	15	7	40	150	n/a	23800	23640	0.7 %	1234	2712 M	1330 M	327
X9	500	50	15	4	184	50	n/a	154920	134760	15.0 %	1234	2541 M	1156 M	522
X10	500	40	15	4	184	500	n/a	152280	137040	11.2 %	1234	2739 M	1183 M	546
average										7.3 %		3894 M	1129 M	790

Table 1. Benchmarks A, B, X: characteristics and results (M = million).

The characteristics of each instance are given on the left part of the table: the number n of interventions, the number m of technicians, the number d of skill domains, the number l of skill levels, the number P of (non transitive) precedences between interventions, the budget B available. For each instance, 5 runs were performed, each one limited to 1200 seconds (20 minutes). In the middle

part of the table, the columns "FT", "EGN", "BEST", "% gap", "priority" contain respectively the result obtained by France Télécom's algorithm, the worst result obtained by our algorithm (over the 5 runs), the best result obtained among all the competitors (including the 5 runs of our algorithm), the relative gap (in %) between the values of the two previous columns, and the ordering of priorities used by the EGN algorithm (for example, the value 3214 means that the priorities were scheduled according to the ordering 3, 2, 1, 4). In the right part of the table, the column "attempt" (resp. "accept", "improve") reports the average number of attempted transformations (resp. accepted transformations, strictly improving transformations).

A weak gap is observed between the results of the 5 runs of our algorithm (that is why only the worst result is given here). Note that this gap increases with the number of planned days. Thus, gaps greater than 1 % between runs are observed for the following instances: X1 (57 days), X5 (52 days), X9 (50 days), X10 (49 days). Then, the relative gap between the results of our algorithm and the best results of the Challenge shows that this one is very competitive. On average, EGN algorithm reduces by 30 % the cost of the solutions proposed by France Télécom (and by 41 % for the sole benchmark B). On the other hand, the gap between our solutions and the best solutions obtained among all competitors is of 7.3 % on average (with a standard deviation of 7.5 %). On the 30 instances, our algorithm obtains the best solution for 13 ones (7 for A, 6 for B, 3 for X) and obtains a solution having a cost lower than 6 % of the cost of the best solution for 18 instances (9 for A, 6 for B, 3 for X).

data	EGN	EGN*	BEST	% gap	priority
A5	29700	28845	28845	0.0	3214
A8	20100	16979	16979	0.0	2134
B7	36900	35700	33300	7.3	2134
B9	32700	28080	28080	0.0	2134
B10	41280	34440	34440	0.0	2314
X2	8370	7440	7260	2.5	2134
X6	10440	10140	9480	7.0	2134
X7	38400	32280	32280	0.0	2134
X8	23800	23220	23220	0.0	2134

data	20 min	1 hr	3 hrs	9 hrs
X1	180240	170460	168240	158280
X5	178560	167280	165120	164760
X9	154920	146520	146040	141720
X10	152280	144360	140340	140160

Table 2. Results with optimal priority ordering (left) or extended time limits (right).

Besides, we are able to explain why EGN algorithm fails to find the best solution for the 17 remaining instances. The main reason is that the ordering of priorities computed in the preprocessing stage is not the most appropriate. The table on the left part of Table 2 shows the cost obtained by our algorithm assuming that the optimal ordering is known. This cost appears in the column named "EGN*" and the optimal ordering appears in the column named "priority". In this case, one can observe that for 6 more instances we obtain the best solution. The second reason is still due to the multi-objective nature of the cost

function. For example, for instance A4, EGN algorithm obtains the following solution: $t_1 = 315$, $t_2 = 540$ and $t_3 = 660$ with global cost 14040. Now, relaxing slightly the ending date of interventions with priority 1 allows to improve the global cost thanks to the compensation of the two first terms of the objective function: $t_1 = 324$, $t_2 = 480$ and $t_3 = 660$ with global cost 13452 (best known solution).

However, our local-search approach is overcome on instances X1, X5, X9, X10 by large neighborhood search approaches of Hurkens [14], winner of the Challenge, and to a lesser extend, of Cordeau et al. [4] ranked second ex æquo. In fact, these instances contain in majority long interventions (of length 60 or 120) requiring many technicians, which reduces considerably the combinatorics induced by the assignment of interventions to teams and then allows integer programming approaches for tackling subproblems. To make up for this weakness, it seems therefore appropriate to add some moves with larger neighborhoods to our pool of transformations (as done in [5] for car sequencing problems). A first simple idea is to implement (k, l)-swap transformations, consisting in exchanging k interventions with l other ones (here only $(1, 1)$-swaps are done). The table on the right part of Table 2 gives results obtained for these 4 instances with extended time limits, showing that our algorithm converges toward a solution of quality near from the ones of Hurkens and Cordeau et al. [25].

References

1. R.K. Ahuja, Ö. Ergun, J.B. Orlin, A.P. Punnen (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, pp. 75–102.
2. E. Aarts, J.K. Lenstra, eds (1997). *Local Search in Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester, England, UK.
3. T. Benoist, B. Estellon, F. Gardi, A. Jeanjean (2009). High-performance local search for solving real-life inventory routing problems. (to appear in *SLS 2009, the 2nd International Workshop on Engineering Stochastic Local Search Algorithms*)
4. J.-F. Cordeau, G. Laporte, F. Pasin, S. Ropke (2007). ROADEF 2007 Challenge: scheduling of technicians and interventions in a telecommunications company. In *ROADEF 2007, le 8ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (G. Finke, ed.). Grenoble, France. (in French)
5. B. Estellon, F. Gardi, K. Nouioua (2006). Large neighborhood improvements for solving car sequencing problems. *RAIRO Operations Research* 40(4), pp. 355–379.
6. B. Estellon, F. Gardi, K. Nouioua (2008). Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research* 191(3), pp. 928-944.
7. J. Fenlason, R. Stallman (1998). *GNU gprof: the GNU profiler.* `http//www.gnu.org/software/binutils/`
8. M.R. Garey, D.S. Johnson (1979). *Computer and Intractability: a Guide to the Theory of NP-Completeness.* W.H. Freeman & Co, San Francisco, CA.
9. F.W. Glover, G.A. Kochenberger (2002). *Handbook of Metaheuristics.* International Series in Operations Research and Management Science, Vol. 57, Kluwer Academic Publishers, Norwell, MA.

10. P. Hansen, N. Mladenović, J.A. Moreno Pérez (2008). Variable neighborhood search: methods and applications. *4OR* 6(4), pp. 319-360.
11. K. Helsgaun (1998). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *Datalogiske Skrifter (Writings on Computer Science)*, Technical Report No. 81. Roskilde University, Danemark.
12. K. Helsgaun (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126(1), pp. 106–130.
13. K. Helsgaun (2006) An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic. *Datalogiske Skrifter (Writings on Computer Science)*, Technical Report No. 109. Roskilde University, Danemark.
14. C.A.J. Hurkens (2007). Incorporating the strength of MIP modeling in schedule construction. In *ROADEF 2007, the 8th Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (G. Finke, ed.). Grenoble, France.
15. K. Jansen, G.J. Woeginger, Z. Yu (1995). UET-scheduling with chain-type precedence constraints. *Computers and Operations Research* 22(9), pp. 915–920.
16. I. Katriel, L. Michel, P. Van Hentenryck (2005). Maintaining longest paths incrementally. *Constraints* 10(2), pp. 159–183.
17. A. Løkketangen (2007). The importance of being careful. In *Proceedings of SLS 2007, the 1st International Workshop on Engineering Stochastic Local Search Algorithms*, *LNCS* 4638, pp. 1–15. Springer-Verlag, Heidelberg, Germany.
18. L. Michel, P. Van Hentenryck (2002). A constraint-based architecture for local search. In *Proceedings of OOPSLA 2002, the 2002 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, *SIGPLAN Notices* 37(11), pp. 83–100. ACM Press, New York, NY.
19. M. Minoux (2008). *Programmation mathématique : théorie et algorithmes*. Éditions Tec & Doc, Lavoisier, Paris. (2nd edition, in French)
20. B.M.E. Moret (2002). Towards a discipline of experimental algorithmics. In *Data Structures, Near Neighbor Searches, and Methodology: 5th and 6th DIMACS Implementation Challenges* (M.H. Goldwasser, D.S. Johnson, C.C. McGeoch, eds.), DIMACS Monographs, Vol. 59, pp. 197–213. American Mathematical Society, Providence, RI.
21. B.M.E. Moret, D.A. Bader, T. Warnow (2002). High-performance algorithm engineering for computational phylogenetics. *Journal of Supercomputing* 22(1), pp. 99–111.
22. P. Pellegrini, M. Birattari (2007). Implementation effort and performance. In *Proceedings of SLS 2007, the 1st International Workshop on Engineering Stochastic Local Search Algorithms*, *LNCS* 4638, pp. 31–45. Springer-Verlag, Heidelberg, Germany.
23. W.H. Press, S.A. Tenkolsky, W.T. Vetterling, B.P. Flannery (1995). *Numerical Recipes in C: the Art of Scientific Computing*. Cambridge University Press, Cambridge, Royaume-Uni. (2nd edition)
24. ROADEF 2005 Challenge: `www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2005/`
25. ROADEF 2007 Challenge: `www.g-scop.fr/ChallengeROADEF2007/`
26. D.S. Rosenblum (1992). Towards a method of programming with assertions. In *Proceedings of ICSE 1992, the 14th International Conference on Software Engineering*, pp. 92–104. ACM Press, New-York, NY.
27. L. Zhang, S. Malik (2003). Cache performance of SAT solvers: a case study for efficient implementation of algorithms. In *Proceedings of SAT 2003, the 6th International Conference on Theory and Applications of Satisfiability Testing* (S.M. Ligure, ed.), pp. 287-298.