Efficient Greedy Algorithm with Hill Climbing for Technicians and Interventions Scheduling Problem

Wojciech Jaśkowski and Szymon Wąsik

{wjaskowski|swasik}@cs.put.poznan.pl

1 Introduction

We present a greedy algorithm for the Technicians and Interventions Scheduling Problem described in [1]. Our method consists of several parts related to urgency classes of interventions. In each part the algorithm tries various intervention sorting strategies in order to obtain the best partial solution. Our method incorporates also a hill climbing local search metaheuristic.

The algorithm described in this paper finds significantly better solutions on the test sets than the reference ones[1]. Moreover, our algorithm is efficient and its simplified version finds very good results rapidly.

2 Method

Our method is driven by a greedy procedure (see Algorithm 1). It processes interventions one by one trying to assign them to teams as early as possible. Above that, it iterates over the intervention urgency classes and several possible sorting strategies. At the top level, different permutations of intervention priorities are considered. Finally, the algorithm tries to improve the current partial solution S by applying a hill climbing metaheuristic and by abandoning some interventions. The improve procedure is called for the second time after the abandon routine.

In addition, to the notation introduced in [1], let us define:

- $Succ(I)$ as the set of direct successors of I and
- $Succ^{+}(I)$ as the set of *all* (direct and indirect) successors of I.

It is easy to notice that it is not always optimal to place the interventions with the lowest priority at the beginning of the schedule. In simple words, it depends on the trade-off between the weights in the scoring procedure and the distribution of intervention priorities. Therefore, let us abstract from intervention priorities by defining the *intervention class*:

$$Class(I) = Perm\left(Prio(I)\right), \tag{1}$$

where $Perm(x)$ is a certain permutation function of priorities $\{1, 2, 3, 4\}$.

In order to analyze all possible orderings of priority classes, the whole algorithm is executed for six different permutations $Perm$: (1,2,3,4), (1,3,2,4), (2,1,3,4), (2,3,1,4), (3,1,2,4) and (3,2,1,4). Eventually, the final solution is the best obtained. Notice that the last priority never changes, because the scoring function is defined in such a way that it does not take into account the finish time of interventions of priority 4.

One can observe that, in general, it is profitable to finish processing interventions from a certain class before starting processing interventions from another class. The dependencies between interventions can, however, make this process impossible. In order to divide the set of interventions into smaller class-related groups, we define the *urgency class* of the intervention I:

$$Urg(I) = \min_{I' \in \{I\} \cup Succ^{+}(I)} Class(I'). \tag{2}$$

The concept of *urgency class* allow to apply a greedy approach to finish processing interventions with urgency class u before starting interventions with urgency $v > u$.

[1] Both the reference results and the test sets were provided by ROADEF 2007 Challenge organizers.

Algorithm 1 The pseudo-code of our algorithm. S, S^1, S^2, S^3 denote (partial) solutions.

```
procedure solve()
  S ← ∅
  for each priority permutation Perm
    S^1 ← ∅
    for each urgency class U
      S^2 ← ∅
      for each sorting strategy Q {
        S^3 ← S^1
        sort all I ∈ U, using Q
        for each I ∈ U {
          A ←find best assignment for I
          S^3 ← S^3 ∪ {A}
        }
        improve S^3
        abandon some I ∈ U from S^3
        improve S^3
      }
      S^2 ← better(S^2, S^3)
    }
    S ← better(S, S^1)
  }
  return S
```

Within one urgency class, it is crucial to design a good interventions sorting strategy that determine which not yet assigned intervention should the greedy algorithm process as the first one. In practical examples, the bottleneck of the problem instance is often the longest path in the graph of dependencies between interventions of the same urgency. Let $Succ^*(I)$ be the set of direct successors of I with the same urgency as I:

$$Succ^*(I) = \{I' \in Succ(I) \,|\, Urg(I') = Urg(I)\} \tag{3}$$

We define *rank*[2] of intervention I as

$$Rank(I) = \max_{I' \in Succ^*(I)} (Rank(I')) + 1, \tag{4}$$

and the *delay* of intervention I as

$$Delay(I) = \max_{I' \in Succ^*(I)} (Delay(I')) + T(I). \tag{5}$$

Both $Delay(I)$ and $Rank(I)$ are measures of the length of the longest path from I to any other intervention of the same urgency. Although, $Delay(I)$ is supposed to be more precise, our preliminary experiments have shown that sometimes the use of $Rank(I)$ leads to better solutions. Therefore, we use two sorting strategies: descending by the intervention rank and descending by the intervention delay. Notice also that both sorting strategies ensure that all predecessors of I are processed before I.

When $Rank(I_1) = Rank(I_2)$, the second criterion of comparison is the *intervention importance* defined as

$$Imp(I) = \sum_i \sum_n R(I, i, n) \times Signif(i, n), \tag{6}$$

where

$$Signif(i, n) = Req(i, n)/Avail(i, n) \tag{7}$$

is the *significance* of domain i and level n; $Req(i, n)$ and $Avail(i, n)$ are defined as follows:

[2] We assume that $max_{x \in X} f(x) = 0$ when X is empty.

$$Req(i,n) = \sum_{I} Prio(I) \times T(I) \times R(I,i,n), \tag{8}$$

$$Avail(i,n) = \sum_{t} C(t,i). \tag{9}$$

The greater $Signif(i,n)$, the more technicians in domain i and level n is required with reference to technicians available in domain i and level n. $Imp(I)$ biases the algorithm to process earlier the interventions that are likely to be bottlenecks due to their domain/level requirements. $Imp(I)$ is used also as a tie-breaker when $Delay(I_1) = Delay(I_2)$.

In order to assign a team to intervention I, our algorithm iterates over consecutive days of the current solution in order to find a suitable team for the intervention. It stops on the day when it is possible to either (i) assign I to an existing team t or (ii) assign I to a new team t. In both cases t must meet the domain/level requirements of I. The algorithm ensures also that the dependency restrictions of the interventions are satisfied. I is assigned to t at the earliest possible moment.

When, at certain day, I is to be assigned to a new team (ii), the algorithm uses a greedy approach to choose the best subset from the set of free technicians that day. At first, the algorithm, creates a team with all the free technicians. Then, it tries to remove the technicians from the team, one by one. The final team t is minimal in the sense that if any technician from t is removed, t does not longer meet the requirements of I. Since some technicians are more valuable than the others and it is profitable to save them, the most valuable technicians are tested first. The value of technicians is defined as similarly as the importance of interventions:

$$Val(t) = \sum_{i} C(t,i) \times Signif(i, C(t,i)), \tag{10}$$

where $Singif(i,n)$ was defined in Eq. 7.

We have also explored the possibility to use a solver, instead of the greedy algorithm, to choose the best subset from the set of free technicians for certain I, but it did not improve the results, on average. Thus, due to its high computation time costs, we did not use it in the final algorithm.

The process of abandoning interventions is also a greedy one. For certain urgency class U, our algorithm abandons as many interventions $I \in U$ from the end of the current scheduling S as lead to the better evaluation of S. It ensures that the total budget A is not exceeded. Abandoning intervention I carry with it the need to abandon all $I' \in Succ^{+}(I)$.

The improvement phase of our algorithm consists of two hill climbing metaheuristics, which aim to make the current (partial) solution S better. The first one tries to free some technicians that are currently assigned. It finds a pair of interventions I_1, I_2 (assigned to teams t_1, t_2) that could be swapped without breaking any dependency restrictions. I_1 and I_2 are swapped when such a move would cause that at least one of the teams t_1, t_2 is not minimal anymore. The superfluous technicians are excluded from the team. The heuristic works as long as it can free any technician.

The second heuristic tries to compress the current solution. When it finds an intervention that could be assigned earlier by any existing team or a new team, then it move that intervention. The heuristic works as long as it is possible to assign an intervention earlier. When both metaheuristics can not make any move, the improvement phase ends.

In addition to the techniques described above, we designed a technique of connecting similar interventions. I_1 and I_2 are considered similar, if the total difference between their domain/level requirements are lower than a certain constant ε, i.e., when

$$\sum_{i} \sum_{n} |R(I_1,i,n) - R(I_2,i,n)| < \varepsilon. \tag{11}$$

Obviously, to connect I_1 and I_2, $T(I_1)+T(I_2) \leq H_{max}$ must be satisfied. Our algorithm connects only interventions that have no dependencies. Preliminary experiments have shown that, generally, connecting is a useful technique, but for $\varepsilon > 7$ it always produces worst results. Therefore, we chose the best solution produced by algorithms with $\varepsilon = 0..7$. Notice that, for $\varepsilon = 0$, no connection occurs. The total improvement of the connection procedure against the pure algorithm is about 1.5%. The connection procedure was not shown in the Algorithm 1 for clarity.

3 Results

In the Table 1 we present results of our greedy approach and the reference ones provided by ROADEF 2007 Challenge organizers. The column '% diff' shows the relative percentage difference computed as $(x - x^*)/x^*$, where x^* is a reference result whereas x is our result. Clearly, our results are in all cases not worst then the reference ones and significantly better in the vast majority of them. In set A, our algorithm, on average, outperforms the reference results by 16.1%. Set B contains much harder and bigger instances and the superiority of our greedy algorithm is even more evident — 36.0%, on average.

Table 1. The comparison of results.

	set A				set B			
#	reference	our	% diff	time [s]	reference	our	% diff	time [s]
1	2490	2490	0.0%	0.2	69960	46995	32.8%	27.7
2	4755	4755	0.0%	0.3	34065	19890	41.6%	46.1
3	15840	12600	20.5%	0.9	34095	20340	40.3%	87.5
4	14880	14040	5.6%	1.0	50340	29460	41.5%	365.5
5	41220	32400	21.4%	3.1	150360	100080	33.4%	242.2
6	30090	21120	29.8%	3.0	47595	34230	28.1%	182.4
7	38580	32520	15.7%	9.0	56940	36060	36.7%	269.7
8	26820	19380	27.7%	8.5	51720	35550	31.3%	550.4
9	35600	28280	20.6%	7.1	44640	29460	34.0%	16.3
10	51720	41580	19.6%	9.0	61560	36960	40.0%	14.6
avg			16.1%				36.0%	

It is important to note, that our algorithm has low time complexity. The complexity of the greedy part is limited by $O(ITLN)$, where I, T, N, L are the number of interventions, number of technicians, number of domains and number of levels, respectively. Hill climbing metaheuristics have much higher worst-case complexity. Each their run is limited by $O(I^2TLN)$. While theoretically there can be at most I runs, in practice it happens very rarely to be than one. An experiment with randomly generated instances, that we have performed, suggests that the average complexity of the whole algorithm is $O(I^2TLN)$.

The time results presented in Table 1 concern the algorithm with the connecting extension enabled and all the sorting strategies used. The simplified version of the algorithm with disabled the connection loop, use only the sorting strategy with the $Delay()$ measure, performs only a little bit worst then the original one, being significantly faster at the same time (See Table 2). We believe that such a fast algorithm would be convenient for the operator in a telecommunication company when the time of scheduling process could be also a significant issue.

Table 2. The results for the simplified algorithm.

	set A			setB		
#	simplified greedy	% diff	time [s]	simplified greedy	% diff	time [s]
1	2625	-5.4%	0.00	57240	18.2%	0.18
2	4755	0.0%	0.01	23820	30.1%	0.39
3	13860	12.5%	0.02	23865	30.0%	0.70
4	15300	-2.8%	0.01	34860	30.8%	5.88
5	34380	16.6%	0.03	126960	15.6%	1.42
6	24855	17.4%	0.03	35340	25.7%	1.35
7	33360	13.5%	0.08	36660	35.6%	1.29
8	21465	20.0%	0.16	35550	31.3%	2.60
9	30075	15.5%	0.06	29880	33.1%	0.10
10	44940	13.1%	0.08	36960	40.0%	0.10
avg		10.0%			29.0%	

An examplary solution produced by our algorithm for the instance 8 from set A was shown shown in Figure 1.

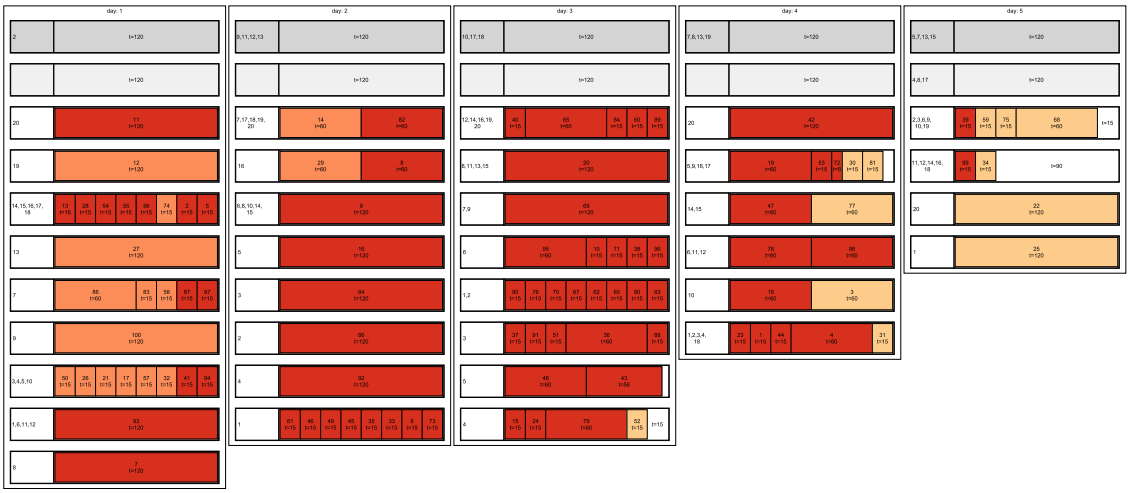

Fig. 1. A scheduling produced by our algorithm for instance 8 from set A. Each column represents one day. Each rectangular block is a team (the number of technicians on the left). The grey team is a team of technicans off work, whereas white teams group technicians that are free. Color rectangles represent interventions. The darker the color, the lower the priority (red for priority 1).

4 Conclusions

We presented a deterministic greedy algorithm that uses a hill climbing local search for improving partial solutions. The results of our algorithm are significantly superior to the reference results. The algorithm is simple and easy to implement. The simplified version of the algorithm, while still performing very well, is characterized by much higher time performance.

References

1. Dutot, P.F., Laugier, A., Bustos, A.M.: Technicians and interventions scheduling for telecommunications. (2007)