A hybrid multi-objective/goal programming approach for timetabling

João Pedro Pedroso

Dep. Ciência Computadores, Fac. Ciências, Universidade do Porto
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal
jpp@ncc.up.pt

1 Introduction

We propose a goal programming, multi-criteria approach for the solution of timetabling problems. The problem is to find a time slot and a room for each element of a set of events $\mathcal{E}$, such that a set of constraints $\mathcal{C}$ is satisfied "as much as possible", and in a predefined order.

Given the set of events $\mathcal{E}$, a set of time slots $\mathcal{T}$ and a set of rooms $\mathcal{R}$, the search space is $\mathcal{S} = (\mathcal{E} \times \mathcal{T} \times \mathcal{R})$. Any solution $s \in \mathcal{S}$ is considered feasible, i.e., all the constraints are considered soft. Each goal is defined by one or more constraints. Let $\mathcal{C}_i$ be the subset of constraints defining goal i, and $v_i(x)$ the amount of violation of constraint i of a solution $x \in \mathcal{S}$. Then, goal i concerns the minimization, for all the constraints in $\mathcal{C}_i$, of the sum of the weighted violation

$$g_i(x) = \sum_{k \in \mathcal{C}_i} w_k v_k(x).$$

We will denote $g(x)$ as the N-dimensional vector of goals, $g(x) = (g_1(x), \ldots, g_N(x))$.

Let the first goal be $g_1^* = \min\{g_1(x), \forall x \in \mathcal{S}\}$. The second goal is then $g_2^* = \min\{g_2(x), \forall x \in \mathcal{S} : g_1(x) = g_1^*\}$. If there are N goals, the objective of the problem is then to obtain:

$$g_N^* = \min\{g_N(x), \forall x \in \mathcal{S} : g_{N-1}(x) = g_{N-1}^*, \ldots, g_1(x) = g_1^*\}$$

An additional assumption made is that there are no constraints violated on an empty solution, and that the number of violations in general increase through the steps of construction of a solution.

2 Solution classification

During the search, we will classify solutions $x \in \mathcal{S}$ according to the corresponding value of the N goals. For two solutions $x, y \in \mathcal{S}$, we define a precedence relation between them as:

$$
\begin{aligned}
g(x) \prec g(y) \quad \Leftrightarrow \quad & g_1(x) < g_1(y); \\
& g_1(x) = g_1(y) \text{ and } g_2(x) < g_2(y); \\
& \ldots \\
& g_1(x) = g_1(y),\ g_2(x) = g_2(y),\ \ldots,\ g_{N-1}(x) = g_{N-1}(y) \text{ and } g_N(x) < g_N(y).
\end{aligned}
$$

This relation will be used to compare solutions; notice that it can be used both for the case of complete solutions and the case of partially constructed solutions. For the latter case, the condition for this relation to make sense is that the number of unplaced events on the two solutions being compared is the same.

We define a function $G(g)$ which returns the order k from which the goals on the vector g_k and g_k^* are different (hence $G(g) = 1$ means that $g_1(g) \neq g_1^*$, and $G(g) = 3$ means that $g_1 = g_1^*,\ g_2 = g_2^*,\ g_3 \neq g_3^*$, for example).

3 Algorithm

The algorithm proposed is a GRASP variant, where solutions are constructed in a semi-greedy way, and then bound into a local optimum. After that—as opposed to traditional GRASP [6], where a new construction starts—the best found solution is partially destroyed, also in a semi-greedy way, until the first unsatisfied goal becomes satisfied. From that partially destroyed solution, a new semi-greedy construction starts, followed by local search.

The aim of construction/destruction cycles is to provide a good balance between intensification and diversification, as suggested in [2], for example.

The algorithm stops when the CPU time (or the iteration count) limit is reached.

Algorithm 1: The REGRASP algorithm.

```
REGRASP()
(1)    x = x* = {}
(2)    while stopping criterion is not satisfied
(3)       x = SemiGreedy(x)
(4)       x = LocalSearch(x)
(5)       if x* = {} or g(x) ≺ g(x*)
(6)          x* = x
(7)       x = SemiUnGreedy(x*)
(8)    return x*
```

3.1 Semi-greedy construction and destruction

On the construction procedure (Algorithm 2) we check the best placement (slot and room) for each event that has not yet been assigned (lines 4 to 8). We update the goal evaluation for

the best and worst events that can be assigned (lines 9 to 12). Then, from the events that are to be assigned, one is selected in a semi-greedy way (lines 13 to 18). For this selection, only events that have the same level of goal satisfiability k of the best classified event may enter the restricted candidate list (RCL).

Algorithm 2: Semi-greedy solution construction.

```
SemiGreedy(x)
(1)   U = set of events with no attributed room/slot on x
(2)   while U ≠ {}
(3)     foreach e ∈ U
(4)       for s = 1 to NS
(5)         for r = 1 to NR
(6)           x̄ = x ∪ {(e, s, r)}
(7)           if g* not initialized or g(x̄) ≺ g*
(8)             s*[e] = s; r*[e] = r; g*[e] = g(x̄)
(9)       if g^min not initialized or g*(e) ≺ g^min
(10)        g^min = g*[e]
(11)      if g^max not initialized or G(g^max) = G(g*[e]) and g^max ≺ g*[e]
(12)        g^max = g*[e]
(13)    foreach e ∈ U
(14)      α = rand[0,1] k = G(g*[e])
(15)      if k = G(g^min) and g*_k[e] < g^min_k + α (g^max_k − g^min_k)
(16)        RCL = RCL ∪ {e}
(17)    e = RandomChoice(RCL)
(18)    x = x ∪ {(e, r*[e], s*[e])}
(19)    U = U\{e}
(20)  return x
```

The destruction procedure (Algorithm 2) selects, from the events that can be removed from the solution, those that lead to greatest improvements in the solution, in a semi-greedy way. This is done by checking, for all the events in the solution, what is the partial objective if the event is removed, and updating the best and worst removal possibilities (lines 4 to 11).

Then, from the events assigned, one is selected in a semi-greedy way (lines 12 to 17). As for the construction case, for this selection only events that have the same level of goal satisfiability k of the best classified event may enter the restricted candidate list (RCL).

3.2 Neighborhoods and local search

We propose two neighborhoods for local search. For a given solution x, the first neighborhood considered, $N_1(x)$ consists of the solutions that can be obtained from x by changing the slot and/or room of each of the events.

The second neighborhood, $N_2(x)$ consists of the solutions that can be obtained from x by exchanging the slot and room of an event with the slot and room of another.

The neighborhoods are explored using ideas borrowed from a strategy called variable neigh-

Algorithm 3: Semi-greedy solution destruction.

SemiUnGreedy(x)
(1) A = set of events with attributed room/slot on x
(2) $k = G(g(x))$
(3) **while** $g_k^* < g_k(x)$
(4) **foreach** $e \in A$
(5) $\bar{x} = x \backslash \{e\}$)
(6) **if** g^{min} not initialized **or** $g(\bar{x}) \prec g^{min}$
(7) $g^{min} = g(\bar{x})$
(8) **if** $G(g(\bar{x})) < k$
(9) $k = G(g(\bar{x}))$; $g^{max} = g(\bar{x})$
(10) **if** g^{max} not initialized **or** $G(g(\bar{x})) = k$ **and** $g^{max} \prec g(\bar{x})$
(11) $g^{max} = g(\bar{x})$
(12) **foreach** $e \in A$
(13) $\alpha = \text{rand}[0,1]$ $\bar{x} = x \backslash \{e\}$)
(14) **if** $G(g(\bar{x})) = k$ **and** $g(\bar{x}) < g_k^{min} + \alpha\ (g_k^{max} - g_k^{min})$
(15) $RCL = RCL \cup \{e\}$
(16) $e =$RandomChoice(RCL)
(17) $x = x \backslash \{(e, r^*[e], s^*[e])\}$
(18) $A = A \backslash \{e\}$
(19) **return** x

borhood search [3], searching neighborhoods by increasing size.

Algorithm 4: Local search main cycle.

LocalSearch(x)
(1) $s =$ Improve(x)
(2) **while** $s \neq x$
(3) $x = s$
(4) $s =$ Improve(x)
(5) **return** s

Algorithm 5: Improvements.

Improve(x)
(1) $S = N_1(x)$
(2) **while** $S \neq \{\}$
(3) $s =$ RandomChoice(S)
(4) **if** s is better than x
(5) **return** s
(6) $S = S \backslash \{s\}$
(7) $S = N_2(x)$
(8) **while** $S \neq \{\}$
(9) *do the same as above*
(10) **return** x

4 Results and conclusion

Some preliminary tests of this algorithm with the benchmark problems of the *International Timetabling Competition (ITC)* [5] (more details on the benchmark test suite and on other meta-heuristics for this problem are presented in [1]) and some variants of it indicate that the algorithm provides a valid strategy for automated timetabling. In particular, feasible solutions (i.e., with no room clashes, no student clashes, and no features missing for any event) for the original benchmarks were found in less than 600 seconds of CPU time on a computer running the Linux operating system, with a pentium CPU at 1000 MHz.

In the ITC problem, the first goal (which defines "feasible" solutions) corresponds to having no room or student clashes, and have all rooms are compatible with the events placed there. Quality of the solutions is measured on a second goal, and comprises three (soft) constraints: on student schedules there should be no three or more events on a row, there should be no single-event days, and there should be no events on the last slot of the day.

All the instances provided in the ITC had an optimal solution with no violations on any of the constraints, and this information was available at the time of the competition. The method proposed in this paper does not make use of this; it is usable even on instances which do not have any satisfiable constraint. Hence, it is not fully comparable to other methods presented in the ITC. Anyway, in this contest were presented methods much faster than the one we propose here. Table 1 presents a comparison of the results of the approach presented in this paper to those of the winner of the ITC [4].

The algorithms presented in this paper are not problem-specific, and are hence employable unchanged in a wide range of problems. As timetabling problems are very unsteady, changing from institution to institution, and often changing with the time too, it is expected that the approach introduced is of practical relevance.

References

[1] K. Socha M. Sampels M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium, 2003.

[2] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[3] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[4] Philipp A. Kostuch. Timetabling Competition – SA-based heuristic. Internet repository, 2003. http://www.stats.ox.ac.uk/ kostuch/TTcomp.pdf.

[5] Ben Paechter. International timetabling competition. Internet repository, 2003. http://www.idsia.ch/Files/ttcomp2002.

[6] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedure. In Fred Glover and G. Kochenberger, editors, *State of the Art Handbook in Metaheuristics*. Kluwer Academic Publishers, 2001.

Problem number	Violation T	D	L	Total	Winner Results
1	91	6	109	206	45
2	80	4	103	187	25
3	88	8	111	207	65
4	198	19	275	492	115
5	225	8	260	493	102
6	189	12	126	327	13
7	183	12	251	446	44
8	132	4	152	288	29
9	113	8	113	234	17
10	80	3	174	257	61
11	121	7	143	271	44
12	114	8	228	350	107
13	165	5	199	369	78
14	213	14	306	533	52
15	169	13	199	381	24
16	64	5	102	171	22
17	210	22	186	418	86
18	77	6	59	142	31
19	136	11	371	518	44
20	114	8	91	213	7
Sum				6503	1011

Table 1: Results for the International Timetabling Competition benchmarks, and comparison to the algorithm that won the competition. All the solutions are feasible (i.e., there are no room or student clashes, and all the rooms are compatible with the events placed there). Quality of the solutions: T is the number of three or more events on a row, D the number of single-event days, and L the number events on the last slot of the day, on student schedules.