UNIVERSITY OF UDINE

MASTER IN COMPUTER SCIENCE

ACADEMIC YEAR 2018-2019

Report of the Advanced Scheduling Systems project: Referee Assignment problem

Student:
Michele Collevati

Matriculation number:
116286

E-mail:
collevati.michele@spes.uniud.it

Problem: Referee Assignment

The *referee assignment* is one of the classical problems in sport scheduling. We consider a variant of this problem arising from an amateur basketball league with many divisions. The problem consists in assigning a *variable* number of referees to each game of every division for the full season, according to a set of constraints and objectives.
In our formulation, a referee receives a lump sum for the performance plus a mileage allowance for the distance travelled. For this reason, the minimization of the total amount of kilometres travelled is one of the main aspects of our problem[1].

Additional feature

A referee unavailability for a period of days can be easily specified in the input file using the following syntax: *dd/mm/yyyy hh:mm∼dd/mm/yyyy hh:mm* (e.g. *4/1/2019 18:00∼8/1/2019 21:30*) .

Makefile

The project was developed using `GCC 8.3.0` compiler.
Four techniques have been developed, namely: Enumeration, Backtracking, Greedy and Local Search. Each one has its own folder (under `/Techniques/`) and Makefile.
Below the commands to use the Makefile:

- *make*: to compile the sources and create the executable file
- *make clean*: to delete files generated by the compilation of sources

Enumeration solution

It enumerates only the solutions that satisfy the minimum and maximum number of referees hard constraints, in order to reduce the search space.

Backtracking solution

As for Enumeration, only the solutions that satisfy the minimum and maximum number of referees are taken into consideration.

Greedy solution

Steps of the algorithm:

(1) Randomly choose the first game g to be assigned

(2) <u>If</u> the number of referees assigned to g is less than the maximum number of assignable referees, <u>then</u> goto (2.1) <u>else</u> goto (3):

[1] For the full problem specification please see *Referee_Assignment_problem.pdf*

(2.1) If the number of referees assigned to g is less than the minimum number of referees to be assigned, then goto (2.1.1) else goto (2.1.2):

(2.1.1) Assign to g the referee that costs less for the current partial solution

(2.1.2) Assign to g the referee that minimizes the current partial solution, if such referee exists (breaking ties randomly)

(3) Randomly take another game *not* yet considered and goto (2), until *not* all the games have been processed

(4) Repeat the previous steps until at least one referee is assigned to a game

Required invariant: at each referee assignment hard constraints are *not* violated, except for the minimum number of referees, of course.

Since at each iteration all the games are scanned and at least one referee is assigned, the worst case time complexity of the algorithm is $\mathcal{O}(maxR_{g'} \times numG^2)$, where $maxR_g$ is the maximum number of referees assignable to g, $maxR_{g'} := \max_g(maxR_g)$ and $numG$ is the number of games.

Local Search solution

Soft constraints weights:
A soft constraint weight can be specified via a command-line parameter by typing `--main::` followed by its label:

- `loe (int)` for the LackOfExperience weight
- `gd (int)` for the GamesDistribution weight
- `td (int)` for the TotalDistance weight
- `o (int)` for the OptionalReferee weight
- `af (int)` for the AssignmentFrequency weight
- `ri (int)` for the RefereeIncompatibility weight
- `ti (int)` for the TeamIncompatibility weight

If *not* specified, a weight is equal 1 by default.

Greedy state:
A new state is generated using the greedy algorithm explained above, which has been integrated into the framework.

Random state:
A random state is generated trying to guarantee that the minimum number of referees is assigned to each game. As for the other hard constraints, they must all be satisfied. This (possibly) allows us to start the search from a feasible

solution, avoiding the problem specified in the next section.

Feasibility of a move:
A move is feasible only if *no* hard constraint is violated. This is because otherwise we can incur in states for which then is difficult to satisfy certain types of hard constraints, such as the minimum level required for a game. For example, if a referee does *not* satisfy this constraint for multiple games, unassign the referee from a single higher level game does *not* improve the cost of that constraint, which remains violated. So, the next improving move depends only on the cost of other constraints.

ChangeAssignedReferees Move:

Syntax: g: $\{Ri, ..., Rj\} \rightarrow \{Rk, ..., Rl\}$
where $1 \leq g \leq number\_of\_games$ is a game and $\forall\, 1 \leq i \leq number\_of\_referees$ Ri is a referee. The set on the left of the arrow, i.e. $\{Ri, ..., Rj\}$, must be equal to that of the currently assigned referees to g. Whereas the one on the right, i.e. $\{Rk, ..., Rl\}$, is the set of new assigned referees to g.
For each game g, the number of different possible moves is given by:

$$\sum_{i=minR_g}^{maxR_g} \binom{numR}{i} \tag{1}$$

where $minR_g$ is the minimum number of referees to be assigned to g, $maxR_g$ is the maximum number of referees assignable to g and $numR$ is the number of referees. Since for a fixed value of i, $\binom{numR}{i}$ is $\mathcal{O}(numR^i)$, the function 1 is $\mathcal{O}(numR^{minR_g} + \ldots + numR^{maxR_g})$ which is $\mathcal{O}(numR^{maxR_g})$, because $maxR_g \geq minR_g$. Hence, the size of the neighborhood is $\mathcal{O}(numR^{maxR_{g'}} \times numG)$, where $maxR_{g'} \coloneqq \max\limits_{g}(maxR_g)$ and $numG$ is the number of games.

AddRemoveReferee Move:

Syntax: g: $Ri \rightarrow Rj$
where $1 \leq g \leq number\_of\_games$ is a game and $\forall\, 1 \leq i \leq number\_of\_referees$ Ri is a referee. The referee on the left of the arrow, i.e. Ri, is the removed one and therefore must belong to the set of the currently assigned referees to g. Whereas the referee on the right, i.e. Rj, is the assigned one and therefore must *not* belong to it. 0 in place of Ri means remove no referee, while in place of Rj means assign no referee.
For each game g, the number of different possible moves is given by (in case removal and assignment are both possible w.r.t. the minimum and maximum referees constraints):

$$numA + \overline{numA} + numA \times \overline{numA} \tag{2}$$

where $numA$ is the number of currently assigned referees to g and $\overline{numA}$ is the number of the remaining ones (*not* assigned). Since $numA$ can be equal to

$numR/2$, the function 2 is $\mathcal{O}(numR/2 + numR/2 + numR/2 \times numR/2)$ which is $\mathcal{O}(numR^2)$. Hence, the size of the neighborhood is $\mathcal{O}(numR^2 \times numG)$. However, thanks to the maximum referees constraint, $numA$ is usually very small w.r.t. $\overline{numA}$ and so we are far away from the worst case. It follows that, the neighborhood resulting from this move is smaller and faster to be explored than the previous one.

Execution results

The benchmarks were made on the Asus N550JK laptop with an Intel Core i7-4700HQ CPU @ 2.40GHz × 8, 16GB of DDR3L 1600MHz SDRAM and Antergos Linux 64-bit operating system.

Time limit (t.l.): *3 min.*
Soft constraints weights: $loe = gd = td = o = af = ri = ti = 1$
Local Search metaheuristic used: *Simulated Annealing (SA)*, with *ChangeAssignedReferees (CAR)* and *AddRemoveReferee (ARR)* moves and the following parameters:

- start temperature: *1000.0*
- min temperature: *1.0*
- cooling rate: *0.999*
- neighbors sampled: *500*
- neighbors accepted: *50*

	Enumeration		Backtracking		Greedy		Local Search (SA-CAR)		Local Search (SA-ARR)	
	best value found	time	best value found	time	best value found	time	best value found	time	best value found	time
RA-1-8	unknown	t.l.	6539	t.l.	4415	0.008s	4343	97.5929s	4343	51.8646s
RA-1-10	unknown	t.l.	6361	t.l.	2740	0.015s	2706	45.5183s	2717	35.4899s
RA-2-16	unknown	t.l.	12490	t.l.	3215	0.066s	3180	59.0577s	3185	46.3944s
RA-2-20	unknown	t.l.	15720	t.l.	5400	0.113s	5400	84.5807s	5369	50.7522s
RA-3-24	unknown	t.l.	13260	t.l.	4484	0.151s	4447	58.752s	4430	51.1456s
RA-3-30	unknown	t.l.	26483	t.l.	8562	0.443s	8530	121.325s	8493	72.7224s
RA-4-32	unknown	t.l.	27537	t.l.	8775	0.489s	8828	113.967s	8695	87.0091s
RA-4-40	unknown	t.l.	19385	t.l.	5892	0.775s	5967	83.2675s	5905	72.4499s
RA-5-40	unknown	t.l.	35725	t.l.	10931	1.485s	10927	143.552s	10739	118.375s
RA-5-50	unknown	t.l.	42626	t.l.	11768	3.726s	12573	165.692s	12088	146.629s