An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition, 2010

Edmund K. Burke[1], **Tim Curtois**[1]

[1]School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham. NG8 1BB. UK

We present a modelling approach and the results of two different algorithms applied to the instances of the first international nurse rostering competition. The first algorithm, variable depth search, is an ejection chain based method. It was applied to the *sprint* instances for which a maximum computation time of 10 seconds was allowed. The second algorithm, a branch and price method, was applied to the *medium* and *long* instances for which maximum computation times of 10 minutes and 10 hours respectively were allowed. As will be shown, however, the branch and price algorithm was generally able to solve these instances to proven optimality well within the time allowed.

The Model

The first step for both the algorithms is to convert the competition instance to our staff rostering model. The model was designed with the aim of being able to apply it to a wide variety of rostering problems. To achieve that goal, the model allows many different types of constraint and objectives (soft constraints) to be defined. It includes:

- Three general constraints which can be used to model the requirements on individual employee work schedules:
  - *PatternMatch* – A constraint based on regular expressions and pattern matching.
  - *Workload* – A constraint for modelling requirements such as min/max time units worked between two dates.
  - *Conditionals* – For modelling requirements of the form IF… THEN…
- Multiple levels of shift/time period coverage requirements which can also be defined by skill level and/or groups of employees.
- Pairing constraints to ensure two (or more) employees work at the same time or oppositely do not work at the same time.
- And more...

More information on the model and its xml based data format can be found at http://www.cs.nott.ac.uk/~tec/NRP/ . At this website, there is also a graphical user interface (Roster Booster) available for download which can be used to create and load models and view and modify instances and solutions. The algorithms presented here are also included in Roster Booster. The website also provides a wide variety of rostering benchmark instances collected from a number of different sources.

For the competition instances, all individual employee schedule constraints were modelled using the *PatternMatch* constraint except one which was modelled using the *Conditional* constraint.

Replicating the competition's model and objective function was challenging due to its complexity and the presence of some unusual constraints. The results shown here are based on our implementation of the organiser's model. Although we verified every solution generated using the checker provided by the organisers and we believe our objective functions to be identical, it cannot be guaranteed. For example, we (or the organisers) could easily have incorrectly implemented something defined in the problem description which would mean we are solving slightly different instances.

Variable Depth Search Algorithm

The variable depth search is an ejection chain based algorithm. It was applied to the *sprint* instances which have a 10 second maximum computation time. The basic algorithm is described in [2]. Since that technical report was published however, its performance has been improved though faster implementation and additional heuristics. It now also includes fast, hill climbing methods at the start of the algorithm. These hill climbers use a number of different search neighbourhoods. In its constructive phase at the start it also uses a dynamic programming algorithm to build individual schedules for single employees. The same algorithm is used by the branch and price method to solve the pricing problem (i.e. to generate new columns).

Branch and Price Algorithm

Branch and price is a branch and bound method where each node of the branch and bound tree is a linear programming relaxation which is solved using column generation. The column generation consists of a restricted master problem and a pricing problem. Solving the pricing problem provides new negative reduced cost columns to add to the master problem. We modelled the pricing problem as a resource constrained shorted path problem and solve it using a dynamic programming approach. If the full complexity of the *PatternMatch* and *Conditional* constraints is examined, it can be imagined how non-trivial the dynamic programming algorithm is and how challenging it was to ensure it was fast yet robust.
The pricing problem can be basically regarded as the problem of finding the optimal work schedule for an individual employee but with the addition of dual costs, that is, additional (possibly negative) costs based on which assignments are made or not made. In non-root nodes of the branch and bound tree, there may also be additional constraints on certain assignments that must or must not be made. Previous applications of branch and price to staff rostering problems include [3, 5, 7, 8]. For a recent overview of column generation, see [6] and for further reading on resource constrained shortest path problems see [4].
To solve the master problem we used the simplex method of the open-source, Coin-OR linear programming solver (clp) [1].

Profiling the algorithm reveals that during the column generation, typically about 5% of the computation time is spent re-solving the restricted master problem (the simplex method) whereas the other 95% of the time is used in solving the sub-problems (generating the new columns using the dynamic programming algorithm). This meant that the performance of the algorithm could be most significantly improved through the following two approaches.

1) Reducing the number of calls to the pricing problem solver.
2) Improving the performance of the pricing problem solver.

Stabilisation (reducing the oscillation of the dual values) was particularly important and effective for the first. For the second, additional heuristics were very effective, especially exploiting the fact that it is not necessary to find the most negative reduced cost column each time (i.e. the pricing problem does not have to be solved to optimality until it is necessary to show that there are no more negative reduced cost columns).

Within the allowed time limit, two different heuristic branching strategies are applied in the branch and bound tree to try and find new solutions (upper bounds). The initial solution is provided by applying the variable depth search algorithm for 5 seconds.

Results

Table 1 contains the best solutions found for the *sprint* instances using the variable depth search with a maximum execution time of 10 seconds. Table 2 shows the results of applying the branch and price algorithm to all instances. Although the competition allowed 10 hours for the *long* instances, we restricted it to 10 minutes as optimal or very near optimal solutions to these instances were usually found within 10 minutes anyway. The lower bound column shows the objective function value after solving the root node. The upper bound column shows the best solution found within the maximum time allowed. Solutions in **bold** are optimal. All experiments were performed on a desktop PC with an Intel Core 2 Duo 2.83GHz processor.

Instance	Solution	Time (secs)
sprint01	**56**	10.0
sprint02	**58**	10.0
sprint03	**51**	10.0
sprint04	**59**	10.0
sprint05	58	10.0
sprint06	**54**	10.0
sprint07	**56**	10.0
sprint08	**56**	10.0
sprint09	**55**	10.0
sprint10	**52**	10.0
sprint_late01	**37**	10.0
sprint_late02	**42**	10.0
sprint_late03	**48**	10.0
sprint_late04	75	10.0
sprint_late05	**44**	10.0
sprint_late06	**42**	10.0
sprint_late07	**42**	10.0
sprint_late08	**17**	10.0
sprint_late09	**17**	10.0
sprint_late10	**43**	10.0

Table 1. Results of the variable depth search applied to the *sprint* instances.

Instance	LB (root node)	Best UB	Time (secs)
sprint01	56	**56**	32.3
sprint02	58	**58**	16.8
sprint03	51	**51**	61.6
sprint04	58.5	**59**	29.6
sprint05	57	58	270.4

sprint06	54	**54**	27.4
sprint07	56	**56**	29.6
sprint08	56	**56**	14.0
sprint09	55	**55**	20.2
sprint10	52	**52**	22.9
sprint_late01	37	**37**	25.0
sprint_late02	41.4	**42**	16.1
sprint_late03	47.83333333	**48**	24.0
sprint_late04	72.5	**73**	131.1
sprint_late05	43.66666667	**44**	29.2
sprint_late06	41.5	**42**	151.4
sprint_late07	42	**42**	17.9
sprint_late08	17	**17**	10.3
sprint_late09	17	**17**	22.8
sprint_late10	42.85714286	**43**	27.9
medium01	240	**240**	34.2
medium02	239.25	**240**	41.1
medium03	235.5	**236**	49.6
medium04	236.21875	**237**	171.3
medium05	302.1	**303**	150.7
medium_late01	156	157	600.0
medium_late02	18	**18**	17.1
medium_late03	28.25	**29**	94.0
medium_late04	34.33333333	**35**	152.2
medium_late05	106.6666667	**107**	189.0
long01	197	**197**	84.9
long02	218.5	**219**	108.2
long03	240	**240**	69.9
long04	303	**303**	120.3
long05	284	**284**	84.4
long_late01	235	**235**	162.8
long_late02	229	**229**	590.9
long_late03	218.5	220	600.2
long_late04	220.6666667	**221**	188.0
long_late05	82.5	**83**	373.7

Table2. Results of the branch and price applied to all instances.

The Competition

Table 3. shows how the instances we submitted ranked against the instances submitted by the other competitors. As can be seen, for every instance, our solution was either first or first equal. To produce the final ranking of the competitors the competition organisers created some hidden instances and ran the submitted solvers on these hidden instances. Unfortunately for us however, the organisers made a change in some of the hidden instances which we were not expecting. In all the non-hidden instances (including the 'hint' instances) the start date of the planning period was the 1st January 2010 (which is a **Friday**). In 17 out of the 20 hidden instances the organisers kept the planning horizon as four weeks but changed the start date of the planning period to 1st June 2010 (which is a **Tuesday**). As changing the start date does not effect the complexity of the problem but just makes the modelling process slightly more complicated we were not expecting the start date to change in the hidden instances. However, the change meant that in our objective function all the indices for weekend related constraints were out by exactly 4. As such our solver's

objective function values for nearly all the hidden instances was incorrect which had a significant adverse effect on our final rankings in the competition.

Instance	Solution	Ranking
sprint01	**56**	1st =
sprint02	**58**	1st =
sprint03	**51**	1st =
sprint04	**59**	1st =
sprint05	58	1st =
sprint06	**54**	1st =
sprint07	**56**	1st =
sprint08	**56**	1st =
sprint09	**55**	1st =
sprint10	**52**	1st =
sprint_late01	**37**	1st =
sprint_late02	**42**	1st =
sprint_late03	**48**	1st =
sprint_late04	75	1st
sprint_late05	**44**	1st =
sprint_late06	**42**	1st =
sprint_late07	**42**	1st
sprint_late08	**17**	1st =
sprint_late09	**17**	1st =
sprint_late10	**43**	1st
medium01	**240**	1st =
medium02	**240**	1st =
medium03	**236**	1st =
medium04	**237**	1st =
medium05	**303**	1st =
medium_late01	157	1st
medium_late02	**18**	1st
medium_late03	**29**	1st
medium_late04	**35**	1st
medium_late05	**107**	1st
long01	**197**	1st =
long02	**219**	1st =
long03	**240**	1st =
long04	**303**	1st =
long05	**284**	1st =
long_late01	**235**	1st
long_late02	**229**	1st
long_late03	220	1st
long_late04	**221**	1st
long_late05	**83**	1st =

Table 3. Competition Ranking

References

1. COIN-OR Linear programming solver (https://projects.coin-or.org/Clp). 2010.
2. Burke, E.K., T. Curtois, R. Qu, and G. Vanden Berghe, A Time Predefined Variable Depth Search for Nurse Rostering. 2007, School of Computer Science and IT, University of Nottingham. Technical Report. Available from: http://www.cs.nott.ac.uk/TR/2007/2007-6.pdf

3. Eveborn, P. and M. Rönnqvist, *Scheduler – A System for Staff Planning.* Annals of Operations Research, 2004. **128**: pp. 21–45.
4. Irnich, S. and G. Desaulniers, *Shortest Path Problems with Resource Constraints*, in *Column Generation*, G. Desaulniers, J. Desrosiers, and M.M. Solomon, Editors. 2005, Springer. pp. 33-65.
5. Jaumard, B., F. Semet, and T. Vovor, *A Generalized Linear Programming Model for Nurse Scheduling.* European Journal of Operational Research 1998. **107**(1): pp. 1-18.
6. Lubbecke, M.E. and J. Desrosiers, *Selected Topics in Column Generation.* Operations Research, 2005. **53**(6): pp. 1007-1023.
7. Maenhout, B. and M. Vanhoucke, *Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem.* Journal of Scheduling, 2010. **13**(1): pp. 77-93.
8. Mason, A.J. and M.C. Smith. *A Nested Column Generator for solving Rostering Problems with Integer Programming*. in International Conference on Optimisation: Techniques and Applications. 1998. Perth, Australia. L. Caccetta, et al. (ed). pp. 827-834.