MILP formulations of cumulative constraints for railway scheduling — A comparative study

Martin Aronsson[1], Markus Bohlin[1] and Per Kreuger[1]

Swedish Institute of Computer Science,
Box 1263, SE-164 29 Kista, Sweden
Martin.Aronsson@sics.se, Markus.Bohlin@sics.se, Per.Kreuger@sics.se

Abstract This paper introduces two Mixed Integer Linear Programming (MILP) models for railway traffic planning using a cumulative scheduling constraint and associated pre-processing filters. We compare standard solver performance for these models on three sets of problems from the railway domain and for two of them, where tasks have unitary resource consumption, we also compare them with two more conventional models. In the experiments, the solver performance of one of the cumulative models is clearly the best and is also shown to scale very well for a large scale practical railway scheduling problem.

Keywords. Railway transport scheduling, Cumulative scheduling, Mixed Integer Linear Programming (MILP) modelling and pre-processing

1 Introduction

Railway scheduling is a rich source of challenging optimisation and combinatorial decision problems. Along with vehicle routing problems with some unique properties [1,2,3], track resource scheduling [4,5] is at the core of timetable construction for modern rail traffic planning. The methods described in this paper may be used to verify feasibility of proposed timetables, search (or optimise) for timetables with certain properties, or reduce conflicts between disparate requirements originating from e.g. customers, business areas or transport political priorities within the infrastructure manager. The presentation of the methods is rather technical but most of the problems used in the empirical sections are derived from real fixed timetables and early stage timetable proposals. The results clearly indicate one of the described methods as superior for this important practical railway scheduling problem.

Constraint programming (CP) techniques have been quite successful in solving both academic [6,7,8,9,10] and real-world scheduling problems [11,12,13,14,15]. One of the main benefits of CP for such problems is the presence, in most modern solvers, of very efficient filtering mechanisms in the form of constraint abstractions for both classical job shop and generalisations such as the cumulative resource scheduling problem. Using demand-driven filtering during search for integer solutions constitutes a powerful decision mechanism that have also been used successfully for optimisation [16,8]. However, to optimise classical job shop problems and their cumulative generalisations efficiently it is generally also necessary to employ quite sophisticated search heuristics.

Mixed Integer Linear Programming (MILP) is another technique for combinatorial problem solving which have been applied to a wide variety of industrial-level problems. For scheduling problems with unitary resources, standard linear boolean formulations also scale very well, especially for problems with a lot of linear side conditions that can be exploited by modern MILP solvers.

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2141

For cumulative scheduling problems, however, there do not seem to exist any standard MILP formulations. For certain classes of problems, e.g. where all tasks have unitary resource consumption, formulations based on geometric placement can be used [5]. These can, as we will see, be quite efficient for the problems they *can* encode.

Cumulative constraints [17] are well known in the CP community where efficient algorithms based on sweep [18] and/or task-intervals [19] are used to prune the search space, both as a pre-processing mechanism and on demand for variable domain reduction during search. Several variants of the constraint have been described e.g. in [20].

These constraints normally restrict the cumulative capacity utilisation of tasks executing simultaneously not to exceed an upper bound. Capacities and capacity utilisation are normally fixed integers while the start times and durations are decision variables. Variants where the resource consumption of each task is also variable and possibly constrained by the start time and duration occur as well. In this paper we focus on the case where the capacity and the resource consumption are constant integers. We have not found this to be restrictive in practice for practical problems in the railway domain.

Geometric placement constraints are related to cumulative constraints. The most common form is probably that of filtering for non-overlap of rectangles in the plane [21] which, in the context of scheduling, corresponds to allocation of unit capacity resources to tasks with unit resource consumption combined with a multi-resource scheduling problem. The resource allocation is represented as the placement of a a unit height rectangle in the y-dimension and the start time as the placement of its left edge and the duration as its length in the x-dimension.

In classical cumulative scheduling, there is no concept corresponding to the placement of the lower edge on the y-axis, and the resource consumption is arbitrary. Still, the special case of unit resource consumption is of considerable practical interest, and for these, the placement formulation can be used by considering the number of resources as a cumulative capacity and just ignoring the values of the y-placement variables. Any solution to the placement problem is clearly feasible for the cumulative as well.

We will describe four different models, two for the placement formulation and two for the cumulative constraint, define filtering methods for each, note some of their complexity properties and investigate solving performance for them on three separate sets of problems. The first two sets of problems are derived from a practical case in rail traffic scheduling where all the tasks have unit resource consumption. In the third, a set of random problems with a more general structure and of varying sizes and difficulties are studied.

In addition, in a fourth, empirical section, we briefly describe the results of using a selection of the described methods in an industrial scale rail transport scheduling problem. This problem was what originally motivated our research, and even though the problem has a quite special structure it is of great practical importance. We conclude with a summary of our findings.

2 Preliminaries and notation

2.1 Notation for model parameters and variables

Let n denote the number of tasks (individual trains using a track or station resource) in the problem and use $0 < i, j \leq n$ as task indices. Let, furthermore, c denote the resource (station) capacity limit and h_i the resource consumption for task i. Let s_i denote the start time variable for task i, bounded by an interval $\underline{s_i} \leq s_i \leq \overline{s_i}$ and d_i the duration variable for task i, bounded by an interval $\underline{d_i} \leq d_i \leq \overline{d_i}$.

2.2 Maximal clique construction

In cumulative scheduling it is often useful to do an analysis of the parameters and bounds of the problem. One of the most obvious ways to do this is to construct subsets of tasks that *can* overlap in time. In CP, this type of computation is performed iteratively during search to filter the domains or bounds of the decision variables, but it can also be used for pre-processing in MILP formulations to filter equations and booleans that need not be maintained by the solver.

Formally, this is achieved by considering the tasks of the problem as nodes in a graph and letting two tasks i and j be connected by a link if and only if they can overlap in time. Then, all *maximal* cliques (completely connected sub-graphs) of this graph will have the property that, unless a task is already in the clique, it cannot overlap *all* the others.

This is a very useful property in cumulative scheduling since when we wish to limit the number of simultaneously overlapping task, it is sufficient to consider each maximal clique separately and the complexity of enforcing cumulative conditions on the set of all tasks is often bounded by some function of the sizes of the maximal cliques, rather than the size of the task set itself. In practical problems this is often of great value, since the majority of tasks cannot be arbitrarily placed in time. This makes the maximal cliques small compared to the total number of tasks.

To construct the set of all maximal cliques used in the models below, we use a straightforward sweep algorithm which has linear time complexity in the size of the set of tasks. In the model description below we will often generate a set of equations for each maximal clique Clq_k and where $1 \leq n_k \leq n$ is the size of the k'th clique.

3 Model descriptions

The first two models described below are restricted to handle tasks with unitary resource requirements. The reason for this is that these are based on a rectangle placement approach which does not capture the general cumulative constraint which may be satisfied even though no rectangle placement exists. They are, in fact, more close to models for placing non-overlapping rectangles of unit height onto the plane. In practice however, these are quite useful models since in many situations where the cumulative constraint is used, there is an underlying problem structure of this type. E.g. in train scheduling, a station may be modelled as a cumulative resource that allows a maximum number of trains to occupy the station at any one time. The type of model proposed here allows us to also exclude the use of certain tracks for a particular train, depending on track lengths or other capacity restrictions, which is not straightforward in a pure cumulative model.

The next two models capture the semantics of a general cumulative constraint with a fixed upper bound on resource consumption and arbitrary but fixed resource consumption for all tasks.

3.1 Explicit unitary resource allocation (integer formulation)

This model treats each cumulative resource as a collection of unitary sub-resources and explicitly allocate these to tasks with unit resource consumption. This is achieved through the use of an integer decision variable y_i for each task i to denote the individual sub-resource allocated to the task. If two tasks i and j use the same sub-resource, they must be non-overlapping in time. The model uses two boolean variables p_{ij} and w_{ij} for each pair of transports i and j. $p_{ij} = 1$ is used to encode that the task i completely precedes task j and $w_{ij} = 1$ that they *do* overlap in time, and thus must use different sub-resources.

First, let us express a non-overlap constraint: Either the end time of task i is less than or equal to the start time of task j: $s_i + d_i - s_j \leq 0$ or the same is true for task j in relation to task i: $s_i - s_j - d_j \geq 0$. We reflect this disjunction in the boolean p_{ij}:

$$s_i + d_i - s_j - M\,(1 - p_{ij}) \le 0$$
$$s_i - s_j - d_j + M\,p_{ij} \ge 0$$

where M is any constant large enough to dominate the equation in which it occurs. This is, of course, a standard formulation that occurs everywhere in the literature (see e.g. [22,23]) but how do we proceed if we want to *count* and limit the number of overlapping tasks?

In the case where we *do* want to allow an overlap we need an additional boolean that cancels the effect of the above equations. We want to do this in a way so that whenever this variable takes the value 0, our equations will be equivalent to the ones above, and cancel them completely otherwise:

$$s_i + d_i - s_j - M\ (1 - p_{ij}) - M\,w_{ij} \le 0$$
$$s_i - s_j - d_j + M\ p_{ij} + M\,w_{ij} \ge 0$$

When the two tasks *do* overlap, and the variable w_{ij} thus takes the value 1, we need to ensure that the two tasks are allocated different sub-resources. We can do this by ensuring that the difference between y_i and y_j is nonzero:

$$y_i - y_j + M\,u_{ij} + M\,(1 - w_{ij}) > 0$$
$$y_j - y_i + M\,(1 - u_{ij}) + M\,(1 - \ w_{ij}) > 0$$

where y_i, y_j are integers and the booleans u_{ij} encodes if $y_i < y_j$ or the other way around, in the case where w_{ij} is 0.

As noted above, it is sufficient to enforce these conditions for each pair of tasks in the maximal cliques, so that for each clique Clq_k with n_k tasks, the number of integer variables will be n_k, the number of booleans $3\frac{n_k(n_k-1)}{2}$ and the number of equations will be $2n_k(n_k - 1)$. Note that, by sharing variables between the cliques, the total numbers are significantly less than the sum over all cliques and is, for the integer variables, bounded by n and for the booleans, by $3\frac{n(n-1)}{2}$.

In summary, the temporal non-overlap condition for tasks allocated the same sub-resource can (since y-variables are integers) thus be stated in linear form as:

$$s_i - s_j + d_i + M\,p_{ij} - M\,w_{ij} \le M$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,w_{ij} \ge 0$$
$$y_i - y_j + M\,u_{ij} - M\,w_{ij} \ge 1 - M$$
$$y_j - y_i - M\,u_{ij} - M\,w_{ij} \ge 1 - 2\,M$$

for all pairs $i < j \in Clq_k$ of tasks and each maximal clique Clq_k, where p_{ij}, w_{ij}, u_{ij} are booleans and $1 \le y_i, y_j \le c$ are integers. Note that we need to enforce the equations in the solver only when the size of the clique is strictly larger then the resource capacity.

3.2 Explicit unitary resource allocation (boolean formulation)

This model is very similar to the one above but uses, instead of each integer variable y_i, c number of booleans m_{ik}, each being one, denoting that the task i is allocated sub-resource k. We want to enforce the overlap condition between two tasks i and j if and only if $m_{ik} = m_{jk} = 1$ for some k i.e. if $(1 - m_{ik}) = (1 - m_{jk}) = 0$. The equations stating the non-overlap can then be formulated:

$$s_i + d_i - s_j - M\,(1 - p_{ij}) - M\,(1 - m_{ik}) - M\,(1 - m_{jk}) \le 0$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,(1 - m_{ik}) + M\,(1 - m_{jk}) \ge 0$$

which in linear form becomes

$$s_i + d_i - s_j + M\,p_{ij} + M\,m_{ik} + M\,m_{jk} \leq 3\,M$$
$$s_i - s_j - d_j + M\,p_{ij} - M\,m_{ik} - M\,m_{jk} \geq 2\,M$$

for all pairs of tasks $i < j \in Clq_k$, for each maximal clique Clq_k, and each $0 < k < c$ and where, in addition, the resource condition is stated:

$$\sum_{0<k\leq c} m_{ik} = 1$$

for all tasks i, i.e. essentially a set partitioning formulation.

Note that the number of booleans and overlap equations now increase by a factor of $2c$ to become $cn_k(n_k-1)$ where n_k is the size of the clique and c the resource capacity. The number of resource conditions, on the other hand, now depends *linearly* on the product of cn_k. We would expect this model to be reasonably efficient when c is small in comparison to the clique size n_k. If, on the other hand these parameters are of comparable size, the number of booleans is effectively cubic. The advantage of this type of model is that the modern MILP-solvers tend to treat pure boolean formulations more efficiently than general MILP formulations.

A similar model for a traffic (re)scheduling problem was presented in [5] as part of a larger model capturing several more aspects of a train (re)scheduling problem but this type of model is probably more or less a standard formulation.

3.3 Min conflicting sub-clique model

This model captures the classical cumulative constraint more exactly than the ones proposed above in the sense that tasks may have arbitrary resource consumption and that there is no notion of sub-resources.

The idea behind this model is that for each maximal clique with tasks of sufficient cumulative resource consumption, there exists a (possibly large) number of *minimal* sub-cliques such that the sum of the resource consumptions of the involved tasks exceeds the resource capacity c. They need to be minimal in the sense that removing any single element would make the sum of resource consumptions of the remaining tasks less than or equal to the resource capacity. This means that we can limit the number of actual overlaps in the sub-clique to be *strictly* less than the number of pairs in the (minimal) clique itself.

Since each larger sub-clique that can contribute to a violation of the constraint can do so only by violating a minimal sub-clique of itself, it is sufficient to state the resource conditions for the minimal sub-cliques. We will use the same formulation for the non-overlap condition as before, i.e.

$$s_i + d_i - s_j + M\,p_{ij} - M\,w_{ij} \leq M$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,w_{ij} \geq 0$$

for all $i < j \in Clq_k$ and each maximal clique Clq_k. We may now count and limit the number of overlaps in each *minimal* sub-clique as follows

$$\forall Mn \subseteq Clq_k \left(\left(\sum_{i\in Mn} h_i > c \right) \wedge \left(\forall Sb \subset Mn \sum_{i\in Sb} h_i \leq c \right) \rightarrow \sum_{i\leq j\in Mn} w_{ij} < \binom{|Mn|}{2} \right)$$

for each maximal clique Clq_k in the problem where the first conjunct in the precondition of the implication requires that the sub-clique can in fact contribute to a resource conflict, the second states the minimality condition and the conclusion limits the number of overlap variables that can take the value one to be strictly less than the number of pairs in the minimal sub-clique. Note that the tests for each potential sub-clique can be done when generating the equations and only the linear sum expression $\sum_{i\leq j\in Mn} w_{ij} \leq \binom{|Mn|}{2} - 1$ needs to be enforced by the solver.

In this model, for each clique Clq_k with n_k tasks, both the number of booleans and number of *overlap* equations will be $n_k(n_k-1)$. The number of minimal sub-cliques and corresponding *clique equations*, for a given max clique, however, depends both on the clique size $|Clq_k|$, the resource capacity c and the distribution of resource consumption for the involved tasks, and may in the worst case be exponential in the first two parameters. E.g. if the resource consumption of all tasks is one, the number of minimal sub-cliques will be the number of sub-cliques of a given size c, i.e. $\binom{|Clq_k|}{c+1}$. Even though modern IP-solvers are much more sensitive to the number of booleans than to the number of equations, this is clearly a disadvantage of this model.

Even worse, the number of sub-cliques to be tested for minimality is always exponential in the clique size. This means that the algorithm generating the equations should be very sensitive to increase in clique size. Still, for a typical randomly generated problem consisting of 300 tasks on a single resource, arbitrary resource consumptions up to a resource capacity of 5, max/average clique size of 26/18 and 139 separate cliques, all 9 752 equations are generated in about 170 seconds on a 1.6 GHz i686 laptop, so the filtering *does* scale to practical problem sizes and, for many large scale practical problems, the method performs, as we will see in section 4, very well.

3.4 Start point clique height sum model

This model is based on the observation that for each start point of a task, it suffices to measure and limit the resource consumptions of the other tasks that are possibly active at that point.

For each task i of a maximal clique with elements of sufficient size to generate a conflict, consider each other task j in the clique that has an earliest start point less than or equal to the *latest* start point of task i and a latest end point greater than the *earliest* start of i. Since only these can overlap task i we construct for each such task a boolean variable w_{ij} which will take the value 1 if and only if the start of task i falls within the duration of task j, i.e. if $s_j \leq s_i < s_j + d_j$. In order to do this, consider first the situation where this is *not* the case, i.e. where either $s_j > s_i$ or $s_i \geq s_j + d_j$. Encode this disjunction with a boolean p_{ij} such that:

$$s_i - s_j - M\,(1 - p_{ij}) < 0$$
$$s_i - s_j - d_j + M\,p_{ij} \geq 0$$

and use $w_{ij} = 1$ to encode the cancellation of *these* equations as follows:

$$s_i - s_j + M\,p_{ij} - M\,w_{ij} < M$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,w_{ij} \geq 0.$$

where p_{ij}, w_{ij} are booleans and the *strict* inequality in the first equation would in a pure MILP formulation be handled by the addition of a suitably small ϵ on the RHS.

Now, for each element i in each clique Clq_k constrain the scalar products: $\sum_{j\in Clq_k\setminus\{i\}} h_j w_{ij}$ to be less than or equal to the resource capacity c minus the resource consumption h_i of the task i:

$$\forall i \in Clq_k \sum_{j\in Clq_k\setminus\{i\}} h_j w_{ij} \leq c - h_i$$

for all maximal cliques Clq_k where w_{ij} are booleans. The number of clique equations is linear in the (maximal) clique size, but since the overlap equations are no longer symmetric, these must be stated for each *ordered* pair of tasks in the clique. This means that the number of booleans and overlap equations will both be $2n_k(n_k-1)$, which is twice as many as in the model of section 3.3.

4 Empirical findings

This section reports trial runs of the proposed methods on a number of different problems. Most of the problems are derived from an application in train scheduling, but since these only have tasks with unitary resource consumption, we have also evaluated the methods on a set of randomly generated problems where the resource consumption varies up to the resource capacity. Two sets of examples are single resource problems while the other two are more realistic examples consisting of trains using several resources in fixed sequences, job shop style.

4.1 Single resource unitary resource consumption examples

We have evaluated all four models on a set of problems derived from the domain of train timetable generation. More results on the full problem is presented in section 4.4 below. In this section, we consider a single resource at the time and present results for a number of representative station resources of varying size.

In table 1 the problem parameters and properties are summarised. We note that all problems

Table 1. Problem statistics for a selections of stations in the train problem

Station	KS1	FA	TÄL	LLN	MH	ÖB	LÅÖ	GDÖ	SK	HPBG
Capacity	1	2	2	2	3	3	3	4	5	10
Tasks	471	711	1000	1000	684	907	1000	717	804	1391
Cliques	246	319	520	591	194	356	489	120	63	43
Max/Avr clq size	7/3	7/4	8/5	9/5	7/4	8/5	9/5	7/5	8/6	14/11

are fairly large in terms of number of tasks but since the problems were generated by introducing a fixed amount of slack (±15 minutes) in a given feasible solution, the number of potential conflicts and hence clique sizes is relatively small. We would argue that this is a quite common situation in many large scale practical problems, and as shown in section 4.4, methods to solve such problems can certainly be put to very good use. Here we try to show that the methods we have described are in fact very good at exploiting this type of problem structure and scale surprisingly well considering that only default settings of the CPLEX solver were used to produce the solutions.

Table 2 gives the number of equations, booleans and integers for each of the four models and run-times for CPLEX 9.0 on a single core 2.6 GHz i686 Xenon processor. In addition, the time taken to generate the equation sets for each of the models is given in the last four rows. The short names of models used in the table are "MC" for the "Min conflicting sub-clique model" of section 3.3, "SC" for the "Start point height sum model" of section 3.4, "RB" for the boolean formulation of the "Explicit resource allocation model" of section 3.2 and "RI" for the integer version presented in section 3.1.

We note that the MC model is always best in terms of CPLEX execution time but that for some of the larger problems, the time to generate the equation set increases the total time to solve the problem significantly. Just adding the times together does not necessarily tell the whole

Table 2. Solution statistics for a selection of stations in the train scheduling problem

Param.	Method(s)	KS1	FA	TÄL	LLN	MH	ÖB	LÅÖ	GDÖ	SK	HPBG
Bools	MC	1 588	3 108	6 532	6 666	2 416	4 820	6 448	2 040	1 644	2 780
	SC	3 175	6 216	13 054	13 319	4 810	9 580	12 882	4 080	3 249	5 403
	RB	1 231	2 832	5 166	5 239	2 663	4 717	5 906	2 516	2 227	3 680
	RI	2 382	4 662	9 798	9 999	3 624	7 230	9 672	3 060	2 466	4 170
Integers	MC, SC, RB	0	0	0	0	0	0	0	0	0	0
	RI	437	639	950	953	485	769	894	374	281	229
Eqns.	MC	2 382	4 800	11 842	12 067	3 204	8 146	11 660	2 432	1 863	4 185
	SC	3 964	7 532	15 711	16 342	5 700	11 451	15 486	4 727	3 632	5 801
	RB	2 025	6 858	14 014	14 285	7 733	15 229	20 238	8 534	8 501	28 029
	RI	3 176	6 216	13 064	13 332	4 832	9 640	12 896	4 080	3 288	5 560
Solve (s)	MC	**0.03**	**0.29**	**1.23**	**1.54**	**0.09**	**0.34**	**0.64**	**0.06**	**0.06**	**0.25**
	SC	0.14	11.16	45.07	38.04	0.73	6.42	14.00	0.20	0.18	0.39
	RB	**0.03**	1.06	3.18	11.26	1.22	5.04	13.23	1.90	0.73	5.08
	RI	0.04	3.08	23.89	22.66	0.58	1.78	17.93	0.51	0.39	1.70
Gen. (s)	MC	0.17	0.69	2.42	2.52	0.63	2.47	3.87	0.54	0.47	10.86
	SC	0.30	0.70	1.71	1.93	0.48	1.20	1.74	0.45	0.34	0.82
	RB	0.17	0.51	1.09	1.13	0.55	1.06	1.51	0.61	0.59	1.73
	RI	0.15	0.35	0.69	0.71	0.27	0.51	0.70	0.24	0.20	0.35

story either, since the time to generate the equations may still be small in comparison with the solver time for e.g. problems with several distinct resources. We will next consider such a case.

4.2 Multiple resource unitary resource consumption example

In this section we explore the models on a more complex scheduling problem derived from the same domain as those above. In this case we extracted all the traffic through an area around the town of Hässleholm in southern Sweden. The area consists of 21 distinct resources of which 12 are unitary (track) resources, 2 are large stations with capacities of 24 and 16 respectively and the rest are smaller stations and track segments with a capacity of either one or two. Starting from a feasible timetable consisting of 5972 individual tasks, we reconstructed the precedence relations for all the jobs (trains) and relaxed the start times of all tasks to slack sizes of 50, 70 and 90 minutes respectively. The resulting problem properties and run time statistics is summarised in table 3. For each problem, the resulting number of cliques, the maximum and average clique size is given and then, for each model, the number of booleans, integers and equations generated and run time to produce an optimal solution is given. The last column gives the time to generate the equations for this experiment.

For all problems the MC method is again clearly the best, even if we include the time taken to generate the equations.

4.3 Single resource arbitrary resource consumption examples

To test and compare the two models that effectively handle tasks with arbitrary resource consumption we generated a set of random problems with different number of tasks, upper bounds on latest completion and slack. For each such problem size we generated 10 problems and attempted to solve each with the two methods with a time limit of 15 minutes.

Table 3. Problem and solution statistics for the 21 resource problem

Slk	Clqs	Mx	Av	Method	Bools/Ints	Eqtns	Solv Tm	Gen Tm
50	2 538	10	2.64	MC	18 766/0	32 737	5.14	2.93
				SC	37 404/0	70 071	25.62	5.46
				RB	14 365/0	30 946	6.08	2.78
				RI	28 149/4 244	42 284	20.16	2.35
70	2 652	13	3.41	MC	28 196/0	47 802	11.27	4.65
				SC	56 173/0	101 126	154.22	9.32
				RB	19 553/0	42 733	36.01	3.89
				RI	42 294/4 527	61 144	73.64	3.30
90	2 672	15	4.06	MC	36 280/0	61 514	22.64	6.67
				SC	72 404/0	127 937	252.87	13.63
				RB	23 677/0	52 382	78.55	5.07
				RI	54 420/4 588	77 312	134.69	4.24

Table 4. Run times for a set of random problems with varying resource consumption

				Clq Sz		MC		SC	
Tasks	Cpct	End	Slack	Mx	Av	Failed	Avr rnTm	Failed	Avr rnTm
20	3	50	10	7	3	0	0.01	0	0.02
			15	9	4	0	0.05	0	0.23
			20	10	5	0	4.41	1	71.06
			25	14	7	2	60.10	6	115.33
			30	14	7	2	101.01	6	44.23
			35	13	8	3	183.02	9	483.98
			40	13	9	4	136.71	7	379.66
			45	16	9	6	297.42	9	78.71
			50	17	11	5	152.75	10	-
30	3	75	10	8	4	0	0.05	0	0.56
			15	10	4	0	28.73	2	4.85
			20	10	6	2	52.94	6	150.41
			25	12	7	1	233.90	7	268.85
			30	14	7	8	21.70	10	-
			35	16	8	9	277.75	10	-
50	3	150	10	9	3	0	2.46	0	13.25
			15	8	4	0	11.68	1	37.01
			20	12	5	5	141.06	9	316.92
			25	12	6	7	210.24	10	-

Each row in table 4gives the number of tasks, the capacity of the resource, the latest end time and the maximum slack size of the problem class and reports the maximum and average clique sizes for the ten generated problems. For each model MC and SC, we then give the number of problems (out of 10) we *failed to solve in the allotted time* (15 minutes) and the *average solver run time* for the problems were we *did* manage to find and prove the optimal solution.

All the problems were fairly tight, with the sum of task surfaces generally covering between 85 and 100% of the resource area. Slack sizes were also randomly generated from a given (non-optimal) solution but limited by a maximum time window. These properties make these examples quite different from those from the train domain that consist of huge amounts of tasks but with small slack sizes.

We can see again that the methods exploit the given problem structure very well but that performance degrade quickly as the clique maximum sizes increase above around 10. The clique maximum and average size are clearly functions of the slack in the start time of each task. The larger the slack, the more tasks potentially overlap which is precisely what the clique size measures.

Once more, the MC model is clearly the best in terms of run time of the solver and in the number of solutions proved optimal. The accumulated time to generate the equations for each class of problems was in this experiment small (< 4 seconds) in comparison with the solver run time and, somewhat surprisingly, very similar for the two models, even for the more difficult problems.

To explore the relative scaling of the two methods with respect to equation generation/filtering time more closely, we also studied the effect of increasing the slack for a set of larger randomly generated problems. We fixed the number of tasks to 200, the latest end time to 600 and the resource capacity to 5. Plotting *only the time to generate* the equations against the maximum slack for the two models, yielded the graph in figure 1. Each entry in the plot represents the

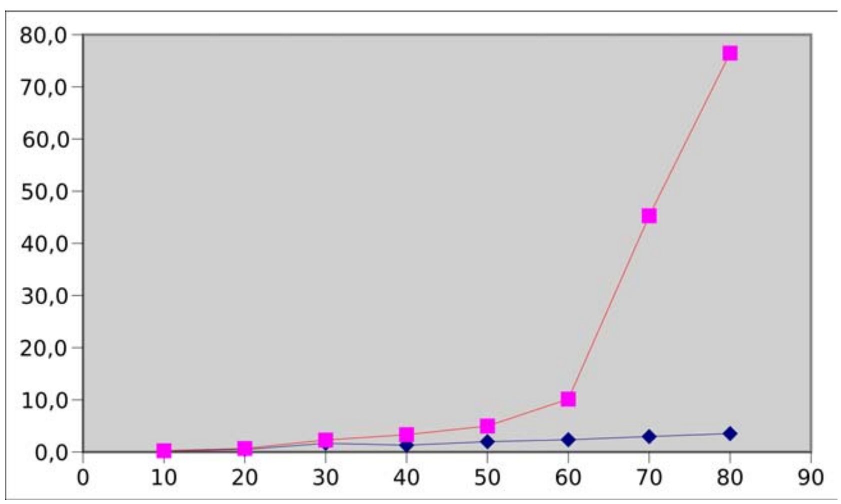

Figure 1. Time in seconds to generate the equations for the two models (MC=squares, SC=diamonds) against increasing start time slack size

mean of 10 random problems of each slack size, from 10 to 80.

Here the exponential growth for the MC model is more clearly visible but already for a slack of 60, typical max/average clique sizes are around 20/15 and the number of booleans for SC is about 9000. For problems of this size the solver time completely dominates the total time. Going up to even larger cliques, i.e. above max/average 30/20 , the generator (a Prolog program) runs out of memory for MC, so this method is no longer an option. The value of SC would still have to be questioned for problems of this size since the solver would most likely spend hours and probably days, to find solutions in such cases. However, it may still be of value for other types of problems, though at this point we have not found a way to characterise such a class of problems.

4.4 Large scale real world application

All the models described in this in these papers were originally developed as alternatives to an earlier CP-based scheduling system for train timetable generation [24,4,25] but for the full size version of this problem we have thoroughly investigated only the MC model of section 3.3. The test runs were performed on a number of problems selected from the real train timetable generation problem of the Swedish rail system for two consecutive years, 2004 and 2005.

One set of problems was extracted from the actual timetable for 2004 and then relaxed with respect to departure times. Tracks are considered unitary resources except in the case of single track lines which accommodate trains in both directions (see [4] for details) while stations were modelled as cumulative resources accommodating from 2 up to some 20 simultaneous trains.

Included in this set was a large area around the most important shunting yard in Sweden, Hallsberg. This problem consists of 175 tracks and 146 stations, 2 821 trains and around 60 000 tasks. The start time for each task was relaxed ±15 minutes from a given solution and precedence and resource constraints were generated, resulting in a very large problem but where the size of each individual clique was fairly small. Finding a feasible solution to this problem with CPLEX 9.0 took about 70 seconds on IBM Thinkpad T42 with a single core 1.8 GHz i686 processor. A second smaller problem generated in the same way, consisting of some 24 000 tasks, was solved in 27 seconds on the same machine.

A second set of problems was extracted from the capacity requests from the various rail traffic operators for the following year. Since the capacity requests come from several different and unrelated sources we typically have many unresolved resource conflicts at the start of the planning process. For this problem we again introduced a slack of ±15 minutes for the start time of the stated requirement. For one sub-problem consisting of some 15 000 tasks and with 149 unresolved conflicts, a partial solution with only 2 remaining conflicts was generated in about 100 seconds.

For the problem in the area around Hallsberg in this set we also tried allowing the system to introduce new low priority resource conflicts[1] where it would help to eliminate the 137 original high priority conflicts. In this case we introduced a smaller slack of ±5 minutes. All high priority conflicts were eliminated in 40 seconds of execution time at the cost of introducing only *one* new low priority conflict.

The largest single problem we approached consists of most of the traffic in the northern part of the country, with 3 643 trains, almost 199 620 tasks on 661 tracks and 611 stations. Initially the data contained 1 030 high priority conflicts. Running CPLEX 9.0 on a faster 2.6GHz Xenon processor for about 600 seconds eliminated all high priority conflicts and introduced 6 new low priority conflicts. Running the solver for several days on this problem we were able to prove that no solution exists with less than 4 such low priority conflicts.

[1] i.e. between certain cargo trains for which the uncertainty in actual arrival times and tolerance for smaller delays was larger.

5 Conclusion

We have introduced two MILP models for the general cumulative scheduling constraint and compared them to two for the special case where resource consumption is unitary based on geometric placement models. For each of these, we have defined pre-processing filters and compared solver performance on up to three sets of problems.

In all the experiments, the solver performance of one of the general cumulative models, the "Minimum conflicting sub-clique (MC) model", is clearly the best in terms of solver time. For this model, the filtering mechanism has exponential time complexity in general but in practice this has little impact on total time to generate and solve the problem. This is so, at least, for the type of problems considered, since the filtering time becomes significant only for problems where the solver would struggle to find any integer solution.

We also report briefly on a full scale industrial scheduling problem where the MC model is used to produce feasible schedules for several hundred thousands of tasks on thousands of resources. These problems are solvable only because the start time window of each task is small and the potential number of overlaps between tasks on each resources are often orders of magnitude smaller than the total number of tasks. For such problems the filtering proposed methods are very efficient.

References

1. Ahuja, R.K., Liu, J.N., Orlin, James B.and Sharma, D., Shughart, L.A.: Solving real-life locomotive-scheduling problems. Transportation Science **39**(4) (November 2005)
2. Aronsson, M., Kreuger, P., Gjerdrum, J.: An efficient mip model for locomotive scheduling with time windows. In Jacob, R., Müller-Hannemann, M., eds.: ATMOS 2006 - 6th Workshop on Algorithmic Methods and Models for Optimization of Railways. Number 06002 in Dagstuhl Seminar Proceedings, DagstUhl, Germany, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006) <http://drops.dagstuhl.de/opus/volltexte/2006/683> [date of citation: 2006-01-01].
3. Vaidyanathan, B., Ahuja, R.R., Jian, L., Shughart, L.A.: Real-life locomotive planning: New fromulations, algorithms and computational results. Technical report, Innovative Scheduling (2006) Revised: January 3, 2006.
4. Kreuger, P., Carlsson, M., Sjöland, T., Åström, E.: Sequence dependent task extensions for trip scheduling. Technical Report T2001:14, SICS (2001)
5. Törnquist, J.: Railway Traffic Disturbance Management. PhD Thesis, Blekinge Institute of Technology (2006) Doctoral dissertation series No. 2006:03.
6. Caseau, Y., Laburthe, F.: Improved CLP scheduling with task intervals. In van Hentenryck, P., ed.: Proceedings of the eleventh International Conference on Logic Programming ICLP'94. Volume 78., Santa Margherita Ligure, Italy, MIT Press (1994)
7. Baptiste, P., Le Pape, C.: A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In: Proceedings of the fourteenth international joint conference on artificial intelligence, Montreal, Quebec (1995) 400–606
8. Caseau, Y., Laburthe, F.: Improving branch and bound for job shop scheduling with constraint propagation. Technical report, Laboratoire d'Informatique de l'Ecole Normale Supérieure LIENS, Département de Mathématiques ed d'Informatique, 45 rue d'Ulm, 75232 Paris Cedex 05, France (1996)
9. Caseau, Y.: Using constraint propagation for complex scheduling problems: managing size, complex resources and travel. In Smolka, G., ed.: CP'97 – Principles and Practise of Constraint Programming. Volume 1330 of LNCS., Linz, Austria, Springer-Verlag (Oct/Nov 1997)
10. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling. Kluwer Academic Publishers, Norwell, MA, USA (2001)

11. Christodoulou, N., Stefanitsis, D., Kaltsas, E., Assimakopoulos, V.: A constraint logic programming approach to the vehicle-fleet scheduling problem. In: Proceedings of Practical Applications of Prolog. (1994)
12. Chiu, C., Chou, C., Lee, J., Leung, H., Leung, Y.: A constraint-based interactive train rescheduling tool. In: ? (1996) 104–118
13. Goltz, H.J., John, U.: Methods for solving practical problems of job-shop scheduling modeled in clp(*FD*). In: Proceedings of Practical Applications of Constraint Technology (PACT'96), PA (Apr 1996)
14. Gosselin, V.: Train scheduling using constraint programming techniques. In: 13th conference on AI, expert systems and natural language, Avignon (1993)
15. Carlsson, M., Kreuger, P., Åström, E.: Constraint-based resource allocation and scheduling in steel manufacturing. In: Practical Aspects of Declarative Languages. Volume 1551 of Lecture Notes in Computer Science., San Antonio, Springer Verlag (Januari 1999) 335–349
16. Caseau, Y., Laburthe, F.: Cumulative scheduling with task intervals. Technical report, Laboratoire d'Informatique de l'Ecole Normale Supérieure LIENS, Département de Mathématiques ed d'Informatique, 45 rue d'Ulm, 75232 Paris Cedex 05, France (1995)
17. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. Mathematical Computer Modelling **17**(7) (1993) 57–73
18. Beldiceanu, N., Carlsson, M.: Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In: CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, London, UK, Springer-Verlag (2001) 377–391
19. Caseau, Y., Laburthe, F.: Cumulative scheduling with task intervals. In: Joint InternationAl Conference and Symposium on Logic Programming. (1996) 363–377
20. Beldiceanu, N.: Global constraints as graph properties on a structured network of elementary constraints of the same type. In: Principles and Practice of Constraint Programming. (2000) 52–66
21. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical computer modelling **20**(12) (1994) 97–123
22. Blazewicz, J., Dror, M., Weglarz, J.: Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research **51**(3) (April 1991) 283–300
23. Keha, A.B., Khowala, K., Fowler, J.W.: Mixed integer programming formulations for single machine scheduling problems. Comput. Ind. Eng. **56**(1) (2009) 357–367
24. Kreuger, P., Carlsson, M., Olsson, J., Sjöland, T., Åström, E.: The TUFF train scheduler – Trip scheduling on single track networks. In: The proceedings of the Workshop on Industrial Constraint-Directed Scheduling at the Third International Conference on Principles and Practise of Constraint Programming, Schloss Hagenberg, Linz, Austria, Ed. A. Davenport (1997)
25. Sjöland, T., Aronsson, M., Kreuger, P.: Heterogeneous scheduling and rotation. In Kakas, A., Sadri, F., eds.: In Computational Logic: Logic Programming and Beyond, Part I. Lecture Notes in Artificial Intelligence. Springer-Verlag (2003) 655–676