REPORTS
IN
INFORMATICS

Modeling Safety Constraints in Oil and Gas Maintenance Scheduling

Bård Henning Tvedt   **Marc Bezem**

Department of Informatics

UNIVERSITY OF BERGEN
Bergen, Norway

Modeling Safety Constraints in Oil and Gas Maintenance Scheduling

Bård Henning Tvedt[*]   Marc Bezem[†]

June 14, 2011

Abstract

Scheduling problems in the oil industry are characterized by a large number of absolute safety constraints. These require an extension of the well-known jobshop model with cumulative resources. We discuss novel strategies for dealing with these safety constraints and demonstrate their usefulness on a set of benchmark problems.

1 Motivation

An oil and gas operator's objectives are to minimize the number of hazardous situations, minimizing the environmental footprint and maximize production. Meeting these, sometimes conflicting, targets is a complicated and challenging task, and meticulous preparations in form of planning and scheduling are done within all disciplines. These include drilling & completion, reservoir & production and operations & maintenance.

The particular scheduling situations depend on the location and age of a field, and on the discipline in question. At an offshore field, for instance, many maintenance activities are to be performed in a relatively small, enclosed space, while on a land-based field a great many maintenance activities are scheduled across vast areas. Even though the particulars of the problems themselves are different, several aspects are common for most scheduling problems in the industry. These aspects include:

- size – the number of activities can be in the range from hundreds to tens of thousand, making the schedule intractable;

- complexity – a large number of constraints to be satisfied, making the generation of a feasible schedule difficult;

- dynamics – the operating conditions are continuously changing, and the dependencies on weather, logistic and equipment failure *will* disrupt schedules.

Today's scheduling strategies are naturally shaped by these aspects. The size of the problem has historically been reduced by splitting the problem into several discipline- or department-related subproblems. Each separate problem becomes easier to solve, but the interdependencies between them are lost. The resulting schedules may therefore be unfeasible right from the start, and conflict resolution requires good communication between several departments and companies.

Even though the size of the problems are reduced, there is no guarantee that the same applies for the complexity. Several of the considerations taken into account are based on the scheduler's extensive knowledge of the problem, and not on formalized constraints. There is widespread belief that many of these considerations and best practices are difficult to implement in a solving strategy. This perception may be correct, but has not been properly tested.

---

[*]Epsis AS, P.O.Box 27, N-5863 Bergen, Norway, and University of Bergen, Department of Physics and Technology

[†]University of Bergen, Department of Informatics, P.O.Box 7803, N-5020 Bergen, Norway

Another implication of size and complexity is quantification of the schedules quality with respect to one or several objectives. If it is difficult to ensure the feasability of one schedule, how should several schedule suggestions be compared to determine the best solution?

The dynamics of the problem is a considerable challenge during execution of a schedule. Unforeseen events will occur. Due to the limited time available to reorganize a disrupted schedule, the simple strategy of postponing activities is often applied. Such a strategy can be combined with scheduling initially with slack in order to make room for some distortion. This might be beneficial if the schedule experiences the right amount of distortion during execution. However, if the execution is smooth resources become idle and new tasks must be prioritized and assigned on the fly. On the other hand, there is no guarantee that distortions larger then the induced slack cannot occur. In such cases the schedule is rendered unfeasible and must be rescheduled.

The challenges mentioned above can be met by applying automated scheduling algorithms providing a consistent strategy for generating feasible schedule suggestions. To fully utilize the existing potential, several of the considerations made by schedulers must be turned into formal constraints. Some of these can be modeled and solved by well known scheduling techniques, while others, for example safety constraints, must be handled by tailor-made techniques as discussed in this paper. The expected benefits are greater control and overview of safety issues, better prioritizing of activities, reduced idle time of resources, improved regularity of operations and thus increased production.

2   Problem description

Our main focus in this paper is the maintenance and operations domain. The problem definition has been simplified as much as possible while still keeping industry-specific characteristics like safety constraints. More common scheduling elements like cumulative resources and dependency constraints are included to provide real-world resemblance. The solution objective is to minimize the *makespan*, the total completion time of all activities in the schedule. This particular objective aims at efficiency, but the techniques developed can be applied to other objectives as well.

The problem is set at an imaginary oil-platform subdivided into a set of locations. Equipment in need of maintenance is randomly distributed across the platform, and various maintenance activities are to be scheduled. The activities are created with a given set of resource requirements and potential dependencies to other activities. All activities require a crew to perform them and a location to be performed at. In addition some activities require crane resources because heavy lifting is involved. The crew and crane resources are scarce, meaning that they are in limited supply.

Thus far the problem classifies as a 'Resource-Constrained Project Scheduling Problem' [1], which is described by:

- a set of resources with given capacities;

- a set of non-interruptible activities of given processing times;

- a network of precedence constraints between the activities;

- the amount of resources required by the activities.

Even though the RCPSP description includes a wide range of scheduling problems there are several additional constraints, typical for the oil-industry as well as for other large-scale industrial settings, which do not fit into this framework. The objective when generating a set of problem instances was to ensure that formalized examples of industry-related constraints were involved. An example is safety considerations surrounding dangerous work like for instance crane usage. Today such schedule information is included manually by the schedulers. By formalizing the prerequisites that activate the safety constraints we get a well-defined problem description. A solution to a problem $S(P_i)$ is a schedule where all activities are assigned a start time and all constraints are satisfied.

In the following paragraphs we will take a detailed look at the problem instances described in the above setting.

2.1 Notation and Terminology

A problem instance P consists of activities to be performed, resources needed to perform activities and constraints representing limitations between activities and resource use. We distinguish between *decision variables*, given *constants* and *derived variables*. An example of a decision variable is the starting time of an activity Act_i denoted $v_{sta}(Act_i)$. An activity's duration is assumed to be fixed and therefore a constant denoted $c_{dur}(Act_i)$. Finally, there are derived variables like an activity's end time, which is the sum of it's start time and duration, given by $w_{end}(Act_i)$. Real objects like activities and resources are written with a capital letter.

2.2 Resources

A *location* $Loc_l \in Locs = \{Loc_1, \ldots, Loc_n\}$ is the place where activities are being executed. Even though the locations are viewed as resources there are no restrictions to the number of activities that can be performed simultaneously on a location. However, when dangerous work like heavy lifting is performed, the location resource is inaccessible to all other activities. When a location is closed due to crane usage we say that a safety zone has been erected. How to apply safety zones is an important part of the different solution heuristics discussed in chapter 3.

The *crews* are responsible for executing the activities. A crew is denoted $Crew_j \in Crews = \{Crew_1, \ldots, Crew_n\}$ The resource demand is taken to be one for the duration of the activity. A crew can therefore simultaneously work on the same number of activities as its capacity. It is not possible to pool resources to reduce the duration of an activity.

A *crane* $Crane_k \in Cranes = \{Crane_1, \ldots, Crane_n\}$ is a potential resource for activities. Some of the activities need a crane, and all problem instances contains a small number of cranes. The cranes are unary resources meaning that they can only perform one activity at the time. An activity requiring a crane resource does not specify a particular crane, but simply that it needs a crane. A feasible solution must therefore assign one crane to all activities requiring crane from the set of cranes available, i.e. $v_{crane}(Act_i) \in Cranes$. This makes the set of cranes an alternative resource [6].

Cranes have a location $c_{loc}(Crane_k) \in Locs$, and each location can only have one crane. Due to the hazardous nature of heavy lifting, crane usage is surrounded by safety zones. These safety zones are erected at both the location where the activity requiring the crane is performed and the crane's own location. The safety zones erected will therefore vary according to which crane is assigned to the activity.

2.3 Activities

An *activity* $Act_i \in Acts = \{Act_1, \ldots, Act_n\}$ comes with a start variable, a constant duration and resource requirements. Initially the domain of the start variable is $v_{sta}(Act_i) \in [0, c_{hor}(P))$, where the horizon, indicating the schedule's maximum completion time, is given as $c_{hor}(P) = \sum_i c_{dur}(Act_i)$.

An activity Act_i requires a crew $c_{crew}(Act_i) \in Crews$ to perform it, and a location $c_{loc}(Act_i) \in Locs$ to be performed at. The activity demand one single member of a crew and it is not possible to pool resources to reduce the duration. Cranes are the final resource kind that is available, but it is not needed by all activities.

In addition to resource requirements an activity can depend on other activities, meaning that it can not start before some other activities are finished.

2.4 Constraints

Dependencies between activities are common in the industry. A maintenance activity can for instance be dependent on both delivery of spare parts and raising of scaffolding to ensure access to the area in question. The relation stating that activity $Act_{i'}$ depends on activity Act_i is expressed by the following constraint:

$$w_{end}(Act_i) \leq v_{sta}(Act_{i'}) \tag{1}$$

A *cumulative resource constraint* is applied for all the crews ensuring that the total resource consumption is not exceeding the available capacity. It is expressed by:

$$\forall t, j \ : \ \sharp\{Act_i \mid t \in [v_{sta}(Act_i), w_{end}(Act_i)) \wedge c_{crew}(Act_i) = Crew_j\}$$
$$\leq c_{cap}(Crew_j) \tag{2}$$

where $c_{cap}(Crew_j)$ is the capacity of the jth crew.

Cranes are unique individuals and therefore modeled as a set of *unary resource constraints*. The constraint states that if two activities are assigned to the same crane then they can not be executed simultaneously. We start by defining the predicate *overlap* expressing that two activities are overlapping in time:

$$overlap(Act_i, Act_{i'}) \equiv$$
$$\exists t \ : \ v_{sta}(Act_i), v_{sta}(Act_{i'}) \leq t < w_{end}(Act_i), w_{end}(Act_{i'}) \tag{3}$$

The mutual exclusion created by the unary resource constraint then becomes:

$$\forall i, i' \neq i \ : \ v_{crane}(Act_i) = v_{crane}(Act_{i'}) \rightarrow \neg overlap(Act_i, Act_{i'}) \tag{4}$$

for all activities requiring crane.

The last kind of constraints is the *safety constraints*. They are expressed in terms of the location of the activity requiring crane and of the location of the chosen crane. The first location is known in advance, while the second is dependent on the decision as to which crane to use. The fact that the constraints in the problem change according to the decisions made is interesting because of the added complexity they induce.

Safety constraints shutting out use of the location of an activity requiring crane are:

$$\forall i, i' \neq i \ : \ c_{crane}(Act_i) \wedge c_{loc}(Act_i) = c_{loc}(Act_{i'}) \wedge \neg overlap(Act_i, Act_{i'}) \tag{5}$$

while safety constraints shutting out use of this crane's location are given by:

$$\forall i, i' \neq i : v_{crane}(Act_i) = Crane_j \wedge c_{loc}(Act_{i'}) = c_{loc}(Crane_j)$$
$$\rightarrow \neg overlap(Act_i, Act_{i'}) \tag{6}$$

2.5   Objective

The objective to minimize makespan $w_{ms}(P)$ or the duration of the schedule is defined by:

$$w_{ms}(P) = \max_i\{w_{end}(A_i)\} \in [0, c_{hor}(P)] \tag{7}$$

which states that the makespan equals the latest end or completion time in the set of activities.

2.6   Problem instances

The problems are characterized by their sizes determined by the total number of activities, $\#Acts \in \{50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 750, 1000\}$, and cranes, $\#Cranes = [2, 3]$. A total of 5 problem instances were generated for each of the 24 problem sizes, summed up to a total of 120 instances.

The problem instances were not necessarily feasible since they were generated by randomly assigning crews to activities, locations to activities, locations to cranes, dependencies between activities, and activities' crane requirements. 12 problem instances contained circular dependencies, and 11 instances had more than one crane at the same location. The total number of instances used is 97.

All the problem instances contain 24 locations and 4 kinds of crews with capacity $c_{cap}(Crew_j) \in [2, 3]$ drawn from an uniform distribution. The domain for the activities' start variables are default $v_{sta}(Act_i) = [0, c_{hor}(P))$, and the constant durations $c_{dur}(Act_i)$ is randomly pulled from an uniform distribution in the range of $[1, 6]$ time steps. Approximately 20% of the activities are randomly chosen to require crane, and approximately 10% of the activities are constrained by dependencies to some other activity.

3 Solution strategies

The problem of determining whether a RCPSP solution with makespan smaller than a given deadline exists is NP-hard [2]. This means that the extended problem with safety constrains is also at least NP - hard, and therefore an optimal solution to even the simplest form of the problem is not guaranteed in polynomial time. One none-search-based and one search-based strategy are tested.

The first solution strategy is a non-search based greedy algorithm. A solution is created by assigning activities sequentially to the earliest time slot satisfying the activity's resource demands. This strategy does not backtrack unmaking decisions exploring other possibilities, but it is however guaranteed to produce a feasible solution.

The second set of solution strategies is implemented in C++ using the Ilog Solver 6.6 [4] and Ilog Scheduler 6.6 [3] libraries. A default search goal Ilog : IloSetTimesForward, which is designed to efficiently schedule activities in a chronological order, is applied to the activities, while the default goal Ilog : IloGenerate is used to assign cranes to activities. The search based strategy are constrained by a time limit in the range of seconds to produce a solution. The search is restarted with an additional constraint stating that the makespan of the new solution is less than the current solution, and terminates when it fails to improve on the solution within the time limit. This strategy is applied to four slightly different ways of modeling the safety constraints.

To evaluate the quality of the solutions theoretical lower and upper bounds for the makespans are calculated.

3.1 Theoretical lower bound for makespan

A theoretical lower bound is calculated based on the resource availability for the most constrained crew. The resulting solutions may not be feasible for the entire problem, but are packed tightly for the crew utilizing every crew hour.

$$c_{load}(Crew_j) = \sum_{c_{crew}(Act_i)=Crew_j} c_{dur}(Act_i) \tag{8}$$

$$c_{relload}(Crew_j) = \frac{c_{load}(Crew_j)}{c_{cap}(Crew_j)} \tag{9}$$

$$c_{lb,ms}(P) = \max_j\{c_{relload}(Crew_j)\} \tag{10}$$

3.2 Theoretical upper bound for makespan

Our theoretical upper bound for the makespan is

$$c_{ub,ms}(P) = c_{hor}(P) = \sum_i c_{dur}(Act_i) \tag{11}$$

indicating that in a worst case scenario all activities can be executed one at the time in a sequential order.

3.3 Non-search based strategy

The first approach that can be expected to give some results is a so-called greedy algorithm. A characteristic of such an algorithm is that it schedules the activities one by one, always taking the first available time-slot and (if required) crane such that all constraints are satisfied. Clearly, one has to respect precedence to get a feasible solution and strong heuristics are required to avoid the worst schedules. This is in some sense the classical approach avoiding search: efficient generation of inefficient schedules. Thus it serves very well as a reference method by which the quality of other methods can be measured. Note that the method is not unlike manual scheduling, the crucial difference being scale as well as quality of the heuristics.

Let us explain one of the heuristics which is often applied. Among the resources, it is often possible to identify the one that is most scarce. In our setting, although the safety constraints are complicating, the cranes are not a scarce resource since only 20% of the activities uses a crane. To identify a scarce resource

one has to sum the total duration of all activities using the resource and divide that sum by the resource's capacity, much in the same way as the theoretical lower bound of the makespan is computed in Section 3.1. Having identified the scarcity of the resources in this way, it is often beneficial to schedule activities using a scarce resource first. Of course this should be done in a way compatible with the dependencies.

A first and necessary preparation for a greedy algorithm is all activities can be linearly ordered in a way respecting the precedence relation. This is a well-known procedure called 'topological sort' [5] and can be done in time quadratic in the number of activities. Topological sort detects circular dependencies. From now on we assume that the activities $Act_1, \ldots, Act_n$ are listed such that $i < j$ whenever activity Act_j depends on Act_i. A feasible schedule can then be computed by scheduling each Act_j at the first possible time such that Act_i is finished (if Act_j depends on Act_i), the necessary resource is available and there is a free crane (if Act_j uses a crane), all satisfying the safety and other constraints.

3.4 Search based strategy

The second solution strategy is based on exhaustive search, and is applied to four variations of the model. The changes to the model are introduced to increase efficiency.

3.4.1 Inferred constraints

Solving the problem instances to optimality turned out to be difficult even for the smallest instances. The default ILOG Scheduler search strategies have great performance, but they struggle to find feasible solutions to the problems due to the non - chronological elements induced by the decision dependent safety constraints. To reduce the effect of the non - chronological elements implicit model information is made explicit through additional constraints. The model is expanded with a *cumulative resource constraint* for cranes:

$$\forall t \ : \ \sharp\{Act_i \mid t \in [v_{sta}(Act_i), w_{end}(Act_i)) \wedge c_{crane}(Act_i)\} \leq c_{cap}(Cranes) \tag{12}$$

where $c_{cap}(Cranes)$ equals the number of cranes. The search strategy will attempt to assign start times to activities before cranes are assigned to the activities. The inferred cumulative constraint ensures that the crane capacity is never exceeded.

3.4.2 Underconstrained model

In the underconstrained model the safety constraints are relaxed to fully utilize the chronological search strategy. The fact that an activity Act_i is performed at a location Loc_l is a priori information, but it is not decided which crane that is to perform an activity, i.e., $v_{crane}(Act_i)$ is not initially assigned. The crane's location, which cannot be used for reasons of safety, is therefore unknown. The safety constraints related to an activity's location are given by (5), while the safety constraints related to a crane's location are given by (6). The underconstrained model is reduced to only incorporate the safety constraints in (5). The resulting schedules are not necessarily feasible solutions to the original problem due to the relaxation of constraints.

The motivation for the underconstrained model is twofold. First it is to test the hypothesis that removing non-chronological elements will increase the size of problem instances we are able to solve, and secondly, if they can be solved to optimality, it will provide an improved lower bound for the makespan. The latter turned out to be difficult with the predefined objective Ilog : IloMinimize, which extensively traverses the search tree to ensure optimality. Other implementation strategies or other software like SMT [7] may prove more successful.

3.4.3 First overconstrained model

The first overconstrained model expresses all the safety constraints as non-conditional a priori statements. The conditional statement depending on which crane $v_{crane}(Act_i)$ is in other words bypassed. A feasible schedule to the original model can be generated if the safety constraints shut out use of all crane locations regardless of which crane that is in use. The safety constraints in the original model expressed in constraint (6) are replaced by:

$$\forall i, i' \neq i : c_{crane}(Act_i) \wedge c_{loc}(Act_{i'}) \in Craneloc \cup \{c_{loc}(Act_i)\}$$
$$\rightarrow \neg overlap(Act_i, Act_{i'}) \tag{13}$$

where $Craneloc = \{c_{loc}(Crane_j) \mid Crane_j \in Cranes\}$. Notice that it is only activities and not cranes that are influenced. Instead of forcing a solution to use one crane it encourages to use several cranes simultaneously. It is in other words not similar to setting the cranes capacity to 1.

3.4.4 Second overconstrained model

The overconstrained model in the previous section applies to all activities executed at a crane location. However, if one of these activities also requires a crane the resulting safety constraints will not only affect the activities at crane locations, but also the utilization of the other cranes. In these special cases the crane capacity is in reality reduced to 1. To avoid this, activities that require crane and simultaneously are to be executed at a crane location are forced to be executed by the crane at that location, i.e.:

$$\forall i, j \ : \ c_{crane}(Act_i) \wedge c_{loc}(Act_i) = c_{loc}(Crane_j)$$
$$\rightarrow v_{crane}(Act_i) = Crane_j \tag{14}$$

Constraint (14) ensures that the activity and crane used is at the same location, and the generation of safety zones is therefore handled by constraint (5). These constraints do not conflict with activities at other crane locations, but constraint (13) must also be relaxed to ensure that activities requiring other cranes do not exclude these activities. Constraint (13) is therefore replaced by:

$$\forall i, i' \neq i \ : \ c_{crane}(Act_i) \wedge c_{loc}(Act_{i'}) \in craneloc \wedge \neg c_{crane}(Act_{i'})$$
$$\rightarrow \neg overlap(Act_i, Act_{i'}) \tag{15}$$

4   Results

The upper and lower bounds provide the range for the makespan. Due to the way instances are generated and the nature of the bounds this range is as expected. The durations are randomly pulled from an uniform distribution with mean $\bar{c}_{dur}(Act_i) = 3.5$ units of time. The expected upper bounds are approximately 175 and 3500 units of time for 50 and 1000 activities, respectively. The mean upper bounds of the problem instances are 170 and 3500 units of time.

Based on 4 crews where activities are randomly assigned from an uniform distribution and with a minimum capacity of 2 the lower bounds are expected to be 1/8 of the upper bounds. For 50 and 1000 activities the expected lower bounds are approximately 22 and 438, respectively, while the mean values in the problem instances are 23 and 440.

A measurement of relative quality is used to evaluate the results from the different strategies and models. The derived variable w_{rq} is given in (16).

$$w_{rq} = \frac{1}{|\mathcal{P}_{sol}|} \sum_{P \in \mathcal{P}_{sol}} \frac{w_{ms}(P)}{c_{lb,ms}(P)} \tag{16}$$

where $\mathcal{P}_{sol}$ is the set of problem instances solved by each strategies. The elements in $\mathcal{P}_{sol}$ differs from strategy to strategy and these averages are therefore not entirely comparable, but they give an indication of the quality of the solutions. w_{rq} does not penalize a strategy or model for failing finding a solution, but robustness is shown by the number of solved instances. The results are summarized in Table 1.

A feasible solution with makespan equal the theoretical lower bound may not exist. However, the theoretical lower bound is based on the most constrained crew which is the problem's most constrained resource. It is therefore considered a sensible benchmark to compare our strategies to.

A possible solution with makespan equal the theoretical lower bound is not necessarily violated by introducing a less constrained crane resource, but a combination of safety constraints, resource capacities

Table 1: Relative optimality index w_{rq} for the various models

Model		2 Cranes				3 Cranes				All	
# act's (# pr's)		≤ 100 (25)		> 100 (23)		≤ 100 (28)		> 100 (23)		(99 problems)	
Sect.	Model	w_{rq}	$\%^{(2)}$	w_{rq}	$\%^{(2)}$	w_{rq}	$\%^{(2)}$	w_{rq}	$\%^{(2)}$	w_{rq}	$\%^{(2)}$
3.3	Greedy	1.311	100	1.524	100	1.276	100	1.393	100	1.370	100
3.4.1	Inferred	1.068	100	1.063	35	1.017	32	1.000	4	1.055	43
3.4.2	Under [1]	1.032	100	1.014	100	1.015	100	1.002	96	1.016	99
3.4.3	Over #1	1.143	100	1.076	100	1.176	100	1.037	91	1.114	98
3.4.4	Over #2	1.104	100	1.047	96	1.068	96	1.005	65	1.063	90

[1] This model does not guarantee feasible solutions   [2] percentage solved problem instances

and location availability can increase the makespan of an optimal solution. This is supported by the results from the underconstrained model. The relative quality is overall only 1.6 % larger than the theoretical lower bound. This effect is mainly assigned to the safety constraints surrounding activities that need crane that is not incorporated in the theoretical lower bound. The underconstrainted model does not incorporate all constraints and the results are not guaranteed feasible solutions to the original problem.

The inferred model produces the overall best makespans for feasible schedules, approximately 5.5% larger then the theoretical lower bound. On the other hand this model scales poorly and struggles when the problems become larger and more complex. In the larger problems with 3 cranes it only solves 1 of the 23 instances, and overall it solves only 43% of the problem instances.

The second best makespans for feasible solutions are obtained by the second overconstrainted model and are about 6.3% larger then the theoretical lower bound. It has less difficulties with scaling then the inferred model, but starts to struggle for the larger problems instances and leaves 10% of the problem instances unsolved.

Not surprisingly the first overconstrained model produces slightly worse make-span. This is expected since it is the most constrained model in the search based strategy. The obtained makespans are approximately 11.4% larger then the theoretical lower bound. The model is however more robust than the other two and 98% of the instances are solved.

Our greedy solution strategy produces solutions with makespan on average 37 % larger than the theoretical lower bound. The strategy mimics a manual scheduling procedure for generating feasible schedules, and the results are believed to be comparable. The greedy algorithm's strength is that it always generates a solution. It is in fact the only strategy that solves all the problem instances.

5 Conclusion

The objective of our work was to address some of the challenges related to maintenance scheduling. More precisely, to explore different strategies as to handling large and complex problems, in the highly dynamic environment of oil and gas production, in an efficient and coherent way.

Our non-search based greedy strategy solves all problem instances, regardless of size and complexity, within minutes. The short solution times enable consistent handling of disruptions, caused by for instance unexpected events or delays, through rescheduling. In this manner the dynamics of the problem are also taken into account.

The greedy algorithm mimics a manual procedure for generating a feasible schedule from scratch. Even though the search based strategies are not as robust, the one that performed best generates solutions with on average a 20% shorter makespan. We believe our results indicate a great unutilized potential with respect to quality and generation time of schedules, to be exploited by automated scheduling.

6   Future work

The solution strategies applied in this paper are relatively simple, and improvements to quality, robustness and solving time are expected of more sophisticated strategies. Among the ideas that are expected to yield improvements:

- more informed activity selection by improving heuristics for greedy algorithm;

- more informed activity and resource selection by replacing default Ilog search goals with goals customized for the problem at hand;

- increased and more dispersed coverage of the solution space by applying local search techniques.

References

[1] P. Baptiste, C. le Pape, and W. Nuijten. *Constraint-Based Scheduling, Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 1 edition, 2001. 2

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1 edition, 1979. 3

[3] Ilog. *Ilog Scheduler's User Manual*, 2008. 3

[4] Ilog. *Ilog Solver's User Manual*, 2008. 3

[5] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5:558–562, November 1962. 3.3

[6] C. le Pape and S. F. Smith. *Temporal aspects in Information Systems*, chapter Management of Temporal Constraints for Factory Scheduling. North-Holland, 1988. 2.2

[7] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(*T*). *Journal of the ACM*, 53(6):937–977, 2006. 3.4