Minimization of Useless Work in Resource Failure Recovery of Workflow Schedules

Marek Vlk*‡, Roman Barták* and Zdeněk Hanzálek†‡
*Charles University, Faculty of Mathematics and Physics
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
Email: {vlk,bartak}@ktiml.mff.cuni.cz
Tel.: +420724880797
†Czech Technical University in Prague, Faculty of Electrical Engineering
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
‡Czech Technical University in Prague, Czech Institute of Informatics, Robotics and Cybernetics
Zikova 1903/4, 166 36 Praha 6, Czech Republic
Email: zdenek.hanzalek@cvut.cz

***Abstract*—Real-life scheduling has to face many difficulties such as dynamics of manufacturing environments with unforeseen events occurring during the execution of a schedule. Namely, in the case of a resource failure, it may be necessary to process a lot of work again, or a feasible schedule recovery may not exist at all. Moreover, the time window within which the ongoing schedule must be updated may be very short, and too time-consuming computation of the schedule may lead to a failure of the scheduling mechanism and setback in production. Our approach in the area of predictive-reactive scheduling is to allow for substitution of tasks, which cannot be executed, with a set of alternative tasks. This paper makes use of the model of the hierarchical workflows and gives an SMT and a CSP models to recover an ongoing schedule from a resource failure with the objective to minimize the work processed in vain. The experimental analysis identified parameters for which the SMT model clearly outperforms the CSP model and vice versa.**

1. Introduction

Since the unexpected events such as machine breakdowns inevitably occur on the shop floor during the execution of production schedules, the reactive rescheduling is gaining considerable interest. In the area of predictive-reactive scheduling, it is important to take into consideration not only the original objective function but also an appropriate definition of robustness measures. This paper works with the model of the hierarchical structure of tasks with alternatives that is suitable also in the field of predictive-reactive scheduling as it allows the possibility to substitute a set of jobs with another set of jobs. In response to a resource failure, the aim considered in this paper is to find a new feasible schedule that minimizes the work performed in vain.

Reactive scheduling differs from predictive scheduling mainly by its on-line nature and associated real-time execution requirements. The schedule update must be accomplished before the running schedule becomes invalid, and this time window may be very short in some environments. In this paper, modeling the problem in the formalism called Satisfiability Modulo Theories (SMT) [1] is proposed and this approach is experimentally compared to modeling the problem as a Constraint Satisfaction Problem (CSP).

The following section briefly sets this work into context. Section 3 then describes the problem tackled in this paper. The proposed SMT and CSP models are described in Sects. 4 and 5, respectively. The experimental analysis is given in Sect. 6, and the final part points out possible future work.

2. Related Work

There are basically two approaches to tackle dynamics of the scheduling environment, distinguished according to whether or not the baseline (predictive) schedule is computed beforehand [2]. If the jobs are assigned to resources pursuant to some dispatching rules during the execution, it is referred to as *completely reactive scheduling* or on-line scheduling. If the baseline schedule is crafted before the execution starts and then updated during the execution, the approach is referred to as *predictive-reactive scheduling*. When correcting the ongoing schedule in response to changes within the environment, the schedule modification is often considered as the minimization objective.

There is an extensive literature on rescheduling [3], [4], [5], giving various generic approaches as well as ad-hoc procedures to particular cases. While there is a great amount of work devoted to planning with time and resources and to integrating planning and scheduling techniques, the research carried out aiming at the possibility of re-planning in the field of predictive-reactive scheduling is rare.

The scheduling model described further is based on the notion of *workflows*. Workflow, in general, may be understood as a scheme of performing some complex process, itemized into simpler processes and relationships among them. There exist many formal models to describe workflows [6] that include decision points and loops to describe repetition of operations, but in real-life applications, many

workflows are obtained by decomposition of tasks, which is the motivation for the hierarchical structure of workflows [7]. This paper considers workflows that have a very similar structure to Temporal Planning Networks [8] but are enhanced by extra constraints and renewable resources.

Thanks to the logical constraints in a workflow, the process selection is suitable for Boolean Satisfiability (SAT) [9]. Other constraints for computing a schedule, namely temporal constraints, are linear inequalities so that it is suitable for SAT Modulo Theories (SMT) [1], using the linear integer arithmetic as the background theory. SMT solvers have already been used for solving CSP and combinatorial optimization problems [10], including some scheduling problems [11], and exhibited promising performance.

3. Scheduling Model

In this work, the model of workflows from the FlowOpt system [12] is used. Such workflows match up the structure of Nested Temporal Networks with Alternatives [13], where the nodes of a network correspond to the tasks of a workflow. The tasks decompose into other tasks, referred to as their subtasks. There are two types of decomposition: *parallel* and *alternative*. The tasks that do not decompose further (i.e., leaves) are called *primitive tasks*. The primitive tasks correspond to executable activities and are associated with some additional parameters, namely duration, cost, and resource demands. Only non-interruptible primitive tasks are considered.

The workflows as described define a number of feasible processes. A *process* is a subset of tasks selected to be processed. While a parallel task requires all its children to be processed, an alternative task requires exactly one of its children to be processed. If an arbitrary task is not in the process, none of its subtasks is in the process either. Hence, to ensure that an instance of a workflow is actually processed, its root task has to be in the selected process. An example of a workflow and its decompositions tree along with a selected process are depicted in Figs. 1 and 2 [14]. It contains eight primitive tasks, three parallel tasks, and two alternative tasks.

The nested structure may not be flexible enough to describe naturally some additional relations in real-life processes, for example when an alternative for one task affects the selection of alternatives in other tasks. In order to simplify the description of these additional relations between tasks, a pair of tasks can be bound by a *logical constraint*. Logical constraints include *implications* (if one task is in the process, the other task must be in the process too), *equivalences* (either both tasks must be in the process or neither of them can be in the process), and *mutual exclusions* (at most one of the tasks can be in the process). Further, there are simple temporal constraints [15] to determine the maximum (as well as minimum) time distance between two tasks, provided that both tasks are selected to the process.

Primitive tasks are processed on *resources*. Each resource is specified by its capacity (availability), which is the maximum allowed sum of demands of primitive tasks that are to be processed on that resource at the same time.

Finally, a resulting schedule is *feasible* if all constraints are satisfied.

3.1. Formal Definition of Feasible Schedule

Formally, a scheduling problem S consists of three sets: $Tasks$, $Constraints$, and $Resources$.

3.1.1. Tasks. The set $Tasks$ is a union of three disjoint sets: $Parallel$, $Alternative$, and $Primitive$. For each task T except the $root$ task, function $Parent(T)$ denotes the parent task in the hierarchical structure. Similarly for each task T, the set $Subtasks(T)$ of its child nodes is defined as $Subtasks(T) = \{T_i \in Tasks \mid Parent(T_i) = T\}$.

The tasks from sets $Parallel$ and $Alternative$ are called compound tasks and they decompose to some subtasks, whereas the primitive tasks do not decompose:

$$\forall T \in Parallel \cup Alternative : Subtasks(T) \neq \emptyset$$
$$\forall T \in Primitive : Subtasks(T) = \emptyset$$

Let process $P \subseteq Tasks$ be the set of tasks selected to be processed. To make the process feasible, it is necessary to satisfy the following constraints:

$$\forall T \in P \cap Parallel : Subtasks(T) \subseteq P \quad (1)$$
$$\forall T \in P \cap Alternative : |Subtasks(T) \cap P| = 1 \quad (2)$$
$$\forall T \notin P : Subtasks(T) \cap P = \emptyset \quad (3)$$
$$root \in P \quad (4)$$

Note that without constraint (4), an empty set would be a feasible process.

Let S_i and E_i denote the start time and end time, respectively, of task T_i. Each primitive task T_i is specified by the duration D_i, and the cost W_i. The times for compound tasks are computed from the times of their subtasks, which involves the following constraints:

$$\forall T_i \in P \cap Primitive : E_i = S_i + D_i \quad (5)$$

$$\forall T_i \in P \cap (Parallel \cup Alternative) : S_i = \min\{S_j \mid T_j \in Subtasks(T_i) \cap P\} \quad (6)$$

$$\forall T_i \in P \cap (Parallel \cup Alternative) : E_i = \max\{E_j \mid T_j \in Subtasks(T_i) \cap P\} \quad (7)$$

3.1.2. Extra constraints. There are basically two types of constraints: logical and temporal. Logical constraints are of three types: implications ($i \Rightarrow j$), equivalences ($i \Leftrightarrow j$), and mutexes ($i \; mutex \; j$). The semantics of the constraints is as follows:

$$\forall (i \Rightarrow j) : T_i \in P \Rightarrow T_j \in P \quad (8)$$
$$\forall (i \Leftrightarrow j) : T_i \in P \Leftrightarrow T_j \in P \quad (9)$$
$$\forall (i \; mutex \; j) : T_i \notin P \vee T_j \notin P \quad (10)$$

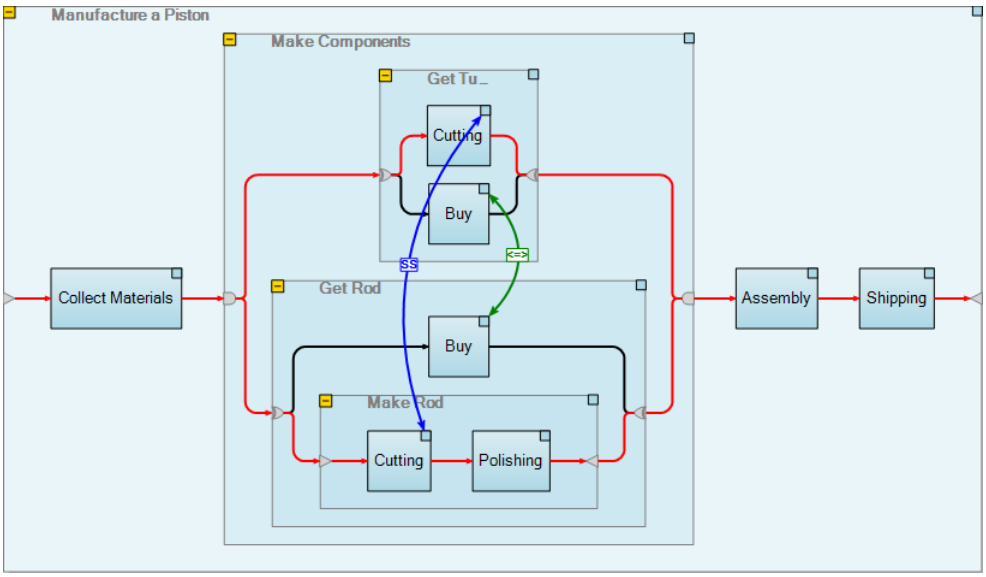

Figure 1. An example of a workflow.

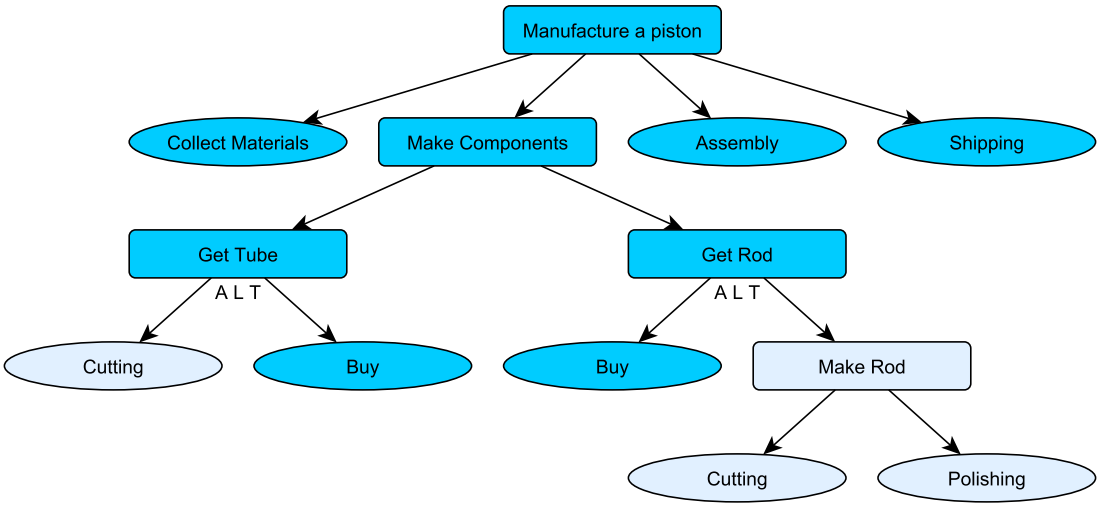

Figure 2. A decomposition tree for the workflow. The label "ALT" beneath tasks stands for alternative decomposition; the other decompositions are parallel. Primitive tasks are ellipse-shaped. The highlighted tasks form an example of a selected process.

The time distance between two distinct primitive tasks may be restricted by a *simple temporal constraint* [15]. This constraint can be written as a triplet (X_i, X_j, l_{ij}), where $l_{ij} \in \mathbb{Z}$, X_i is either S_i or E_i, and X_j is either S_j or E_j. The semantics is as follows:

$$\forall (X_i, X_j, l_{ij}) : T_i \in P \wedge T_j \in P \Rightarrow X_j - X_i \leq l_{ij} \quad (11)$$

3.1.3. Resources. Each resource R_k is specified by its capacity B_k and the amount of resource R_k that is demanded by the primitive task T_i is denoted $b_{i,k}$.

Let us define $Exec(R_k, t)$ as the set of primitive tasks that are to be executed at time t on resource R_k, that is, $Exec(R_k, t) = \{T_i \in Primitive | S_i \leq t < E_i \wedge b_{i,k} > 0\}$. In a feasible schedule, the following must hold:

$$\forall R_k \in Resources, \forall t \in \mathbb{Z} : \sum_{T_i \in Exec(R_k, t)} b_{i,k} \leq B_k \quad (12)$$

3.2. Schedule

A schedule S (sometimes referred to as a resulting schedule or a solution) is acquired by determining the set P, and allocating the tasks from P in time, that is, assigning particular values to the variables S_i and E_i for each $T_i \in P$.

To make a schedule *feasible*, the process selection and the allocation must be conducted in such a way that all the constraints (1)–(12) in the problem are satisfied.

3.3. Resource Failure Recovery Problem

The problem actually tackled is that a particular instance of the scheduling problem is given along with a feasible schedule (referred to as an original schedule), and also with a change in the input data, which corresponds to the disturbance arising from the shop floor. The aim is to find another schedule that is feasible in terms of the new problem definition.

In what follows, only the disruption called a resource failure (also referred to as a machine breakdown) is addressed. The resource failure may happen in a manufacturing system at any point in time, say t_f, and means that the capacity of a particular resource drops to 0, i.e., for all $t \geq t_f$ there is no primitive task allocated to that resource.

Formally, let Sch_0 be the original schedule, with P_0 being the selected process in Sch_0, $S_i(Sch_0)$ and $E_i(Sch_0)$ the start times and end times in Sch_0; and R_f be the resource that failed at some time $t_f : t_f \in \mathbb{Z}, t_f \geq 0$. Next, let us define $Processed$ as the set of primitive tasks that are from P_0 and, whose execution has finished if they use resource R_f or whose execution has at least started if they are allocated to available resources only, that is, $T_i \in Processed$ if and only if:

$$T_i \in Primitive \cap P_0 \ \wedge ((E_i(Sch_0) \leq t_f \wedge b_{i,f} > 0) \vee (S_i(Sch_0) < t_f \wedge b_{i,f} = 0))$$

Finally, the aim is to find a new feasible (recovered) schedule Sch_1, with P_1 being the selected process in Sch_1, $S_i(Sch_1)$ and $E_i(Sch_1)$ the start times and end times in Sch_1, subject to the following constraints:

$$\forall T_i \in Processed, b_{i,f} = 0 : \quad T_i \in P_1 \Rightarrow (S_i(Sch_1) = S_i(Sch_0) \vee S_i(Sch_1) \geq t_f) \tag{13}$$

$$\forall T_i \in Processed, b_{i,f} > 0 : \quad T_i \in P_1 \Rightarrow S_i(Sch_1) = S_i(Sch_0) \tag{14}$$

$$\forall T_i \in Primitive \setminus Processed, b_{i,f} > 0 : \quad T_i \notin P_1 \tag{15}$$

$$\forall T_i \in Primitive \setminus Processed, b_{i,f} = 0 : \quad T_i \in P_1 \Rightarrow S_i(Sch_1) \geq t_f \tag{16}$$

The constraint (13) considers the primitive tasks defined above as $Processed$ and that required only resources that are still available. If the primitive task in question is also in the new process, it is either fixed in time or it is to be executed again, that is, scheduled in future. The primitive tasks processed on a failed resource are handled similarly by the constraint (14), except those primitive tasks cannot be re-executed because the resource R_f failed. The constraint (15) ensures that each primitive task that is not in $Processed$ and required the failed resource is not in the new process P_1. And for the remaining primitive tasks the constraint (16) ensures that no primitive task will be scheduled to the past.

Recall that W_i is the cost of processing the primitive task T_i. The aim is to find a new schedule Sch_1 which minimizes the cost of tasks (work) that have been processed in vain. This is equivalent to maximization of the cost of tasks that are reused in the new schedule. Formally, a primitive task $T_i \in Processed$ has been processed *usefully* if and only if $T_i \in P_1 \wedge S_i(Sch_1) = S_i(Sch_0)$. Let us define $Useful \subseteq Processed$ as the set of primitive tasks that have been processed usefully. Hence, the objective is to maximize the following function:

$$z = \sum_{T_i \in Useful} W_i \tag{17}$$

4. Modeling as Satisfiability Modulo Theories

In this section, the model of the resource failure recovery problem in the SMT formalism is described. Recall that the task of an SMT solver is to find a satisfying assignment of a formula φ or report unsatisfiability. Such formula is usually represented in a Conjunctive Normal Form (CNF), i.e., as the conjunction of clauses, where clauses are disjunctions of literals corresponding to expression in a particular *theory*, in our case difference logic. To ease the description, the clauses are listed in the default form given by their semantics. Such clauses can be transformed to a CNF.

The application interface of the Z3 solver [16] is used. This solver provides the *Optimizer*, which allows the addition of weighted soft clauses to the formula, and then minimizes the sum of weights of soft clauses that are not satisfied by the assignment.

For each task $T_i \in Tasks$, let us introduce propositional variables V_i that will be $true$ if and only if $T_i \in P_1$, and numeric (non-negative integer) variables S_i and E_i determining the start time and end time of task T_i. The SMT formula φ to be satisfied is constructed as follows.

For each compound task T_i, let us denote $Subtasks(T_i) = \{T_{i_1}, ..., T_{i_k}\}$. If $T_i \in Parallel$, then for each $j = i_1, ..., i_k$ the following unit clause is added to the formula φ:

$$(V_i = V_j) \tag{18}$$

If $T_i \in Alternative$, the following clause is added:

$$\left(ite(V_i; 1; 0) = \sum_{j=i_1}^{i_k} ite(V_j; 1; 0) \right) \tag{19}$$

Note that the expression $ite(V_i, 1, 0)$ is evaluated as 1 if V_i is $true$, otherwise 0. It is straightforward to verify that

satisfying the clauses (18) and (19) enforces the satisfaction of constraints (1)–(3). Note that modeling the constraint (2) in a purely logical way, i.e., using disjunction and at-most-one clauses instead of clause (19), did not lead to any measurable difference in run times. To satisfy also the constraint (4), assuming that the $root$ task is T_1, the unit clause is added:

$$(V_1) \tag{20}$$

As to the time allocations for primitive tasks, that is, satisfying the constraint (5), it is enough to add the following clauses:

$$(E_i = S_i + D_i) \tag{21}$$

The time allocations for compound tasks is not that straightforward as the functions min and max are not in the formal specification of SMT. Hence, for a parallel task, the clauses are added to ensure that the start time of the task is always less than or equal to its subtasks, and that it equals one of its subtasks. A similar construction is used for the end times. Formally, for each $T_i \in Parallel$:

$$(V_i \Rightarrow \bigvee_{j=i_1}^{i_k} S_i = S_j) \tag{22}$$

$$(V_i \Rightarrow \bigvee_{j=i_1}^{i_k} E_i = E_j) \tag{23}$$

And for each $j = i_1, ..., i_k$:

$$(V_i \Rightarrow S_i \leq S_j) \tag{24}$$

$$(V_i \Rightarrow E_i \geq E_j) \tag{25}$$

Since alternative tasks have at most one subtask in the process, it suffices to add, for each $T_i \in Alternative$, and for each $j = i_1, ..., i_k$:

$$(V_j \Rightarrow S_i = S_j) \tag{26}$$

$$(V_j \Rightarrow E_i = E_j) \tag{27}$$

It is easy to see that the satisfaction of clauses (22)–(27) ensures satisfying the constraints (6) and (7).

The logical constraints (8)–(10) in the format of $(i \rightarrow j)$, $(i \leftrightarrow j)$, and $(i\ mutex\ j)$ are handled by adding the following clauses, respectively:

$$(V_i \Rightarrow V_j) \tag{28}$$

$$(V_i \Leftrightarrow V_j) \tag{29}$$

$$(\neg V_i \vee \neg V_j) \tag{30}$$

Next, for each temporal constraint (11) (X_i, X_j, l_{ij}), where $l_{ij} \in \mathbb{Z}$, X_i is either S_i or E_i, and X_j is either S_j or E_j, the following clauses are added:

$$((V_i \wedge V_j) \Rightarrow X_j - X_i \leq l_{ij}) \tag{31}$$

The capacities of resources are encoded following the idea of so-called *task formulation* that was presented in [11], which is similar to the *task-resource decomposition* given in [17]. The key idea is that it suffices to check the usage of a resource only for the start times of the activities allocated to the resource, hence unlike in *time formulations* the number of constraints is independent of the length of the scheduling horizon. The constraints (12) are then handled by adding the following clauses, for each $T_j \in Primitive, R_k \in Resources, b_{j,k} > 0$:

$$\left(V_j \Rightarrow B_k - b_{j,k} \geq \sum_{\substack{T_i \in Primitive \\ i \neq j \\ b_{i,k} > 0}} ite(V_j \wedge V_i \wedge S_i \leq S_j \wedge S_j < E_i; b_{i,k}; 0)\right) \tag{32}$$

Note that the V_j in the sum is redundant but it led to a slight speed-up.

If a primitive task from the set $Processed$ that does not need the failed resource is in the new process, it is either not shifted in time or it is to be processed again in the future, that is, to satisfy the constraint (13), add for each $T_i \in Processed, b_{i,f} = 0$:

$$(S_i = S_i(Sch_0) \vee S_i \geq t_f) \tag{33}$$

Similarly, to satisfy the constraint (14), add for each $T_i \in Processed, b_{i,f} > 0$:

$$(S_i = S_i(Sch_0)) \tag{34}$$

To satisfy the constraint (15), add for each $T_i \in Primitive \setminus Processed, b_{i,f} > 0$:

$$(\neg V_i) \tag{35}$$

And to satisfy the constraint (16), add for each $T_i \in Primitive \setminus Processed, b_{i,f} = 0$:

$$(S_i \geq t_f) \tag{36}$$

Note that the clauses (21), (33), (34), and (36) do not need the implication form $(V_i \Rightarrow expr)$, because in case $V_i = false$, the values of the time points S_i and E_i do not affect other variables and thus may be assigned arbitrary values (satisfying the clauses) by an SMT solver, which turned out to be slightly faster than the implication form.

Finally, the optimization objective z is maximized by adding the following *soft* clause (of weight W_i) for each $T_i \in Processed$:

$$(V_i \wedge S_i = S_i(Sch_0)) \tag{37}$$

5. Modeling as Constraint Satisfaction Problem

For modeling the problem in the CSP formalism, the IBM CP Optimizer [18] is used, which provides a modeler with so-called interval variables that also have support for hierarchical decomposition, that is, interval variables may

be optional. The hierarchical structure was modeled using the specialized constraints *alternative* for alternative tasks, and *span* and *equivalences* for parallel tasks. The capacities of resources are handled using the *pulse* constraints, which make it very easy to model.

The objective is to maximize the sum:

$$\sum_{T_i \in Processed} W_i \cdot (Presence(T_i) \wedge Start(T_i) = S_i(Sch_0))$$

The rest is straightforward.

6. Experimental Results

In this section, the SMT model using the Z3 solver is experimentally evaluated. The Z3 solver allows to set several parameters including the engine for the maximal satisfiability. The best engine turned out to be the one set by parameter `wmax` [19]. As to the IBM CP Optimizer, the only adjusted parameter of the optimizer was `DefaultInferenceLevel`, which was set to `Extended`.[1]

The problems are randomly generated as follows. First, the desired number of primitive tasks is generated and the hierarchical structure of tasks is built. Then the process P_0 is found, which may be done in linear time when there are no extra logical constraints, and then the primitive tasks from the process P_0 are allocated on resources in such a way that each primitive task, one at a time, is allocated to the earliest possible start time, which is again easy when there are no extra temporal constraints. Afterward, the desired number of extra logical constraints as well as extra temporal constraints is added in such a way that the already crafted schedule is still feasible. However, randomly selecting a resource to fail may cause that there is no solution.

The experiments were performed with the number of primitive tasks fixed to 100 and the dependence of the runtime on the number of extra constraints was observed. The instances are composed of 5 resources, the capacities of which are randomly generated numbers at least 1 and at most 10. Durations and costs of primitive tasks are random numbers greater than 0 and less than 15, resource demands are limited to one resource for each primitive task and the size of a demand is at least 1 and at most the capacity of the resource. Each compound task has 2–5 subtasks. In order to maximize the number of processes in a workflow and thus make the problems harder, a compound task is an alternative task if the task has at least one primitive task as its subtask, else it is a parallel task.

Logical constraints are implications or mutexes with equal probabilities $\frac{1}{2}$. Temporal constraints are added in pairs $(X_i, X_j, dist)$ and $(X_j, X_i, -dist)$, where each of the time points X_i and X_j is start time or end time with equal probabilities $\frac{1}{2}$, $dist = X_j - X_i$, if $T_i, T_j \in P_0$, else it is a randomly generated number less than the makespan of the original schedule. Adding temporal constraints in pairs with zero slack is motivated by real-life problems containing synchronization constraints. The time t_f of a resource failure is set to be the latest end time of a primitive task on failed resource R_f divided by 2.

1. The models along with all the source code can be downloaded from https://drive.google.com/drive/folders/0BxOh5Kp8klV3UGZUaDFUTUNtcjg?usp=sharing

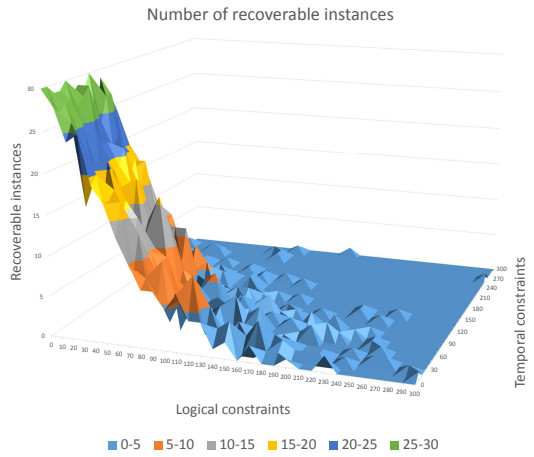

Figure 3. The number of instances with a feasible solution as the function of the number of logical constraints (horizontal axis) and the number of temporal constraints (depth axis).

The solvers were run on a Dell PC with an Intel(R) Core(TM) i7-4610M processor running at 3.00 GHz with 16 GB of RAM.

Figure 3 shows the number of instances (out of 30) with a feasible schedule recovery as the function of the number of logical constraints (horizontal axis) and the number of temporal constraints (depth axis), both parameters ranging from 0 to 300 stepped by 10. It is clear that the more constraints are added to the problem, the fewer instances are recoverable.

Figures 4 and 5 show the number of instances on which the runtime of the SMT solver and the CSP solver, respectively, exceeded one second. It is clear that with the increasing number of added logical constraints, the problem becomes easier for both approaches. However, the most difficult problems for SMT are those containing a smaller number of temporal constraints (30–60), while the most difficult problems for CSP are those containing a high number of temporal constraints (more than 100), as shown in Fig. 6, which depicts the number of one-second excesses aggregated over all numbers of logical constraints.

The total number of one-second excesses was 1101 for SMT and 1795 for CSP. When the time limit is set to one minute, SMT solves all problems in the given limit, whereas the number of one-minute excesses for CSP is still very high (1712) and the graph looks almost the same. This phenomenon happens also for very small instances: CSP either solves a problem in the order of milliseconds or does not solve it in a reasonable time. The deeper analysis discovered that the reason is a temporal inconsistency between a task and some of its ancestors, which the CSP engine is not able to efficiently detect and repeatedly tries instantiating

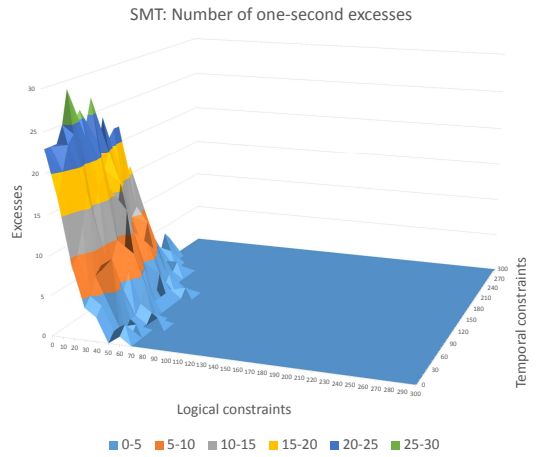

Figure 4. The number of instances that the SMT solver took more than one second to solve as the function of the number of logical constraints (horizontal axis) and the number of temporal constraints (depth axis).

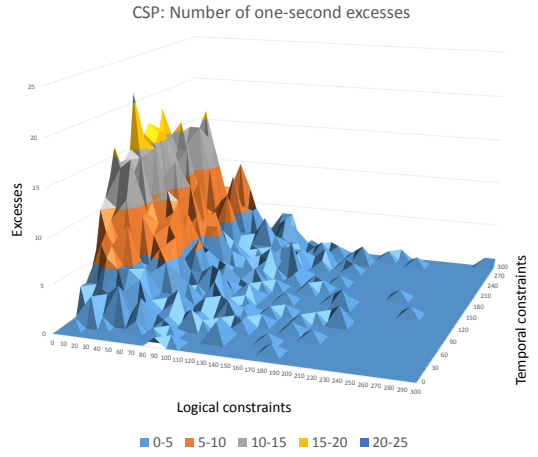

Figure 5. The number of instances that the CSP solver took more than one second to solve as the function of the number of logical constraints (horizontal axis) and the number of temporal constraints (depth axis).

values from too large domains. In these cases, limiting the start (end) times of interval variables helps, but obtaining a reasonable bound on makespan is difficult owing to the alternatives and the temporal constraints.

7. Conclusions

This paper describes the hierarchical model with alternatives that makes it possible to replace tasks in the process by other tasks that are not in the process, i.e., to re-plan some subset of the schedule. The main focus is on the resource failure recovery problem with the objective to minimize the useless work and the SMT and the CSP formulations were given. The experimental analysis identified parameters for which the SMT model clearly outperforms the CSP model and vice versa.

Figure 6. The number of instances that the solvers took more than one second to solve as the function of the number of temporal constraints (x-axis), aggregated over the numbers of logical constraints.

The future target is to propose a decomposition approach in SMT that would bring next speed-up. It is based on dividing the model into two subproblems, where one subproblem passes the reason of inconsistency to the other subproblem, which then adds the unsatisfiable core to the formula as a nogood clause, and another solution is sought. The hitch in dividing the variables naturally into two groups, one for the process selection and one for the time allocation, is that the objective function involves variables from both groups.

Acknowledgments

This research is partially supported by the EU and the Ministry of Industry and Trade of the Czech Republic under the Project OP PIK CZ.01.1.02/0.0/0.0/15_019/0004688, by the Charles University, project GA UK No. 158216, by SVV project number 260 453, and by the Czech Science Foundation under the project P103-15-19877S.

References

[1] 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 (JACM)*, vol. 53, no. 6, pp. 937–977, 2006.

[2] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: a framework of strategies, policies, and methods," *Journal of scheduling*, vol. 6, no. 1, pp. 39–62, 2003.

[3] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[4] A. Raheja and V. Subramaniam, "Reactive recovery of job shop schedules–a review," *The International Journal of Advanced Manufacturing Technology*, vol. 19, no. 10, pp. 756–763, 2002.

[5] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future directions," *European Journal of Operational Research*, vol. 161, no. 1, pp. 86–110, 2005.

[6] W. M. Van Der Aalst and A. H. Ter Hofstede, "Yawl: yet another workflow language," *Information systems*, vol. 30, no. 4, pp. 245–275, 2005.

[7] J. Bae, H. Bae, S.-H. Kang, and Y. Kim, "Automatic control of workflow processes using eca rules," *IEEE transactions on knowledge and data engineering*, vol. 16, no. 8, pp. 1010–1023, 2004.

[8] I.-h. Shu, R. T. Effinger, and B. C. Williams, "Enabling fast flexible planning through incremental temporal reasoning with conflict extraction." in *ICAPS*, 2005, pp. 252–261.

[9] C. A. Tovey, "A simplified np-complete satisfiability problem," *Discrete Applied Mathematics*, vol. 8, no. 1, pp. 85–89, 1984.

[10] M. Bofill, J. Suy, and M. Villaret, "A system for solving constraint satisfaction problems with smt," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2010, pp. 300–305.

[11] C. Ansótegui, M. Bofill, M. Palahı, J. Suy, and M. Villaret, "Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem," in *Proceedings of the 9th Symposium on Abstraction, Reformulation and Approximation (SARA 2011)*, 2011, pp. 2–9.

[12] R. Barták, M. Jaška, L. Novák, V. Rovenskỳ, T. Skalickỳ, M. Cully, C. Sheahan, and D. Thanh-Tung, "Flowopt: Bridging the gap between optimization technology and manufacturing planners," in *Proceedings of the 20th European Conference on Artificial Intelligence*. IOS Press, 2012, pp. 1003–1004.

[13] R. Barták and V. Rovenský, "On verification of nested workflows with extra constraints: From theory to practice," *Expert Systems with Applications*, vol. 41, no. 3, pp. 904–918, 2014.

[14] T. Skalický, "Interactive scheduling and visualisation," Master's thesis, Masters thesis, Charles University in Prague, 2011.

[15] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial intelligence*, vol. 49, no. 1, pp. 61–95, 1991.

[16] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[17] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace, "Why cumulative decomposition is not as bad as it sounds," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2009, pp. 746–761.

[18] P. Laborie, J. Rogerie, P. Shaw, P. Vilim, and F. Wagner, "Ilog cp optimizer: detailed scheduling model and opl formulation," Technical Report 08-002, ILOG, Tech. Rep., 2008.

[19] N. Bjørner and A.-D. Phan, "νz-maximal satisfaction with z3." in *SCSS*, 2014, pp. 1–9.