Modelling Alternatives in Temporal Networks

Roman Barták*, Ondřej Čepek*†, Pavel Surynek*

*Charles University, Faculty of Mathematics and Physics
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
{roman.bartak, ondrej.cepek, pavel.surynek}@mff.cuni.cz

†Institute of Finance and Administration
Estonská 500, 101 00 Praha 10, Czech Republic

Abstract

Temporal Networks play an important role in solving planning problems and they are also used, though not as frequently, when solving scheduling problems. In this paper we propose an extension of temporal networks by parallel and alternative branching. This extension supports modelling of alternative paths in the network; in particular, it is motivated by modelling alternative process routes in manufacturing scheduling. We show that deciding which nodes can be consistently included in this extended temporal network is an NP-complete problem. To simplify solving this problem, we propose a pre-processing step whose goal is to identify classes of equivalent nodes. The ideas are presented using precedence networks, but we also show how they can be extended to simple temporal networks.

Introduction

Scheduling problems typically deal with allocating known activities to available resources and time. Real-life problems are usually more complex than existing theoretical models and, for example, they also require selection among alternative process routes or alternative resources in complex manufacturing enterprises. Due to efficiency issues, selection of alternative processes and resource allocation are frequently done separately from scheduling. However, this also has several drawbacks. First, if the selected route or resource allocation cannot be scheduled, it is necessary to backtrack from the scheduling module to resource allocation and process selection modules. Second, even if the resource allocation and selected routes are feasible, separating the allocation and process selection algorithms from the scheduling algorithm may ruin the quality of the solution. Hence, a better result will be obtained when process selection and resource allocation is done within scheduling. While resource allocation is now an accepted part of scheduling problems and there exist approaches for doing resource allocation within scheduling, for example (Focacci, Laborie, and Nuijten 2000), process selection is still treated separately.

In this paper we propose an extension of temporal networks that can model alternative process routes. We describe the main ideas using networks with only precedence relations, but at the end of the paper we also show how these ideas can be extended to simple temporal networks. To model selection of alternatives, we assign a validity variable to each node in the network. This validity variable indicates whether the node is selected or not to be in the final solution plan. The decision about validity/invalidity of the node is done by the solver. We also augment the precedence network by a description of splitting and joining operations that implicitly define dependencies between nodes in the network. The dependency relations specify which nodes must/cannot be valid in relation to the validity status of other nodes. The main motivation for these operations goes from modelling manufacturing processes. The nodes correspond to activities (or more precisely start times of activities) and the arcs describe flow of products between the activities. In some nodes the manufacturing process can split into two or more parallel sub-processes that can join back to a single process. For example, a piece of wood is cut in parts that are processed in parallel and then assembled together to a final product. This is called *parallel branching*. Another form of branching is *alternative branching* when the process also splits in sub-processes, but these sub-processes are treated as alternatives so exactly one of them is used. The alternative sub-processes can also join back to a single process (Figure 1).

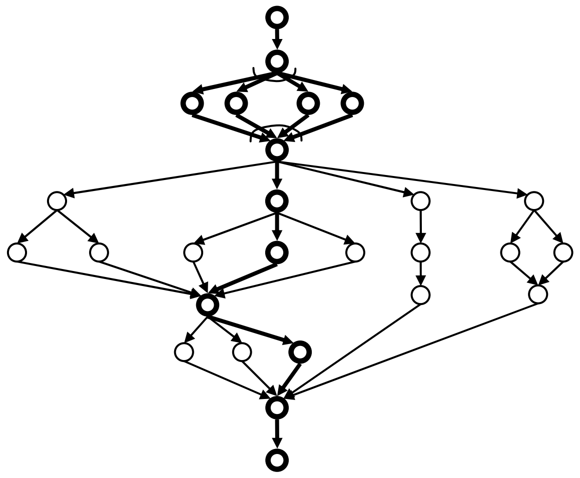


Figure 1. Graph of parallel (top) and alternative processes with a selected process.

In the paper we formally define the above-described precedence graph with parallel and alternative branching that we call a *P/A graph*. We also show that the problem whether there exists an assignment of validity variables consistent with the specified branching is NP-complete. Hence there is a little hope for a fast solving algorithm so we plan to model the problem as a constraint satisfaction problem. To support constraint modelling, we propose a pre-processing step where sets of equivalent nodes are identified in the P/A graph. We call the nodes equivalent, if their validity status is identical in all solutions, that is, such nodes are either all valid or all invalid in any consistent assignment of validity variables. This will help us to bridge alternative routes and consequently improve the constraint model. We conclude the paper by showing that the presented ideas can be extended to simple temporal networks. We compare our proposal with existing works on temporal networks. Namely, we demonstrate that our P/A simple temporal networks can model temporal constraint satisfaction problems.

P/A Graphs

Let G be an acyclic graph. A subgraph of G is called a *fan-out subgraph* if it consists of nodes x, $y_1, \dots, y_k$ (for some k) such that each (x,y_i), $1 \le i \le k$, is an arc in G. Similarly, a subgraph of G is called a *fan-in subgraph* if it consists of nodes x, $y_1, \dots, y_k$ (for some k) such that each (y_i,x), $1 \le i \le k$, is an arc in G. In both cases x is called a *principal node* and all $y_1, \dots, y_k$ are called *branching nodes*. G is called a *P/A graph* if the description of the graph (list of nodes and arcs) is accompanied by a list of pairwise edge-disjoint fan-out and fan-in subgraphs, where each subgraph on the list is marked either as a *parallel* subgraph or an *alternative* subgraph. An *assignment* of 0/1 (false/true) values to nodes of a given P/A graph is called *feasible* if

- in every parallel subgraph all nodes are assigned the same value (both the principal node and all branching nodes are 0 or both the principal node and all branching nodes are 1),
- in every alternative subgraph either all nodes (both the principal node and all branching nodes) are 0 or the principal node and exactly one branching node are 1 while all other branching nodes are 0.

It can be easily noticed that given an arbitrary P/A graph the assignment of the value 0 to all nodes is always feasible. On the other hand, if some of the nodes are required to take value 1 (as we shall see later, this requirement is a very natural one if the P/A graph is used to model a real-life problem), then the existence of a feasible assignment is by no means obvious. Let us now formulate this decision problem formally.

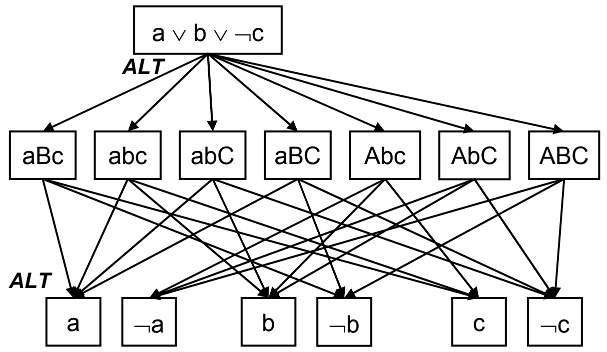
Definition: *P/A graph assignment problem* is given by a P/A graph G and a list of nodes of G which are assigned value 1. The question is whether there exist a feasible assignment of 0/1 values to all nodes of G which extends the prescribed partial assignment.

Remark: The above problem remains the same if we allow forcing the value 1 for just a single vertex. To see this, observe that the general case can be reduced to this special one by adding an extra vertex, forcing it to 1 and connecting it by a fan-in (or fan-out) parallel subgraph to all nodes that were forced to 1 originally. Moreover, if the original graph was acyclic, then so is the new one.

Proposition 1: The P/A graph assignment problem is NP-complete.

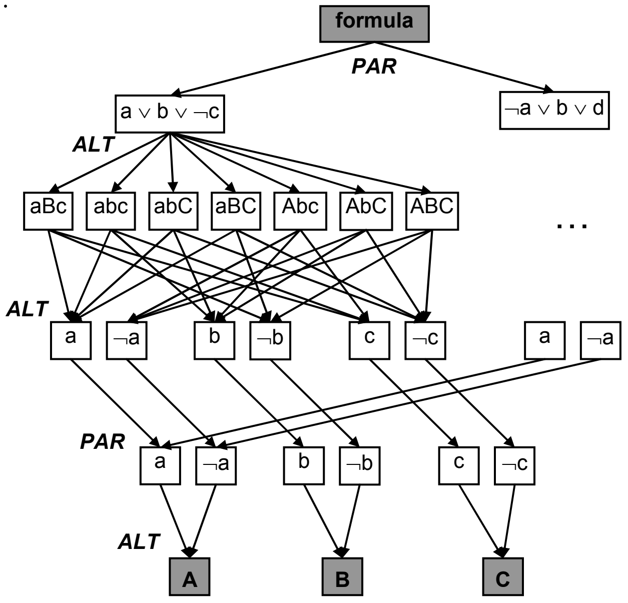
Proof: The problem is obviously in NP, because it suffices to guess the assignment and test its feasibility, which can be done in linear time in the number of parallel and alternative subgraphs (and hence in the number of edges). For the NP-hardness, we shall show that the 3SAT problem, which is known to be NP-complete (Garey and Johnson, 1995), can be reduced (in a polynomial time) to the P/A graph assignment problem. Recall that the 3SAT problem is a problem of deciding whether there exists a model (a satisfying assignment of truth values to propositional variables) for a given formula in a conjunctive normal form, where each clause in the formula consists of exactly three literals. Moreover we may assume that no variable appears twice in a single clause, i.e. each clause consists of literals of three distinct variables.

Now we shall describe how to construct, for a given CNF (an instance of 3SAT), an instance of the P/A graph assignment problem. Consider e.g. a clause $(a \vee b \vee \neg c)$. There exist seven mutually exclusive assignments of truth values to variables a, b, and c satisfying this clause (each assignment except of a=false, b=false, c=true is a satisfying one). We can model this clause using a "clause subgraph" which consists of a node for the clause, seven nodes for the mutually exclusive satisfying assignments, and six nodes for the values of propositional variables (three for positive values and three for negative values, i.e. one for each literal). The clause node is connected to all assignment nodes by a fan-out alternative subgraph and each value node is connected to appropriate assignment nodes (those assignment nodes containing the literal which corresponds to the given value node) by a fan-in alternative subgraph. The following figure shows the clause subgraph for the clause $(a \vee b \vee \neg c)$, where capital letters in the assignment nodes represent value false (so e.g. aBc corresponds to a=true, b=false, c=true).



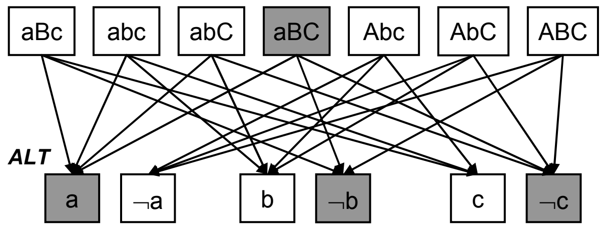
Each clause from the input CNF will be modelled using a clause graph with the above-described structure. To connect the clause graphs, we introduce a formula node and connect it with all clause nodes by a fan-out parallel subgraph. The formula node is forced to take the value 1 (because we need the formula to be satisfied). A variable which is used in more than one clause will have value nodes in all clause graphs where it appears. To interconnect these value nodes we introduce for each variable in the formula a variable node, which is forced to take the value 1, and two literal nodes connected to the variable node by a fan-in alternative subgraph. Finally, each literal node is connected by a fan-in parallel subgraph to all value nodes in clause graphs which correspond to the given literal. The following figure shows these additional nodes and connections (the shaded nodes are the nodes that are forced to take the value 1).

.



First let us observe that the number of nodes in the constructed P/A graph is linear in the size of the input CNF formula. Namely, if there are M clauses and N variables (and hence L = 3M literals) in the input CNF, then we get a graph with (14M + 3N + 1) nodes. Because 14M + 1 ≤ 5L and N ≤ L (assuming each variable appears at least once in the formula) we get that there are at most 8L nodes in the constructed P/A graph.

Now let us assume that the input CNF has a satisfying assignment. We shall construct a feasible assignment of the constructed P/A graph as follows. All clause nodes will get the value 1 to satisfy the parallel fan-out from the formula node. The literal nodes of each variable will get the 0 and 1 values as defined by the satisfying assignment of the input CNF (e.g. if variable b is false in the satisfying assignment, then the node b gets the value 0 and the node ¬b gets the value 1). This satisfies the alternative fan-in into the variable nodes, and moreover it defines the 0 and 1 values for all value nodes via the parallel fan-ins that replicate the literal values into all clause subgraphs. Finally, for each clause exactly one assignment node is made valid, namely the one in which all three literals are valid, which satisfies the alternative fan-out from the clause nodes. It remains to show that also all alternative fan-ins into value nodes are satisfied. So let us consider an arbitrary value node. If it corresponds to a valid literal then it is connected to exactly one valid assignment node (the one where also the other two literals are valid), and if it corresponds to an invalid literal then it is connected only to invalid assignment nodes (see figure below). In both cases this is exactly what we need and hence the constructed assignment of 0/1 values to all nodes is feasible.



To complete the proof let us assume that there exists a feasible assignment of 0/1 values to all nodes of the constructed P/A graph. In this assignment:

- All clause nodes have value 1 to satisfy the parallel fan-out from the formula node.
- For each clause exactly one assignment node has value 1 to satisfy the alternative fan-out from the clause nodes.
- For each variable one literal node has value 1 and the other has value 0 to satisfy the alternative fan-ins into the variable nodes. The literal values are replicated into the value nodes by the parallel fan-ins into the literal nodes.

Now let us check that the truth assignment defined by the values assigned to the literal nodes satisfies the input CNF. To this end let us pick an arbitrary clause and assume by contradiction that it is falsified. That means that the three valid value nodes correspond to the only missing combination among the assignment nodes, or in other words, that the valid assignment node must be connected to an invalid value node. However, this is a contradiction, because the corresponding fan-in subgraph into this value node spoils the feasibility of the assignment (the principal node is 0 while one of its branching nodes is 1). Hence, the input CNF has a satisfying assignment if and only if the constructed P/A graph has a feasible assignment.

Q.E.D.

P/A Graph Pre-processing

Because solving the P/A graph assignment problem is hard, we now focus on inferring some information from the graph that can be used later to improve solver efficiency. In particular, we describe an algorithm for finding equivalent nodes in the P/A graph. We call a set of nodes of the P/A graph *equivalent* if and only if the nodes are assigned the same value in all feasible assignments of 0/1 values to nodes. The problem of finding the largest possible sets of equivalent nodes is probably hard (but we have no formal evidence of it yet), therefore at this stage we focus on discovering certain typical situations only. The most important situation we want to recognize consists of a process which splits in several sub-processes in alternative branching and all these sub-processes join afterwards. Principal nodes where the production process splits and sub-processes join back again are equivalent. We are looking for the algorithm that can discover at least such equivalence classes.

The proposed algorithm has two phases. In the initial phase an undirected hyper-graph is constructed from the input P/A graph. The constructed hyper-graph has almost the same structure and represents almost the same information about the production processes as the original P/A graph. Only the directions of arcs and hence precedence relations are omitted, which is not a problem because the input P/A graph is acyclic so the precedence relations trivially hold (actually, the precedence relations are used only to define fan-in and fan-out subgraphs in acyclic P/A graphs).

The second and major phase of the algorithm repeatedly transforms the given hyper-graph using certain transformation rules into a simpler and more explicit hyper-graph. Sets of equivalent nodes of the input P/A graph are built along these transformation steps. This phase terminates when no transformation rule can be applied or when a conflict in the hyper-graph is detected.

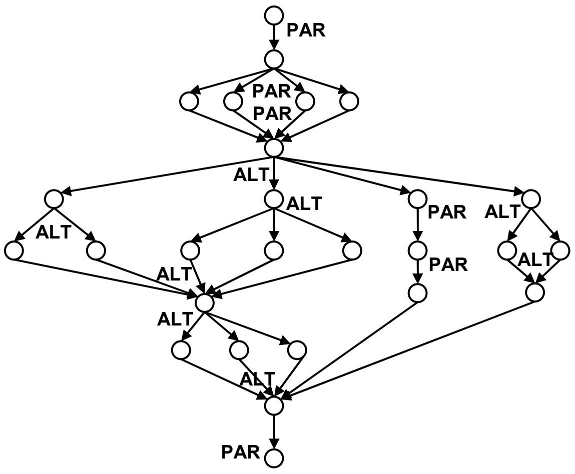


Figure 2. Example of P/A graph with PAR/ALT annotation.

Initial Phase of the Algorithm

Let $G = (V, E)$ be a P/A graph represented as sets of marked fan-in and fan-out sub-graphs (Figure 2). We construct a hyper-graph $H = (V, F)$ over the same set of nodes V in the following way. Let the set of hyper-edges F be empty at the beginning.

For each *fan-in parallel* sub-graph over nodes $x, y_1, y_2, ..., y_k$, where x is the principal node, insert edges (trivial hyper-edges) $\{\{x\},\{y_i\}\}$ for $1 \le i \le k$ into F.

Fan-out sub-graphs marked as *parallel* are treated in the same way as fan-in parallel sub-graphs.

For each *fan-in alternative* sub-graph over nodes $x, y_1, y_2, ..., y_k$, where x is the principal node, insert non-trivial hyper-edge $\{\{x\},\{y_1, y_2, ..., y_k\}\}$ into F.

As in the case of parallel branching, *fan-out* sub-graphs marked as *alternative* are treated in the same way as fan-in alternative sub-graphs.

Informally speaking, we use the same nodes in the hyper-graph as in the P/A graph. For each arc that is a part of parallel branching in the P/A graph we add an edge between the same nodes in the hyper-graph. For a set of arcs that form alternative branching in the P/A graph, we add a non-trivial hyper-edge connecting the same nodes in the hyper-graph (a black dot in Figure 3). We use the convention that an edge with the same structure is added only once (we do not allow multi-edges). Figure 3 shows a hyper-graph constructed for the P/A graph from Figure 2.

The last step of the initial phase is constructing the initial equivalence classes. Let us denote Q_v the equivalence class for $v \in V$. Initially we set $Q_v = \{v\}$ for every $v \in V$. The constructed hyper-graph $H = (V, F)$ with associated equivalence classes is used as the input for the transformation phase of the algorithm.

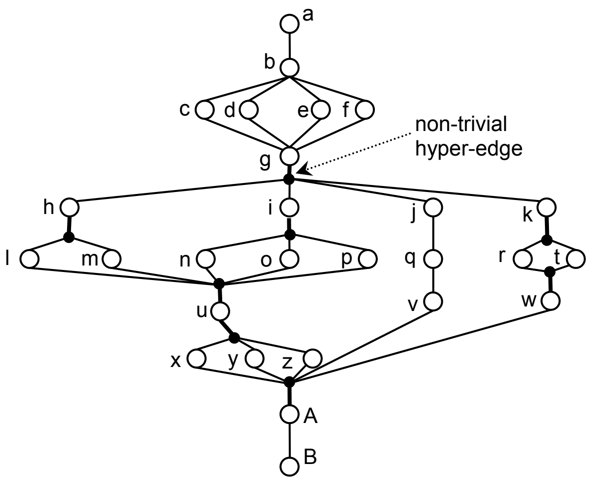


Figure 3. Hyper-graph corresponding to the P/A graph.

Transformation Phase of the Algorithm

The goal of the transformation phase is to modify the hyper-graph while preserving the equivalence classes. This is realized by three transformation rules that update the hyper-graph by adding derived hyper-arcs. During these updates, the initial equivalence classes are being joined

Edge contraction rule. The first transformation rule contracts an edge. An edge $\{\{u\},\{v\}\} \in F$ can be contracted if for any non-trivial hyper-edge $\{X,\{y\}\} \in F$ $\left|\{u,v\} \cap (X \cup \{y\})\right| \le 1$. The case when an edge cannot be contracted is treated separately. The edge contraction rule represents standard operation from graph theory.

Let $\{\{u\},\{v\}\} \in F$ be the edge that can be contracted. Then the following steps are carried out. Erase the vertex v by assigning: $V \leftarrow V - \{v\}$ and $Q_u \leftarrow Q_u \cup Q_v$. Edges and hyper-edges which have v as an endpoint need to be modified to form a correct hyper-graph without v. If there is an edge $\{\{x\},\{v\}\} \in F$, where $x \ne u$, replace the edge $\{\{x\},\{v\}\}$ by $\{\{x\},\{u\}\}$. If there is a non-trivial hyper-edge $\{X,\{v\}\} \in F$ then replace the hyper-edge $\{X,\{v\}\}$ by $\{X,\{u\}\}$. If there is a non-trivial hyper-edge $\{X,\{y\}\} \in F$, where $u \notin X$, $v \in X$ and $y \ne u$ then replace it by hyper-edge $\{(X - \{v\}) \cup \{u\},\{y\}\}$.

Hyper-edge extension rule. If there are non-trivial hyper-edges $\{\{x\},Y\} \in F$ and $\{\{y\},Z\} \in F$, where $y \in Y$ and $(\{x\} \cup Y) \cap (\{y\} \cup Z) = \{y\}$ then add a new hyper-edge $\{\{x\},Y \cup Z - \{y\}\}$ into F. It may happen that the new edge generated by this rule is already present in the hyper-graph from another reason. Then the rule does not change the hyper-graph.

Hyper-edge meet rule. If there are non-trivial hyper-edges $\{\{x\},Y\} \in F$ and $\{Z,\{w\}\} \in F$, where $x \ne w$ and $Z \subseteq Y$ then add a new hyper-edge $\{\{x\},(Y - Z) \cup \{w\}\}$ into F. Again the rule has no effect if the edge generated by this rule is already present in the hyper-graph. A special case of this rule when $Z = Y$ results in addition of a new edge. The edge generated in this case allows further contractions subsequently.

This transformation rule in cooperation with the previous rule can discover the situation when a production chain splits into several alternatives and all these alternatives join again.

Conflict detection rule. The case when an edge that cannot be contracted occurs in the hyper-graph indicates a conflict. This conflict means that some validity variables are forced to be 0. The current version of the algorithm is not able to handle this situation so this rule terminates the algorithm and reports a conflict to the user.

If any of the above-defined transformation rules cannot be applied, the algorithm terminates with success. After successful termination the node equivalence classes associated with nodes remaining in the final hyper-graph represent sets of equivalent nodes. Figures 4 and 5 illustrate several steps of the transformation phase. The algorithm finishes with the following equivalence classes $\{a,b,c,d,e,f,g,A,B\}$, $\{j,q,v\}$, and $\{k,w\}$.

Figure 4 shows a hyper-graph that we obtained from the hyper-graph in Figure 3 by repeatedly applying the edge contraction rule. Namely, we contracted all edges between nodes *a* and *g* so we obtained a single node that describes equivalence class $\{a,b,c,d,e,f,g\}$. We also contracted edges $\{\{j\},\{q\}\}$ and $\{\{q\},\{v\}\}$ and got a node with equivalence class $\{j,q,v\}$. Finally, we contracted edge $\{\{A\},\{B\}\}$ to obtain a node with equivalence class $\{A,B\}$.

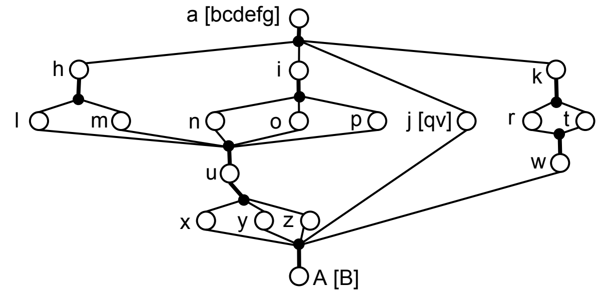


Figure 4. Hyper-graph after edge contractions.

The hyper-graph in Figure 5 was obtained by applying the hyper-edge meet rule to hyper edges $\{\{k\},\{r,t\}\}$ and $\{\{w\},\{r,t\}\}$ that lead to adding a new edge $\{\{k\},\{w\}\}$. This edge was then contracted to find a new equivalence class $\{k,w\}$. The new bottom hyper-edge $\{\{A\},\{u,j,k\}\}$ was then obtained by applying the hyper-edge meet rule using hyper-edge $\{\{u\},\{x,y,z\}\}$. The new top hyper-edge was obtained by applying hyper-edge extension rule.

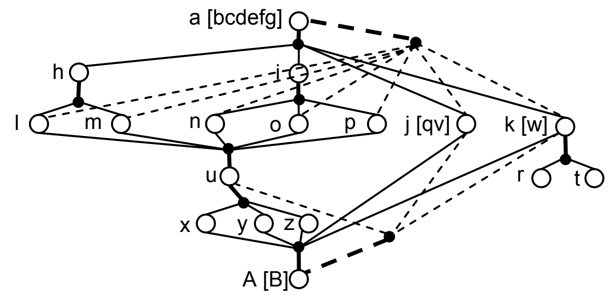


Figure 5. Hyper-graph after extension and meet rule application.

If we apply further the hyper-edge extension rule to the bottom hyper-edge we will obtain two hyper-edges that meet which will later lead to joining equivalence classes $\{a,b,c,d,e,f,g\}$ and $\{A,B\}$ (not shown).

Correctness of the algorithm

We shall show now that the algorithm is correct, that is, it returns groups of equivalent nodes. We can easily define feasible assignment of 0/1 values to nodes of the hyper-graph. This definition should ensure that an assignment of 0/1 values to nodes of the original P/A graph is feasible if and only if it is feasible for the corresponding hyper-graph (the correspondence between P/A graph and hyper-graph is determined by the initial phase of the algorithm).

Let us denote $Q = \bigcup_{v \in V} Q_v$. An assignment $val : Q \to \{0,1\}$ in hyper-graph $H = (V,F)$ is feasible if and only if for every edge $\{\{u\},\{v\}\} \in F$ $val(u) = val(v)$ and for every non-trivial hyper-edge $\{U,\{v\}\} \in F$ $\sum_{u \in U} val(u) = val(v)$. Nodes in the same equivalence class associated with a given node have assigned the same value.

To prove the correctness of the algorithm it is sufficient to show that hyper-graph transformation rules preserve feasible assignments.

Proposition 2 (correctness of edge contraction). An assignment $val : Q \rightarrow \{0,1\}$ in hyper-graph $H = (V, F)$ is feasible if and only if it is feasible for hyper-graph after application of the *edge contraction rule*.

Proof. Let $\{\{u\},\{v\}\} \in F$ be the contracted edge. For $\{\{x\},\{v\}\} \in F$, where $x \neq u$ feasibility for hyper-graph H forces $val(x) = val(v)$ and $val(u) = val(v)$. Feasibility for hyper-graph after edge contraction forces $val(x) = val(u)$ since $\{\{x\},\{v\}\}$ is replaced by $\{\{x\},\{u\}\}$ and $val(u) = val(v)$ since $v \in Q_u$. These two conditions are clearly equivalent.

For hyper-edge $\{X,\{v\}\} \in F$ feasibility for H forces $\sum_{x \in X} val(x) = val(v)$ and $val(u) = val(v)$. Feasibility for hyper-graph after edge contraction forces $\sum_{x \in X} val(x) = val(u)$ since $\{X,\{v\}\}$ is replaced by $\{X,\{u\}\}$ and $val(u) = val(v)$ since $v \in Q_u$. We again obtained equivalence of both conditions.

Finally for hyper-edge $\{X,\{y\}\} \in F$, where $u \notin X$, $v \in X$ and $y \neq u$, feasibility for H forces $\sum_{x \in X} val(x) = val(y)$ and $val(u) = val(v)$. Feasibility for hyper-graph after edge contraction forces $\sum_{x \in X} val(x) - val(v) + val(u) = val(y)$ and $val(u) = val(v)$ since $v \in Q_u$. These conditions are again equivalent.

Q.E.D.

Proposition 3 (correctness of hyper-edge extension). An assignment $val : Q \rightarrow \{0,1\}$ in hyper-graph $H = (V, F)$ is feasible if and only if it is feasible for hyper-graph after application of the *hyper-edge extension rule*.

Proof. Let $\{\{x\},Y\} \in F$ and $\{\{y\},Z\} \in F$, where $y \in Y$ and $(\{x\} \cup Y) \cap (\{y\} \cup Z) = \{y\}$ be the non-trivial hyper-edges that are selected to form a new hyper-edge in resulting hyper-graph. Feasibility for hyper-graph H requires $\Sigma_{z\in Y}$ $val(z)$ = $val(x)$, $\Sigma_{z\in Z}$ $val(z)$ = $val(y)$, and $\Sigma_{z\in Y}$ $val(z)$ + $\Sigma_{z\in Z}$ $val(z)$ – $val(y)$ = $val(x)$. It is exactly the condition forced by the feasibility in the resulting hyper-graph.

Q.E.D.

Proposition 4 (correctness of hyper-edge meet). An assignment $val : Q \rightarrow \{0,1\}$ in hyper-graph $H = (V, F)$ is feasible if and only if it is feasible for hyper-graph after application of the *hyper-edge meet rule*.

Proof. Let $\{\{x\},Y\} \in F$ and $\{Z,\{w\}\} \in F$, where $x \neq z$ and $Z \subseteq Y$ be the non-trivial hyper-edges on which the rule is applied. Feasibility for hyper-graph H requires that $\sum_{y \in Y} val(y) = val(x)$ and $\sum_{z \in Z} val(z) = val(w)$ which is equivalent with the requirement $\sum_{y \in Y} val(y) - \sum_{z \in Z} val(z) + val(w) = val(x)$.

Q.E.D.

Temporal Networks with Alternatives

So far we assumed acyclic graphs describing precedence relations between nodes and we focused on the logical aspects of the network, namely selecting the nodes to satisfy parallel and alternative branching. Nevertheless, in real-life problems we usually need a finer time resolution so we can extend precedence relations to simple temporal relations. It means that each arc (X,Y) in a P/A graph is annotated by a pair of numbers $[a,b]$ where a describes the minimal distance between nodes X and Y and b describes the maximal distance, formally, $a \leq$ Y-X $\leq b$. We call the resulting graph a *P/A simple temporal network*. Now the problem is to decide validity of nodes satisfying parallel and alternative branching and to assign time (number) to each valid node in such a way that all simple temporal relations between valid nodes are satisfied. We call the problem of deciding whether a feasible assignment of validity and time variables exists a *P/A simple temporal network assignment problem*. Again, we assume that validity of some nodes is set to 1 (otherwise, there is a trivial solution where all nodes are invalid). This is a typical situation when the proposed temporal network is used to model real-life problems. The last nodes in the structure of alternative process routes typically describe delivery to a customer. Because the delivery must be fulfilled and we can just select alternative ways how to do it, these nodes must be valid.

Recall, that there exist polynomial algorithms for checking consistency of simple temporal networks (Dechter, Meiri, and Pearl 1991) so solving simple temporal problems is "easy". However, as we showed above adding parallel and alternative branching makes the problem hard.

Proposition 5: The P/A simple temporal network assignment problem is NP-complete.

Proof: The problem is obviously in NP, because it suffices to guess the assignment and test its feasibility, which can be done in linear time in the number of arcs. The P/A simple temporal network is a generalisation of P/A graph in the following sense. For any P/A graph we can construct a P/A simple temporal network where all temporal constraints are in the form $[0,\infty]$. Now, there exists a feasible assignment to the P/A graph if and only if there exists a feasible assignment to the corresponding P/A simple temporal network. Moreover, if we assume that all time variables are set to 0, which trivially satisfies all temporal constraints, we get one-to-one mapping between assignments. Hence, the P/A simple temporal network assignment problem is NP-complete.

Q.E.D.

It may seem that we can further generalise the framework by using a disjunction of simple temporal relations. Formally, each arc (X,Y) is annotated by a set of number pairs $[a_i,b_i]$ for i=1,..,n with the following meaning $\vee_{i=1,\dots,n}\, a_i \leq$ Y-X $\leq b_i$. Nevertheless, this generalisation does not increase the expressive power of the framework because we already have alternatives there. In fact, arc (X,Y) with a disjunctive constraint $\vee_{i=1,...,n}\ a_i \le$ Y-X $\le b_i$ can be substituted by a sub-network with simple temporal constraints as Figure 6 shows. Note that auxiliary nodes x' and y' are necessary to keep fan-out subgraph with principal node X or fan-in subgraph with principal node Y (if such subgraphs exists) "isolated" from the newly added fan-out and fan-in subgraphs (otherwise we may have more than one fan-out or fan-in subgraph branching out of or in the same node, namely X or Y). Nodes x' and y' are equivalent in the sense described in the previous section and the algorithm presented there can detect this equivalence.

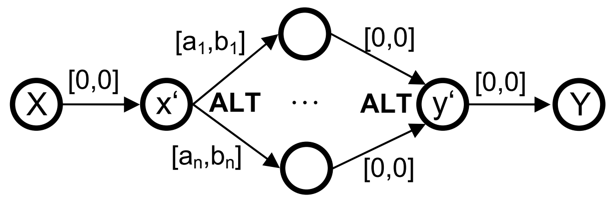


Figure 6. Modelling disjunctions of temporal constraints.

Because our framework covers SAT problems, we believe that it should be possible to model arbitrary temporal disjunctions as specified in disjunctive temporal networks (Stergiou and Koubarakis 1998). Nevertheless, we did no formal analysis in this direction yet because our application area (manufacturing scheduling) assumes only specific temporal disjunctions describing disjunctive resources. To handle these disjunctions, we plan to use existing constraints for disjunctive resources. For example the paper (Barták 2006) presents such a constraint that can handle activities with the validity status.

Example

So far we have discussed mainly the theoretical background of our proposal, so let us now present an example showing how the proposed framework can model a real-world problem. Consider the manufacturing of pistons where each piston consists of a rod and a tube that need to be assembled together to form the piston. Each rod consists of the main body and a special kit that is welded to the rod (the kit needs to be collected from warehouse and then assembled). The rod body is sawn from a large metal stick. The tube can also be sawn from a larger tube. Both rod body and tube must be collected together from the warehouse to ensure that their diameters fit. If the tube is not available, it can be bought from an external supplier. In any case some welding is necessary to be done on the tube before it can be assembled with the rod. Finally, between sawing and welding, both rod and tube must be cleared of metal cuts produced by sawing. Assume that welding and sawing operations require ten time units, assembly operation requires five time units, clearing can be done in two time units, and the material is collected from warehouse in one time unit. If the tube is bought from external supplier then it takes fifty time units to get it. Moreover, tube and rod must cool-down after welding which takes five time units.

The above problem could be easily modelled using a simple temporal network if there is no alternative whether to produce the tube in-house or buy it. Our proposal is focused exactly on this type of problems where (exclusive) alternatives must be modelled. Figure 7 shows a P/A simple temporal network modelling the problem. Each operation is modelled using a single node in the network indicating the start time of the operation. Note that neither disjunctive temporal networks nor other existing extensions of temporal networks can model this problem.

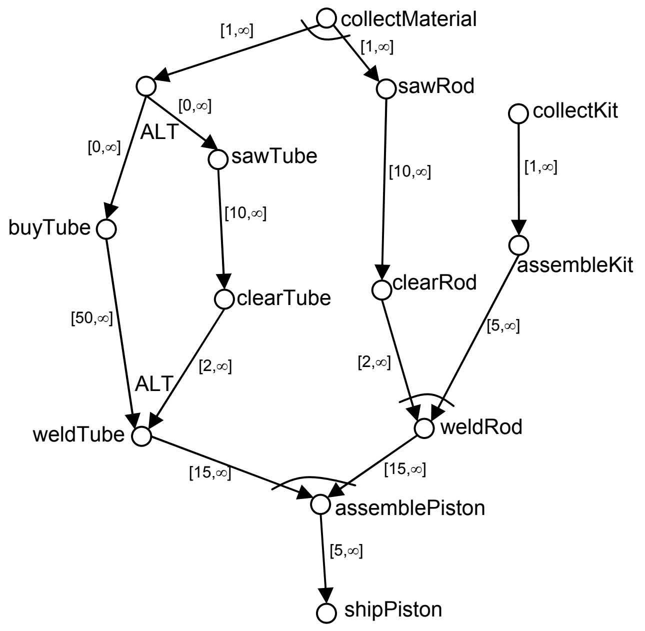


Figure 7. Example of manufacturing process with alternatives.

Related Works

The intended application area for the proposed framework is manufacturing scheduling. There exists a benchmark set MaScLib by ILOG (Nuijten at el. 2003) which contains a formal description of real-life manufacturing scheduling problems. This description includes the concept of validity variables and logical dependencies between them. Temporal and logical relations are modelled separately there and various binary logical relations can be defined between the validity variables. Our framework defines the logical dependencies via branching in the temporal graph. According to our experience this is satisfactory for modelling manufacturing (and other) processes. Moreover, we believe that the coupled definition will lead to more efficient filtering algorithms that use together temporal and logical information. In (Barták and Čepek, 2006) we already showed that integrated filtering of precedence and dependency constraints significantly reduces solving time.

We are not aware about another approach that can handle alternative process routes in the same generality as the proposed P/A simple temporal networks. The paper (Focacci, Laborie, and Nuijten 2000) describes a graph concept for modelling alternative processes, but it cannot be used for alternative routes because all activities must be present. Probably the closest approach to our proposal is the work by Beck and Fox (1999) on modelling alternative processes using PEX (probability of existence) variables. In our framework we focus on logical validity variables (PEX uses an interval of real numbers ⟨0,1⟩) but the main ideas of propagation are very similar. Using validity variables instead of PEX values simplifies integration to existing constraint solvers and we believe that using logical deduction during pre-processing can generate additional input to the filtering algorithm.

Our work is naturally related to temporal networks as we proposed an extension of simple temporal networks. We already showed that the proposed framework covers Temporal Constraint Satisfaction Problems (Dechter, Meiri, and Pearl 1991). Disjunctive Temporal Network (Stergiou and Koubarakis 1998) is another approach to handling temporal alternatives. We have no formal comparison to our P/A simple temporal network yet, but our ambition is slightly different from DTN – we model alternative routes rather than any temporal disjunction.

Recently several other extensions of temporal networks appeared like resource temporal networks (Laborie 2003) or disjunctive temporal networks with finite domain constraints (Moffitt, Peintner, and Pollack 2005). These extensions integrate temporal reasoning with reasoning on non-temporal information, like fluent resources. Our ambition is to extend existing constraint-based scheduling by some planning decisions, namely selection of alternative processes. So we extended temporal reasoning by logical reasoning on existence of nodes in the network. Actually, the possibility to decide about validity/invalidity of the node is the main difference of our approach from the above mentioned works on temporal networks where all nodes must always be present.

There exists Conditional Temporal Planning (Tsamardinos, Vidal, and Pollack 2003) where existence of node in the network depends on some condition. Though there is some similarity in modelling alternative processes/plans, satisfaction of condition in CTP depends on external forces – Nature – rather than being an internal relation between the nodes. In our approach, decision of validity of the node is done internally based on logical relations between the nodes.

Conclusions

The paper reports a work in progress on extension of simple temporal networks for handling alternative process routes. We focused on formalization of the this new modelling framework, showing its complexity, and proposing a pre-processing step for extracting information about logically equivalent nodes from the network. We are currently working on filtering algorithms for removing inconsistencies from the network based on ideas of constructive disjunction. We believe that the pre-processing phase can feed the filtering algorithm by useful information about implied logical relations (such as equivalence classes and dependencies) that can be used to improve the filtering power.

Acknowledgements

The research is supported by the Czech Science Foundation under the contract no. 201/04/1102 and by the EMPOSME project under EU FP6 scheme.

References

Barták, R. 2006. Incremental Propagation of Time Windows on Disjunctive Resources. In *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, AAAI Press, pp. 25-30.

Barták, R.; Čepek, O. 2006. Incremental Filtering Algorithms for Precedence and Dependency Constraints. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence* (ICTAI 2006). IEEE Press (to appear).

Beck, J.Ch. and Fox, M.S. 1999. Scheduling Alternative Activities. *Proceedings of AAAI-99*, USA, pp. 680-687.

Dechter, R.; Meiri, I. and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence*, 49:61.95.

Focacci, F.; Laborie, P.; and Nuijten, W. 2000. Solving Scheduling Problems with Setup Times and Alternative Resources. In *Proceedings of AIPS 2000*.

Garey, M. R. and Johnson, D. S. 1979 *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco.

Laborie, P. 2003. Resource temporal networks: Definition and complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 948-953.

Moffitt, M. D.; Peintner, B.; and Pollack, M. E. 2005. Augmenting Disjunctive Temporal Problems with Finite-Domain Constraints. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pp. 1187-1192. AAAI Press.

Nuijten, W.; Bousonville, T.; Focacci, F.; Godard, D.; Le Pape, C. 2003. MaScLib: Problem description and test bed design, http://www2.ilog.com/masclib

Stergiou, K., and Koubarakis, M. 1998. Backtracking algorithms for disjunctions of temporal constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pp. 248-253. AAAI Press.

Tsamardinos, I.; Vidal, T. and Pollack, M.E. 2003. CTP: A New Constraint-Based Formalism for Conditional Temporal Planning. *Constraints*, 8(4):365.388.