Temporal Reasoning in Nested Temporal Networks with Alternatives

Roman Barták, Ondřej Čepek, Martin Hejna

Charles University in Prague, Faculty of Mathematics and Physics,
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
roman.bartak@mff.cuni.cz, ondrej.cepek@mff.cuni.cz,
mhejna@matfyz.cz

Abstract. Temporal networks play a crucial role in modeling temporal relations in planning and scheduling applications. Temporal Networks with Alternatives (TNAs) were proposed to model alternative and parallel processes in production scheduling, however the problem of deciding which nodes can be consistently included in such networks is NP-complete. A tractable subclass, called Nested TNAs, can still cover a wide range of real-life processes, while the problem of deciding node validity is solvable in polynomial time. In this paper, we show that adding simple temporal constraints (instead of precedence relations) to Nested TNAs makes the problem NP-hard again. We also present several complete and incomplete techniques for temporal reasoning in Nested TNAs.

1 Introduction

Planning and scheduling applications almost always include some form of temporal reasoning, for example, a causal relation (the effect of some activity is required for processing another activity) implies a precedence constraint. These relations are frequently modeled using temporal networks where nodes correspond to activities and arcs are annotated by the temporal relations between activities. Current temporal networks handle well temporal information including disjunction of temporal constraints [13] or uncertainty [4]. Several other extensions of temporal networks appeared recently such as resource temporal networks [10] or disjunctive temporal networks with finite domain constraints [11]. These extensions integrate temporal reasoning with reasoning on non-temporal information, such as fluent resources (for example fuel consumption during car driving). All these approaches assume that all nodes are present in the network, though the position of nodes in time may be influenced by other than temporal constraints. Conditional Temporal Planning [14] introduced an option to decide which node will be present in the solution depending on a certain external condition. Hence CTP can model conditional plans where the nodes actually present in the solution are selected based on external forces. Temporal Plan Networks [8] (TPN) also include conditional branching and they attempt to model all alternative plans in a single graph. Temporal Networks with Alternatives [1] (TNA) introduced a

© Václav Snášel (Ed.): Znalosti 2008, pp. 42–53, ISBN 978-80-227-2827-0.
FIIT STU Bratislava, Ústav informatiky a softvérového inžinierstva, 2008.

different type of alternatives with so called parallel and alternative branching. They are more general than TPN but the problem of deciding which nodes can be consistency included in the network, if some nodes are pre-selected, is NP-complete even if no temporal constraints are imposed. Therefore a restricted form, so called Nested TNAs, was proposed in [2]. Nested TNAs have a similar topology as TPNs though the original motivation for their introduction was different – a Nested TNA focuses on manufacturing processes while a TPN models plans for unmanned vehicles. The paper [2] shows that the problem of deciding whether a subset of nodes can be selected to satisfy the branching constraints is now tractable, but it still leaves open the question what happens if temporal constraints are assumed. In this paper we present a new complexity result for Nested Temporal Networks with Alternatives where simple temporal constraints are included. We also present some new algorithms that can help in solving problems based on (Nested) TNAs. These algorithms exploit the integrated reasoning on both logical (branching) and temporal constraints.

There exist other frameworks mixing temporal and logical reasoning. In problems, such as log-based reconciliation [7], we need to model inter-dependencies between nodes which concern their presence/absence in the final solution. The possibility to select nodes according to logical, temporal, and resource constrains was introduced to manufacturing scheduling by ILOG in their MaScLib [12]. The same idea was independently formalized in Extended Resource Constrained Project Scheduling Problem [9]. In the common model each node has a Boolean validity variable indicating whether the node is selected to be in the solution. These variables are a discrete version of PEX variables used by Beck and Fox [3] for modeling presence of alternative activities in the schedule. In many recent approaches, these variables are interconnected by logical constraints such as the dependency constraint described above.

In this paper, we first give motivation for using Temporal Networks with Alternatives and formally introduce TNAs and their nested form. The main part of the paper shows that using temporal constraints in Nested TNAs makes the problem of deciding which nodes can be consistency included in the network NP-complete again. We also present several techniques that can help in solving the problem. These techniques are proposed in the context of constraint satisfaction so they can be easily integrated with other constraints, for example with constraints that model resources. Hence the proposed techniques are useful for solving oversubscribed real-life scheduling problems.

2 Motivation and Background

Let us consider a manufacturing scheduling problem of piston production. 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 assembled before welding). The rod body is sawn from a large metal stick. The tube can also be sawn from a larger tube. Rod body, the kit, 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 pro-

duced 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 an 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.

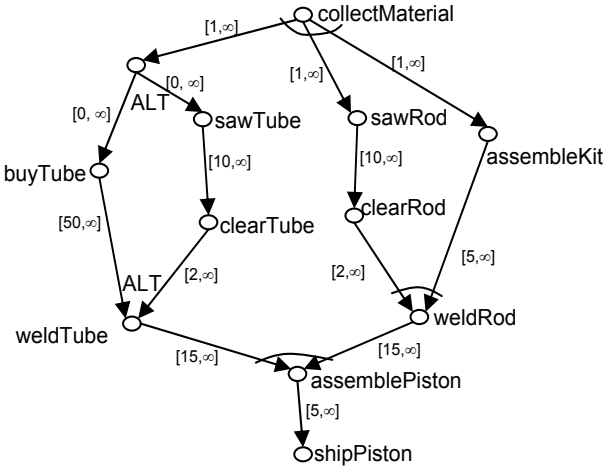

Fig. 1. Example of a manufacturing process with alternatives.

The manufacturing processes from the above problem can be described using a Temporal Network with Alternatives depicted in Figure 1. Nodes correspond to start times of operations and arcs are annotated by simple temporal constraints in the form $[a, b]$, where a describes the minimal distance (in time) between the nodes and b describes the maximal distance. Informally, this network describes the traditional simple temporal constraints [5] together with the specification of branching of processes. There is a *parallel branching* marked by a semi-circle indicating that the process splits and runs in parallel and an *alternative branching* marked by ALT indicating that the process will consists of exactly one alternative path (we can choose between buying a tube and producing it in situ).

3 Temporal Networks with Alternatives

Let us now formally define Temporal Networks with Alternatives from [1]. Let G be a directed acyclic graph. A sub-graph of G is called a *fan-out sub-graph* if it consists of nodes $x, y_1,\ldots, y_k$ (for some k) such that each (x, y_i), $1 \le i \le k$, is an arc in G. If $y_1,\ldots, y_k$ are all and the only successors of x in G (there is no z such that (x, z) is an arc in G and $\forall i = 1,\ldots,k$: $z \ne y_i$) then we call the fan-out sub-graph complete. Similarly, a sub-graph of G is called a *fan-in sub-graph* if it consists of nodes $x, y_1,\ldots, y_k$ (for some k) such that each (y_i, x), $1 \le i \le k$, is an arc in G. A complete fan-in sub-

graph is defined similarly as above. In both cases x is called a *principal node* and all $y_1,\ldots, y_k$ are called *branching nodes*.

Definition 1: A directed acyclic graph G together with its pair wise edge-disjoint decomposition into complete fan-out and fan-in sub-graphs, where each sub-graph in the decomposition is marked either as a *parallel* sub-graph or an *alternative* sub-graph, is called a *P/A graph*.

Definition 2: *Temporal Network with Alternatives* is a P/A graph where each arc (x, y) is annotated by a pair of numbers $[a,b]$ (a *temporal annotation*) where a describes the minimal distance between x and y and b describes the maximal distance, formally, $a \leq t_y - t_x \leq b$, where t_x denotes the position of node x in time.

Figure 1 shows an example of Temporal Network with Alternatives. If we remove the temporal constraints from this network then we get a P/A graph. Note that the arcs (*sawTube*, *clearTube*), (*sawRode*, *clearRod*), and (*assemblePiston*, *shipPiston*) form simple fan-in (or fan-out, it does not matter in this case) sub-graphs. As we will see later, it does not matter whether the sub-graphs consisting of a single arc are marked as parallel or alternative – the logical constraint imposed by the sub-graph will be always the same. Hence, we can omit the explicit marking of such single-arc sub-graphs to make the figure less overcrowded.

We call the special logical relations imposed by the fan-in and fan-out sub-graphs *branching constraints*. Temporarily, we omit the temporal constraints, so we will work with P/A graphs only, but we will return to temporal constraints later in the paper. In particular, we are interested in finding whether it is possible to select a subset of nodes in such a way that they form a feasible graph according to the branching constraints. Formally, the selection of nodes can be described by an *assignment* of 0/1 values to nodes of a given P/A graph, where value 1 means that the node is selected and value 0 means that the node is not selected. The assignment is called *feasible* if

- in every parallel sub-graph all nodes are assigned the same value (both the principal node and all branching nodes are either all 0 or all 1),
- in every alternative sub-graph 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.

Notice that the feasible assignment naturally describes one of the alternative processes in the P/A graph. For example, *weldRod* is present if and only if both *clearRod* and *assembleKit* are present (Figure 1). Similarly, *weldTube* is present if exactly one of nodes *buyTube* or *clearTube* is present (but not both). Though, the alternative branching is quite common in manufacturing scheduling, it cannot be described by binary logical constraints from MaScLib [12] or Extended Resource Constrained Project Scheduling Problem [9]. On the other hand, the branching constraints are specific logical relations that cannot capture all logical relations between the nodes.

It can be easily noticed that given an arbitrary P/A graph the assignment of value 0 to all nodes is always feasible. On the other hand, if some of the nodes are required to take value 1, then the existence of a feasible assignment is by no means obvious. Let us now formulate this decision problem formally.

Definition 3: Given a P/A graph G and a subset of nodes in G which are assigned to 1, *P/A graph assignment problem* is "Is there a feasible assignment of 0/1 values to all nodes of G which extends the prescribed partial assignment?"

Intuition motivated by real-life examples says that it should not be complicated to select the nodes to form a valid process according to the branching constraints described above. The following proposition from [1] says the opposite.

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

Nevertheless, if we look back to the motivation example (Figure 1), we can see that the TNA has a specific topology which is, according to our experience, very typical for real-life processes. First, the process has usually one start point and one end point. Second, the graph is built by decomposing meta-processes into more specific processes until non-decomposable processes (operations) are obtained. There are basically two (three) types of decomposition. The meta-process can split into two or more processes that run in a sequence, that is, after one process is finished, the subsequent process can start. The meta-process can split into two or more sub-processes that run in parallel, that is, all sub-processes start at the same time and the meta-process is finished when all sub-processes are finished. Finally, the meta-process may consists of several alternative sub-processes, that is, exactly one of these sub-processes is selected to do the job of the meta-process. Notice, that the last two decompositions have the same topology of the network, they only differ in the meaning of the branches in the network. Note finally, that we are focusing on modeling instances of processes with particular operations that will be allocated to time. Hence we do not assume loops that are sometimes used to model abstract processes. Figure 2 shows how the network from Figure 1 is constructed from a single arc by applying the above mentioned decomposition steps.

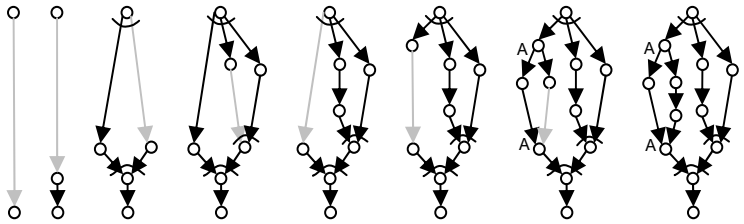

Fig. 2. Building a labeled nested graph.

We will now formally describe this concept that we called nesting. The resulting network is called a Nested Temporal Network with Alternatives [2].

Definition 4: A directed graph G = ($\{s,e\}$, $\{(s,e)\}$) is a *(base) nested graph*. Let G = (V, E) be a graph, $(x,y) \in E$ be its arc, and $z_1,\ldots, z_k$ ($k > 0$) be nodes such that neither z_i is in V. If G is a nested graph (and I = $\{1,\ldots,k\}$) then graph G' = ($V \cup \{z_i \mid i \in I\}$, $E \cup \{(x,z_i), (z_i,y) \mid i \in I\} - \{(x,y)\}$) is also a *nested graph*.

According to Definition 4, any nested graph can be obtained from the base graph with a single arc by repeated substitution of any arc (x,y) by a special sub-graph with

k nodes (see Figure 3). Notice that a single decomposition rule covers both the serial process decomposition ($k = 1$) and the parallel/alternative process decomposition ($k > 1$). Though this definition is constructive rather than fully declarative, it is practically very useful. Namely, interactive process editors can be based on this definition so the users can construct only valid nested graphs by decomposing the base nested graph.

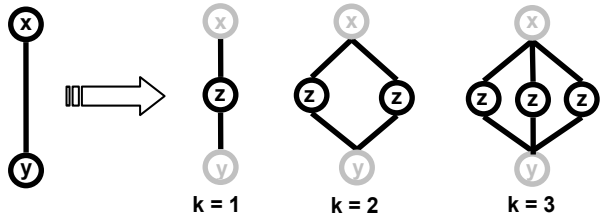

Fig. 3. Arc decomposition in nested graphs.

The directed nested graph defines topology of the nested P/A graph but we also need to annotate all fan-in and fan-out sub-graphs as either alternative or parallel sub-graphs. Moreover, we need to do the annotation carefully so the assignment problem can be solved easily for nested graphs and no node is inherently invalid. The idea is to annotate each node by input and output label which defines the type of branching (fan-in or fan-out sub-graph).

Definition 5: *Labeled nested graph* is a nested graph where each node has (possibly empty) input and output labels defined in the following way. Nodes *s* and *e* in the base nested graph and nodes z_i introduced during decomposition have empty initial labels. Let *k* be the parameter of decomposition when decomposing arc (*x*,*y*). If $k > 1$ then the output label of *x* and the input label of *y* are unified and set either to PAR or to ALT (if one of the labels is non-empty then this label is used for both nodes).

Figure 2 demonstrates how the labeled nested graph is constructed for the motivation example from Figure 1. In particular, notice how the labels of nodes are introduced (a semicircle for PAR label and A for ALT label). When a label is introduced for a node, it never changes in the generation process. If an arc (x, y) is being decomposed into a sub-graph with *k* new nodes where $k > 1$, then we require that the output label of *x* is unified with the input label of *y*. This can be done only if either both labels are identical or at least one of the labels is empty. It is easy to show that the second case always holds [2]. Now, we can formally introduce a nested P/A graph.

Definition 6: *A nested P/A graph* is obtained from a labeled nested graph by removing the labels and defining the fan-in and fan-out sub-graphs in the following way. If the input label of node *x* is non-empty then all arcs (y, x) form a fan-in sub-graph which is parallel for label PAR or alternative for label ALT. Similarly, nodes with a non-empty output label define fan-out sub-graphs. Each arc (x, y) such that both output label of *x* and input label of *y* are empty forms a parallel fan-in sub-graph.

Note, that requesting a single arc to form a parallel fan-in sub-graph is a bit artificial. We use this requirement to formally ensure that each arc is a part of some sub-graph which is required to show that a nested P/A graph is a P/A graph [2]. What is more interesting is that for Nested P/A Graphs the following proposition holds.

Proposition 2: The assignment problem for a nested P/A graph is tractable (can be solved in a polynomial time).

The formal proof in [2] is based on constructing a constraint model for nested P/A graphs where local (namely, arc) consistency, which is achievable in polynomial time, implies global consistency. If global consistency is achieved then the solution can be found using a backtrack-free depth-first search (provided that the problem is globally consistent, otherwise no solution exists). This constraint model is basically a (Berge acyclic) reformulation of the following straightforward model for the P/A graph assignment problem. Each node x is represented using a Boolean validity variable v_x, that is a variable with domain $\{0,1\}$. If the arc between nodes x and y is a part of some parallel sub-graph then we define the following constraint:

$$v_x = v_y. \tag{1}$$

If x is a principal node and $y_1,\ldots, y_k$ for some k are all branching nodes in some alternative sub-graph then the logical relation defining the alternative branching can be described using the following arithmetic constraint:

$$v_x = \Sigma_{j=1,\ldots,k}\, v_{y_j}. \tag{2}$$

Notice that if $k = 1$ then the constraints for parallel and alternative branching are identical (hence, it is not necessary to distinguish between them). Notice also that the arithmetic constraint for alternative branching together with the use of $\{0,1\}$ domains defines exactly the logical relation between the nodes – v_x is assigned to 1 if and only if exactly one of v_{y_j} is assigned to 1.

4 Temporal Constraints

So far, we focused merely on logical relations imposed by the branching constraints to show that logical reasoning is easy for nested P/A graphs (while it is hard for general P/A graphs). Now we return back the temporal constraints. Notice that the selected feasible set of nodes together with arcs between them forms a sub-graph of the original P/A graph. We require this sub-graph to be also *temporally feasible*, which means that all the temporal constraints between the valid nodes are satisfied in the sense of temporal networks [5]. Naturally, the logical and temporal reasoning is interconnected – if a temporal constraint between nodes x and y cannot be satisfied then (at least) one of the nodes must be invalid (it is assigned to 0). Before we go into technical details notice that if the temporal constraints are in the form of precedence relations or in general only minimal distances are specified between the nodes then temporal feasibility is trivially guaranteed thanks to acyclicity of TNAs (any node can be postponed in time).

Formally, we can extend the above logical constraint model by annotating each node i by temporal variable t_i indicating the position of the node in time. For simplicity reasons we assume that the domain of such variables is an interval $\langle 0, MaxTime\rangle$ of integers, where *MaxTime* is a large enough constant given by the user. Recall that the temporal relation between nodes i and j is described by a pair $[a_{i,j}, b_{i,j}]$. This relation can now be naturally represented using the following constraint:

$$v_i * v_j * (t_i + a_{i,j}) \leq t_j \ \wedge v_i * v_j * (t_j - b_{i,j}) \leq t_i. \quad (3)$$

If $b_{i,j} = \infty$ then the second part of conjunction is omitted and similarly if $a_{i,j} = -\infty$ then the first part of conjunction is omitted. Notice that if any v_i or v_j equals zero (some involved node is invalid) then the constraint is trivially satisfied (we get $0 \leq t_j \wedge 0 \leq t_i$). If both v_i and v_j equal 1 then we get ($t_i + a_{i,j} \leq t_j \ \wedge \ t_j - b_{i,j} \leq t_i$), which is exactly the simple temporal relation between nodes i and j. Figure 4 shows how the domains from the previous example (Figure 1) will look after filtering out the infeasible values by making the above constraint model arc consistent. We assume that *shipPiston* (the bottom node) is a valid node and *MaxTime* = 70. Black nodes are valid; validity of white nodes is not decided yet. Notice weak domain pruning of time variables in the white nodes caused by a disjunctive character of the problem. Actually, the left most path (with *buyTube*) cannot be selected due to time constraints but this is not discovered by making the constraints arc consistent.

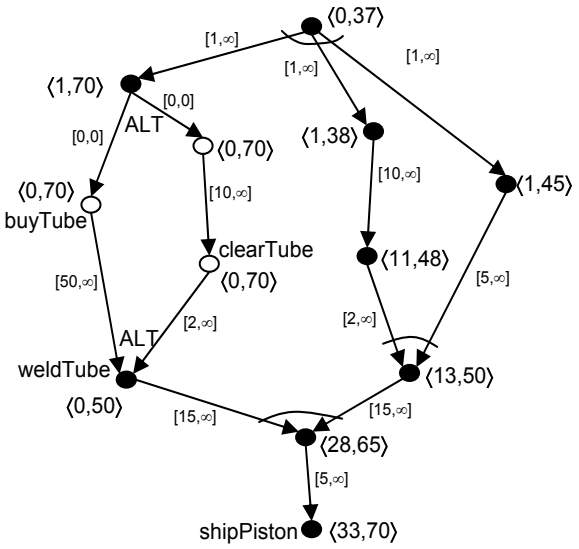

Fig. 4. Domain filtering using the constraint model.

To improve domain filtering we propose to always propagate the temporal constraint even if the validity status of the node is not yet decided. If the temporal constraint is violated then we set some validity variable to 0 (if possible, otherwise a failure is detected). We will describe now the filtering rules that propagate changes of domains between the constrained variables, namely, the values that violate the constraint are removed from the domains. Let $d(x)$ by the domain of variable x, that is, a set of values, and for sets A and B, $A \bullet B = \{a \bullet b \mid a \in A \wedge b \in B\}$ for any binary operation $\bullet$ such as + or –.

Assume that arc (i, j) is a part of a parallel branching, so in the solution either both nodes i and j are valid and the temporal relation must hold, or both nodes are invalid and the temporal relation does not play any role (the domains of temporal variables are irrelevant provided that they are non-empty). Hence, we can always propagate the

temporal relation provided that we properly handle its violation. Let $UP = d(t_j) \cap (d(t_i) + \langle a_{i,j}, b_{i,j}\rangle)$. The following filtering rule is applied whenever $d(t_i)$ changes:

$$\begin{aligned} d(t_j) &\leftarrow UP && \text{if } UP \neq \varnothing \\ d(v_j) &\leftarrow d(v_j) \cap \{0\} && \text{if } UP = \varnothing. \end{aligned} \quad (4)$$

Note that $UP = \varnothing$ means violation of the temporal relation which is accepted only if the nodes are invalid. If the nodes are valid then a failure is generated because the above rule makes the domain of the validity variable empty. Symmetrically, let $DOWN = d(t_i) \cap (d(t_j) - \langle a_{i,j}, b_{i,j}\rangle)$. The following filtering rule is applied whenever $d(t_j)$ changes:

$$\begin{aligned} d(t_i) &\leftarrow DOWN && \text{if } DOWN \neq \varnothing \\ d(v_i) &\leftarrow d(v_i) \cap \{0\} && \text{if } DOWN = \varnothing. \end{aligned} \quad (5)$$

The following example demonstrates the effect of above filtering rules. Assume that the initial domain of temporal variables is $\langle 0, 70\rangle$, the validity of nodes is not yet decided, and there are arcs (i, j) and (j, k) with temporal constraints [10, 30] and [20, 20] respectively. The original constraints do not prune any domain, while our extended filtering rules set the domains of temporal variables t_i, t_j, and t_k to $\langle 0, 40\rangle$, $\langle 10, 50\rangle$, and $\langle 30, 70\rangle$ respectively. If the initial domain is $\langle 0, 20\rangle$ then the original constraints again prune nothing, while our extended filtering rules deduce that the participating nodes are invalid (we assume that logical constraints in the form $v_x = v_y$ are also present).

The propagation of temporal constraints in the alternative branching is more complicated because we do not know which arc is used in the solution. Therefore, the filtering rule uses a union of pruned domains proposed by individual arcs (from non-invalid nodes) which is similar to constructive disjunction of constraints. Let x be the principal node of a fan-in alternative sub-graph and $y_1, \ldots, y_k$ be all branching nodes. We first show how domains of the branching nodes are propagated to the principal node. Let $UP = d(t_x) \cap \cup_{j=1,\ldots,k}\{(d(t_{y_j}) + \langle a_{y_j,x}, b_{y_j,x}\rangle) \mid d(v_{y_j}) \neq \{0\}\}$. The following filtering rule is applied whenever any $d(t_{y_j})$ or $d(v_{y_j})$ changes:

$$\begin{aligned} d(t_x) &\leftarrow UP && \text{if } UP \neq \varnothing \\ d(v_x) &\leftarrow d(v_x) \cap \{0\} && \text{if } UP = \varnothing. \end{aligned} \quad (6)$$

The propagation from $d(x)$ to $d(y_j)$ is done exactly like the *DOWN* propagation described above (rule (5)) and similar filtering rules can be designed for fan-out alternative sub-graphs. Again, the main advantage of these rules is stronger pruning in comparison with the original constraints as we shall show using the example from Figure 4. In particular, if we propagate from *weldTube* to *buyTube* and *clearTube*, we obtain $\langle 0, 0\rangle$ and $\langle 0, 48\rangle$ as new domains of corresponding temporal variables. Now, if we propagate through the other alternative branching going to *buyTube* from top, we deduce that this node is invalid because the corresponding temporal constraint is violated and hence $d(v_{buyTube}) \leftarrow \{0\}$. Consequently, all remaining nodes are valid and we achieved global consistency for both validity and temporal variables. Unfortunately, the proposed filtering rules do not guarantee global consistency in general. Figure 5 shows a nested TNA which is arc consistent, that is, the proposed filtering rules do not remove any inconsistent value from the current domains. However, there does not exist any solution to the problem.

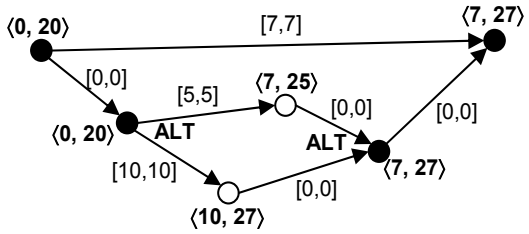

Fig. 5. Locally consistent nested TNA with no solution.

As we shall show below, weak domain filtering of polynomial consistency techniques such as arc consistency is inevitable for (nested) TNAs because the problem of deciding existence of feasible assignment is in fact NP-complete.

Proposition 3: The problem of deciding whether there exists an assignment of times and 0/1 values to all nodes of the (nested) TNA in such a way that all temporal and branching constraints are satisfied 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. For the NP-hardness, we shall show that the subset sum problem, which is known to be NP-complete [6], can be reduced (in a polynomial time) to our assignment problem. The subset sum problem is this: given a set of positive integers Z_i and integer K, does the sum of some subset of $\{Z_i \mid i = 1,\ldots,n\}$ equal to K? We can construct the following nested TNA, where the validity status of the black node is set to 1 and temporal annotation of arcs is [0,0] with the exception of n arcs annotated by $[Z_i, Z_i]$ and one arc annotated by [K,K] (Figure 6). Visibly, the subset sum problem has a solution if and only if there exists a feasible assignment of temporal and validity variables of the constructed nested TNA. The selection of the subset of integers is identical to the choice of alternative branches in the graph. The temporal constraints guarantee that the sum of selected integers equals K (the distance between the leftmost and rightmost node according to the top path). ■

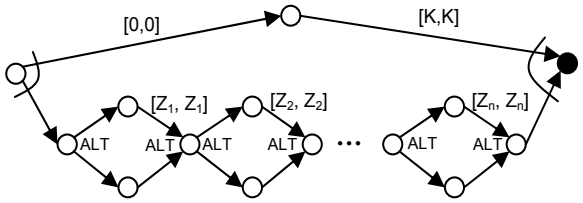

Fig. 6. Subset sum problem formulated as a Nested TNA.

We shall describe now an algorithm which will compute the temporal domains of all vertices in such a way, that every value in every temporal domain is contained in some feasible solution. Let each edge (i, j) in a Nested TNA be labelled by set $S_{ij} \subseteq \langle 0, MaxTime \rangle$ of admissible values for the distance between nodes i and j. Initially, this set corresponds to interval $[a_{i,j}, b_{i,j}]$ specifying the temporal constraint. The pro-

posed algorithm runs in two stages. In the first stage the sequence of decomposition steps used to construct the Nested TNA is followed in the reverse order (this sequence can be found algorithmically in polynomial time for any Nested TNA as shown in [2]). In each composition step in which a parallel or alternative sub-graph with principal vertices x and y and (not invalid) branching vertices $z_1, \ldots, z_k$ (if $k = 1$ then the type of branching is irrelevant) is replaced by a single edge (x, y), the set S_{xy} is computed in the following way:

- $S_{xy} = \cap (S_{xz_i} + S_{z_iy})$ if the replaced sub-graph contains parallel branching,
- $S_{xy} = \cup (S_{xz_i} + S_{z_iy})$ if the replaced sub-graph contains alternative branching.

It is possible to show that the input TNA has a feasible solution if and only if the final base graph with only nodes s and e (into which the input TNA is composed in the end of the first stage) has a feasible solution. If there is no feasible solution, the algorithm terminates. We omit the detailed proof due to a limited number of pages.

In the second stage of the algorithm, we compute restricted temporal constraints T_{ij} and restricted domains of temporal variables t_i containing only globally consistent values starting with the temporal domains in the base graph in the following way:

$d(t_s) \leftarrow \langle 0, MaxTime\rangle \cap (\langle 0, MaxTime\rangle - S_{se})$
$d(t_e) \leftarrow \langle 0, MaxTime\rangle \cap (\langle 0, MaxTime\rangle + S_{se})$
$T_{se} \leftarrow S_{se}$.

After that the base graph is decomposed again into the input graph. During each decomposition step in which a parallel or alternative sub-graph with principal vertices x and y and branching vertices $z_1, \ldots, z_k$ replaces a single edge (x, y), the sets T_{xz_i} and T_{z_iy} and the domains $d(z_i)$ for all $1 \le i \le k$ are computed in the following way:

- $T_{xz_i} = \{u \in S_{xz_i} \mid \exists v \in S_{z_iy} : u + v \in T_{xy}\}$
- $T_{z_iy} = \{v \in S_{z_iy} \mid \exists u \in S_{xz_i} : u + v \in T_{xy}\}$
- $d(t_{z_i}) = \{b \in \langle 0, MaxTime\rangle \mid \exists a \in d(x)\ \exists c \in d(y) : (b - a) \in T_{xz_i} \wedge (c - b) \in T_{z_iy}\}$

If $d(t_{z_i})$ is empty then vertex z_i is invalid so we can set $d(v_{z_i}) \leftarrow \{0\}$ and remove the vertex from the graph. This may happen only for alternative branching due to the way how S_{xy} is computed from S_{xz_i} and S_{z_iy}. Moreover, because S_{xy} is non-empty, at least one node z_i can still be valid so if any node is made invalid this is not propagated elsewhere in the graph. Notice also that $T_{xz_i} \subseteq S_{xz_i}$, $T_{z_iy} \subseteq S_{z_iy}$,and T_{xz_i} and T_{z_iy} contain only those pairs of values which sum up to some value in T_{xy}. Moreover, it is possible to show that only values participating in at least one feasible solution are ever inserted into the temporal domain of any vertex (details omitted due to limited space).

Since we give no implementation details here, it is not possible to determine the exact time complexity of the presented algorithm. However, it should be clear, that any reasonable implementation will work in time polynomial in the size of the input TNA and the upper bound *MaxTime*, thus providing a pseudo-polynomial algorithm with respect to the size of input data (the constant *MaxTime* is part of the input but coded in binary and thus taking *log MaxTime* bits).

5 Conclusions

The paper studies temporal reasoning in Temporal Networks with Alternatives which are useful to model alternative process in production scheduling. We showed that adding simple temporal constraints to Nested TNAs makes the problem of deciding existence of logically and temporally feasible solution NP-complete. We presented a straightforward constraint model and stronger filtering rules that can remove, via arc consistency, some infeasible values from variables' domains, but still cannot guarantee global consistency. We also presented an algorithm for achieving global consistency with pseudo-polynomial time complexity. Note that this algorithm is applicable only to Nested TNAs while the proposed filtering rules work for any TNA.

Acknowledgments. The research is supported by the Czech Science Foundation under the contract no. 201/07/0205.

References

1. R. Barták, O. Čepek, "Temporal Networks with Alternatives: Complexity and Model", Proceedings of the Twentieth International Florida AI Research Society Conference (FLAIRS), AAAI Press, 2007, 641–646.
2. R. Barták, O. Čepek, "Nested Temporal Networks with Alternatives", Papers from the 2007 AAAI Workshop on Spatial and Temporal Reasoning, Technical Report WS-07-12, AAAI Press, 2007, 1-8.
3. J. Ch. Beck, M. S. Fox, "Scheduling Alternative Activities" Proceedings of AAAI-99, AAAI Press, 1999, 680–687.
4. J. Blythe, "An Overview of Planning Under Uncertainty", AI Magazine 20(2), 1999, 37–54.
5. R. Dechter, I. Meiri, J. Pearl, "Temporal Constraint Networks", Artificial Intelligence 49, 1991, 61–95.
6. M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman and Company, San Francisco, 1979.
7. Y. Hamadi, "Cycle-cut decomposition and log-based reconciliation", ICAPS Workshop on Connecting Planning Theory with Practice, 2004, 30–35.
8. P. Kim, B. Williams, M. Abramson, "Executing Reactive, Model-based Programs through Graph-based Temporal Planning", Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2001.
9. J. Kuster, D. Jannach, G. Friedrich, "Handling Alternative Activities in Resource-Constrained Project Scheduling Problems", Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), 2007, 1960–1965.
10. P. Laborie, "Resource temporal networks: Definition and complexity", Proceedings of the 18th International Joint Conference on Artificial Intelligence, 2003, 948–953.
11. M. D. Moffitt, B. Peintner, M. E. Pollack, "Augmenting Disjunctive Temporal Problems with Finite-Domain Constraints" Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005), AAAI Press, 2005, 1187–1192.
12. W. Nuijten, T. Bousonville, F. Focacci, D. Godard, C. Le Pape, "MaScLib: Problem description and test bed design", 2003, http://www2.ilog.com/masclib
13. K. Stergiou, M. Koubarakis, "Backtracking algorithms for disjunctions of temporal constraints", Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), AAAI Press, 1998, 248–253.
14. Tsamardinos, T. Vidal, M. E. Pollack, "CTP: A New Constraint-Based Formalism for Conditional Temporal Planning", Constraints, 8(4), 2003, 365–388.