Nested Precedence Networks with Alternatives: Recognition, Tractability, and Models

Roman Barták*, Ondřej Čepek*†

*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

†Institute of Finance and Administration
Estonská 500, 101 00 Praha 10, Czech Republic
cepek@mail.vsfs.cz

Abstract. Integrated modeling of temporal and logical constraints is important for solving real-life planning and scheduling problems. Logical constrains extend the temporal formalism by reasoning about alternative activities in plans/schedules. Temporal Networks with Alternatives (TNA) were proposed to model alternative and parallel processes, however the problem of deciding which activities can be consistently included in such networks is NP-complete. Therefore a tractable subclass of Temporal Networks with Alternatives was proposed. This paper shows formal properties of these networks where precedence constraints are assumed. Namely, an algorithm that effectively recognizes whether a given network belongs to the proposed sub-class is studied and the proof of tractability is given by proposing a constraint model where global consistency is achieved via arc consistency.

Keywords: temporal networks, alternatives, constraint models, complexity.

1 Introduction

Temporal networks focus on modeling temporal relations in planning and scheduling applications. Nodes of the network describe activities (or more specifically, important time points such as start times and end times of activities) while arcs are annotated by temporal relations between the activities such as precedence relations. Traditional 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, for example, by resource constraints. Conditional Temporal Planning (CTP) [10] 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. Recently, logical dependencies between the nodes were added to temporal networks to capture relations between existences of nodes. For example, the logical dependency $A \Rightarrow B$ says that if node A is present in the solution then node B must be present as well. The possibility to select

nodes according to logical, temporal, and resource constrains was introduced to manufacturing scheduling by ILOG in their MaScLib [9]. The same idea was independently formalized in Extended Resource Constrained Project Scheduling Problem (RCPSP) [8]. In the common model each node has a Boolean validity variable indicating whether the node is selected to be in the solution and these Boolean variables participate in binary logical constraints. The validity variable originates from works by Beck and Fox [4] dealing with alternative activities. *Temporal Networks with Alternatives* (TNA) [1] introduced a similar type of implicit alternatives via so called parallel and alternative branching. Basically, TNA is a directed acyclic graph with nodes corresponding to activities and arcs corresponding to temporal relations, where the input and output branchings in nodes (fan-in and fan-out sub-graphs) are annotated either as parallel or alternative branching. If only precedence relations instead of temporal ones are used, then we are talking about a *parallel/alternative graph* (*P/A graph* in short). The input parallel branching for node *x* means that all direct predecessors of *x* (branching nodes) must be processed, while the input alternative branching means that exactly one direct predecessor of *x* must be processed to allow processing of *x* (and similarly for the output branching). These branching relations describe how the processes are being joined and split (Figure 1). Formally, the branching relations implicitly define special logical constraints between the validity variables. It is possible to show that alternative branching with two or more branching nodes cannot be decomposed into binary logical constraints used in MaScLib and Extended RCPSP [3].

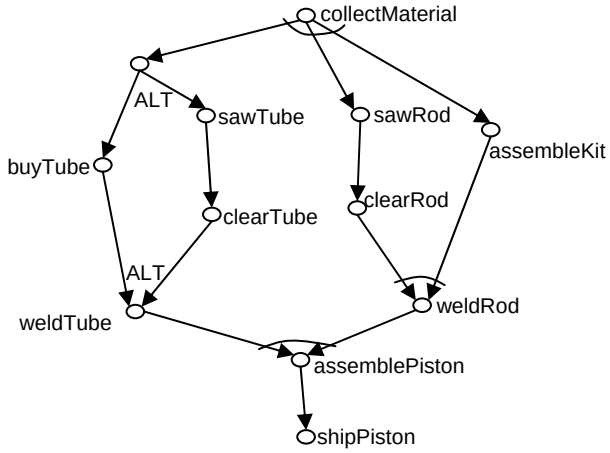


Fig. 1. A real-life manufacturing process with alternatives.

The P/A graph assignment problem [1] is defined as a question whether it is possible to select a subset of nodes satisfying the above branching constraints. In practice, it corresponds to selecting a subset of activities describing a single process among alternative processes. This problem is NP-complete if some node is pre-selected [1]. Beck and Fox [4] informally proposed legal networks where logical reasoning should be easy. Nested TNAs [2] generalize the very same idea using the notion of TNA.

They are motivated by real-life manufacturing processes built by decomposition of meta-processes into operations (the network in Figure 1 is nested). Hence it is not surprising that the structure of Nested TNA is identical to Temporal Planning Networks (TPN) proposed in [7] and also used in TAEMS formalism [6]. TPNs and TAEMS were proposed ad-hoc by giving a language for specifying nested networks and only the nested networks and no attention was paid to theoretical complexity of node selection. Nested TNAs are based on a more general formalism of TNA; examples in [1] show TNAs which cannot be expressed as TPNs.

This paper extends the work [2] (where Nested TNAs were first introduced) by giving formal proofs of soundness and complexity of the algorithm recognizing Nested TNAs and by proving the tractability of P/A graph assignment problem using a constraint model where global consistency is achieved via arc consistency.

3 Nested P/A Graphs

P/A graphs were proposed in [1] to formally describe processes with parallel and alternative branches. Let us first recapitulate the formal definitions of a P/A graph and a P/A graph assignment problem. 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 pairwise 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*.

In this paper, we focus mainly on handling special logical relations imposed by the fan-in and fan-out sub-graphs – we call them *branching constraints*. 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).

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 2: 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.

In the rest of the paper, we will propose a restricted form of the P/A graph, a so called nested P/A graph that can cover many real-life problems while keeping the P/A graph assignment problem tractable.

When we analyzed how the P/A graphs modeling real-life processes look, we noticed several typical features. 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 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 (serial decomposition). 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 (parallel decomposition). 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 (alternative decomposition). The last two decompositions have the same topology of the network (Figure 2), they only differ in the meaning of the branches. Since we focus on modeling instances of processes with particular operations that will be allocated to time, we do not assume loops (used to model abstract processes).

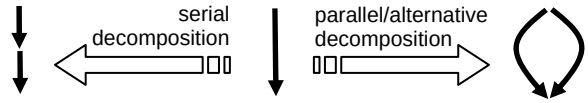


Fig. 2. Possible decompositions of the process.

Based on above observations we propose a recursive definition of a nested graph.

Definition 3: 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 no 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 3, 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 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.

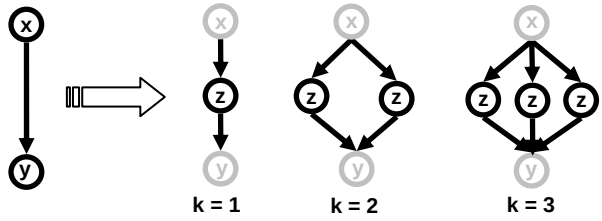


Fig. 3. Arc decompositions in nested graphs.

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. The idea is to annotate each node by input and output label which defines the type of branching. Recall that a fan-out sub-graph with principal node x and branching nodes z_i is a sub-graph consisting of nodes $x, z_1,\ldots, z_k$ (for some k) such that each (x, z_i), $1 \le i \le k$, is an arc in G. Fan-in sub-graph is defined similarly.

Definition 4: *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 4 shows how the labeled nested graph is constructed for the example from Figure 1. Notice how the labels are introduced (a semicircle for PAR label and A for ALT label) or unified in case that one of the labels already exists (see the third step). When a label is introduced for a node, it never changes in the generation process.

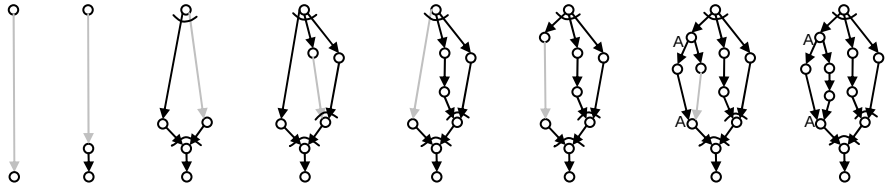


Fig. 4. Building a labelled nested graph.

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. The following lemma shows that the second case always holds.

Lemma 1: For any arc (x, y) in the labeled nested graph, either the output label of x or the input label of y is empty.

Proof: The base nested graph contains a single arc (s, e) and labels for s and e are empty so the arc (the graph) satisfies the lemma. Assume now that graph G satisfies the lemma and we decompose some arc (x, y). During the decomposition, arc (x, y) is removed from the graph and substituted by arcs (x, z_i) and (z_i, y) for new nodes z_i, $1 \le i \le k$, which have empty labels. Hence, the new arcs satisfy the lemma. According to Definition 4 if $k > 1$ the output label of x and the input label of y are set (both or just one of them, if the other one was set already) so we need to check the other arcs going from x or going to y. If there was another arc (x, b) in G in addition to removed (x, y) then some arc (x, c) has already been decomposed to obtain two or more arcs going from x. Hence the output label of x has already been set in G and according to assumption the input label of b was empty which is preserved in the new graph. Symmetrically, if there was additional arc (b, y) in G then the output label of b is empty. So, all arcs in graph G that remain in the new graph still satisfy the lemma. ■

Now, we can formally introduce a nested P/A graph.

Definition 5: *A nested P/A graph* is obtained from a labeled nested graph by removing the labels and defining 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.

Proposition 2: A nested P/A graph is a P/A graph.

Proof: A nested P/A graph is a directed acyclic graph because the base nested graph is acyclic and the decomposition rule does not add a cycle. From Lemma 1, for each arc (x, y) either the output label of x or the input label of y is empty. If both labels are empty then the arc forms a separate fan-in sub-graph. If the output label of x is non-empty then the arc belongs to a fan-out sub-graph with principal node x. Similarly, if the input label of y is non-empty then the arc belongs to a fan-in sub-graph with principal node y. Consequently, each arc belongs to exactly one sub-graph so the nested P/A graph is a P/A graph. ■

3.1 Recognizing Nested P/A Graphs

Proposition 2 claims that a nested P/A graph is a special form of a P/A graph. It is easy to show that there exist P/A graphs which are not nested [1]. Hence, an interesting question is "Can we efficiently recognize whether a given P/A graph is nested?" In this section we will present a polynomial algorithm that can recognize nested P/A graphs by reconstructing how they are built.

First, notice that in a nested P/A graph there are no two different fan-in (fan-out) sub-graphs sharing the same principal node (Definition 5). In other words, either all

arcs going to (from) a given node x belong to a single fan-in (fan-out) sub-graph with the principal node x or there is no fan-in (fan-out) sub-graph with that principal node. This feature is easy to detect so in the rest of the paper, we assume that each node participates as a principal node in at most one fan-in and at most one fan-out sub-graph. This is reflected in the following representation of P/A graphs (Figure 5). The P/A graph is represented as a set of nodes where each node x is annotated by sets of predecessors *pred*(x) and successors *succ*(x) in the graph and by labels *inLab*(x) and *outLab*(x). *inLab*(x) = PAR if x is a principal node in a fan-in parallel sub-graph, *inLab*(x) = ALT if x is a principal node in a fan-in alternative sub-graph. If x is not a principal node in any fan-in sub-graph then *inLab*(x) is empty. A similar definition is done for *outLab*(x) with relation to fan-out sub-graphs. Notice the similarity of labels to labeled nested graphs (Definition 4). The reader should realize that any nested P/A graph can be represented this way: all fan-in and fan-out sub-graphs correspond to non-empty labels and for any arc (x, y) either the label *outLab*(x) or *inLab*(y) is empty.

node	*pred*	*succ*	*inLab*	*outLab*
1		2,3,4	-	PAR
2	1	5,6	-	ALT
3	1	8	-	-
4	1	10	-	-
5	2	9	-	-
6	2	7	-	-
7	6	9	PAR	-
8	3	10	PAR	-
9	5,7	11	ALT	-
10	4,8	11	PAR	-
11	9,10	12	PAR	-
12	11		PAR	-

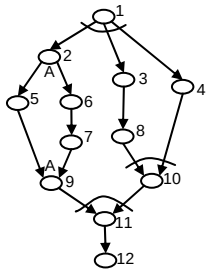


Fig. 5. Representation of a (nested) P/A Graph.

The following algorithm *DetectNested* recognizes labeled nested graphs by reconstructing how they are built. Figure 6 illustrates the recognition process. Notice that some nodes in Q are not eligible for contraction because the condition in line 4 fails (e.g. node 10 in step 3, since 1 has two successors and 11 has two predecessors).

```
algorithm DetectNested(input: graph G, output: {success, failure})
1.   select all nodes x in G such that |pred(x)| = |succ(x)| = 1
2.   sort the selected nodes lexicographically according to index
          (pred(x), succ(x)) to form a queue Q
3.   while non-empty Q do
4.      select and delete a sub-sequence L of size k in Q such that
           all nodes in L have an identical index ({x}, {y}) and
           either |succ(x)| = k or |pred(y)| = k
5.      if no such L exists then stop with failure
6.      if k > 1 & outLab(x) ≠ inLab(y)  then stop with failure
7.      remove nodes z∈L from the graph
8.      remove nodes x, y from Q (if they are there)
9.      add arc (x,y) to the graph (an update succ(x) and pred(y))
10.     if |pred(x)| =  |succ(x)| = 1 then insert x to Q
11.     if |pred(y)| =  |succ(y)| = 1 then insert y to Q
12.  end while
13.  if the graph consists of two nodes connected by an arc then
14.     stop with success
15.  else stop with failure
```

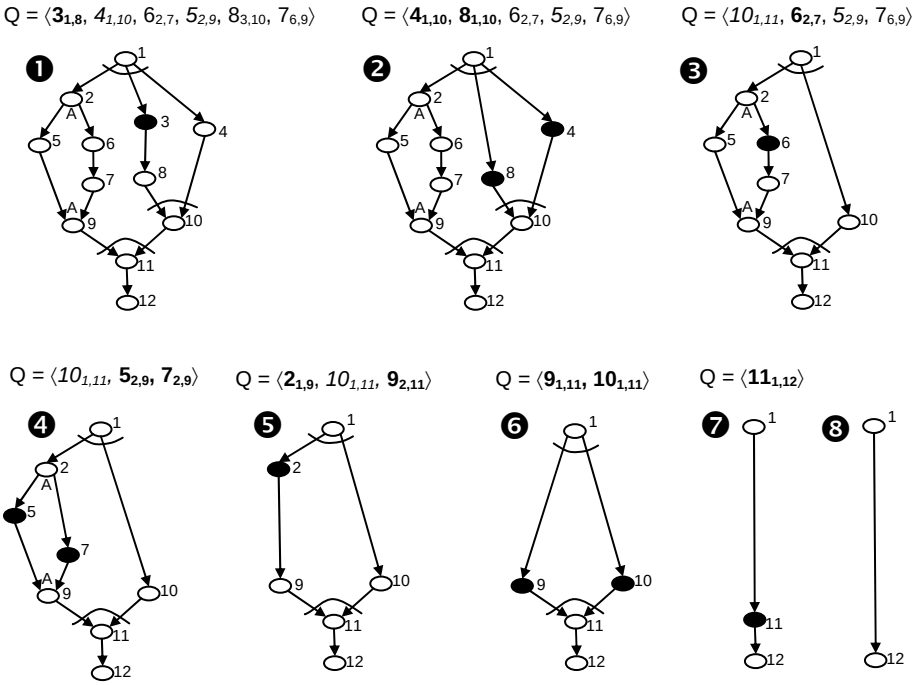


Fig. 6. Detecting nested P/A graphs by sub-graph contraction (contracted nodes are in black).

Proposition 3: Algorithm *DetectNested* always terminates and it stops with success if and only if the input P/A graph is nested.

Proof: Each line of the algorithm terminates. The body of the while loop either terminates with a failure or at least one node is removed from the graph. Because the queue Q consists of nodes that are part of the current graph, it must become empty sometime so the while loop terminates and hence the whole algorithm terminates.

We will show that the algorithm recognizes labeled nested graphs by induction on the number of decomposition steps necessary to generate a graph. The base nested graph is trivially recognized in line 13. Assume now that the algorithm can recognize all nested graphs built using *m* steps. We shall show that:

(i) if *DetectNested* fails to find a set L of nodes to be contracted then the input graph is not a labeled nested graph, and

(ii) if *DetectNested* finds a set L of nodes and contracts them and the input graph is a labeled nested graph build using (*m*+1) steps, then the resulting graph is a labeled nested graph which can be built using *m* steps.

It is easy to see that these two claims are sufficient for the proof of the equivalence part of the proposition.

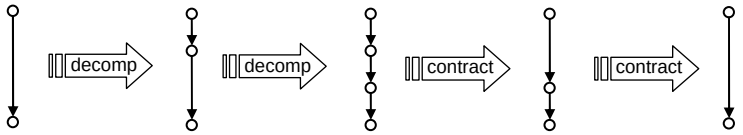
To prove (i) it is enough to realize that in any labeled nested graph constructed in accordance with Definition 3, the nodes added in the last decomposition step always

fulfill the requirements on the set L in *DetectNested*. Thus if *DetectNested* fails to find a suitable set L then the input graph is not a labeled nested graph.

The proof of (ii) is more difficult because of the fact that there may be many suitable sets L = { z_1, …, z_k } which *DetectNested* may find and contract. We have to show that any such choice produces a graph, which is labeled nested and can be built using m steps. Let us consider two cases:

a) $k > 1$. In this case a parallel or alternative sub-graph with nodes x, y, z_1, …, z_k and arcs (x, z_i), (z_i, y) is contracted into arc (x, y). Notice, that (using the assumption that the input graph is nested) this sub-graph must be a result of an arc decomposition of (x, y) during the recursive construction and moreover no arc inside this sub-graph is further decomposed. Therefore the graph which is obtained from the input graph by contraction of L is a labeled nested graph obtainable in m steps. The sequence of decomposition steps is the same as for the input graph except that the decomposition of (x, y) is skipped.

b) $k = 1$. In this case a chain of length ≥ 2 is shortened (by one vertex and one arc) by the contraction of L. In this case there is no guarantee that the contraction can be matched to a decomposition step which built the input graph (see example below).



However, the chain of length l can be produced from a single arc by (l-1) decompositions and can be contracted back into a single arc by (l-1) contractions in *DetectNested* (all with |L| = 1). Thus, similarly as in case a) the graph which is obtained from the input graph by contraction of L is a labeled nested graph obtainable in m steps. The sequence of decomposition steps is the same as for the input graph except that the sub-sequence (not necessarily a sub-interval) of (l-1) decompositions which built the chain is replaced by (l-2) decompositions which build the shorter chain. ■

Proposition 4: The worst-case time complexity of algorithm *DetectNested* is $O(n^2)$, where n is a number of nodes in the graph.

Proof: The initial selection of nodes for the queue can be done in time $O(n)$. Time $O(n.\log n)$ is necessary to sort the queue. The sub-list for contraction can be selected in time $O(n)$ and insertion of nodes into the list can be done in $O(n)$. All other operations can be implemented in constant time. The while-loop is repeated at most n times because each time at least one node is removed from the graph. Together, the while loop takes time $O(n^2)$ so the whole algorithm takes time $O(n^2)$. ■

3.2 Tractability and Constraint Models

The main motivation for introducing nested P/A graphs was to make the P/A graph assignment problem tractable for this special subclass of graphs. Recall that the

assignment problem consists of deciding whether it is possible to complete a partial assignment of validity variables for nodes to obtain a complete feasible assignment. We can reformulate the P/A graph assignment problem as a constraint satisfaction problem in the following way. 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:

$$v_x = v_y.$$

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:

$$v_x = \Sigma_{j=1,\ldots,k}\, v_{yj}.$$

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_{yj} is assigned to 1. Using the arithmetic constraint simplifies a lot the formal model of the logical relation. Notice, that the task whether a completion of the partial assignment of validity variables satisfying all constraints exists is clearly equivalent to the assignment problem for the original P/A graph. Hence, if some local (polynomial) consistency such as arc consistency implies global consistency for the constraint model then the P/A graph assignment problem is trivially tractable. Recall that global consistency means that any value in the domain of a variable is part of some solution so if no domain is empty then a feasible assignment exists and can be found in polynomial time using a backtrack-free search. Unfortunately, arc consistency does not guarantee global consistency of the above-described *basic constraint model*. Assume a simple graph with two alternative branchings that are modeled using constraints $v_a = v_b + v_c$ and $v_d = v_b + v_c$. If we set v_a to 1 then arc consistency is not able to deduce that v_d must also be 1 and hence the problem is not globally consistent. Nevertheless, if we use an equivalent model $v_a = v_b + v_c$ and $v_d = v_a$ then arc consistency implies global consistency. Based on this idea we will now propose an equivalent constraint model of the nested P/A graph where arc consistency implies global consistency.

The *nested constraint model* for a nested P/A graph is designed incrementally along the process of building the nested graph. The base nested graph is modeled by a single constraint $v_s = v_e$. Assume that arc (x, y) is decomposed into a sub-graph (Figure 3) with new nodes $z_1, \ldots, z_k$ and new arcs (x, z_i), (z_i, y). If the decomposition is parallel or $k = 1$ then we add constraints $y = z_i$ into the constraint model. Recall that this decomposition is allowed only if (x, y) was already a part of a serial or a parallel decomposition (or the initial arc - see Definition 4), which implies by induction on the number of decomposition steps that there is already a constraint $x = y$ in the model. Hence, the constraints $x = y$, $y = z_i$ are equivalent to the constraints of the basic model $x = z_i$, $y = z_i$. Assume now that the decomposition is alternative. It means that the arc (x, y) was a part of a serial or of an alternative decomposition (or the initial arc). In the second case, without loss of generality let us assume that x was the principal node of the alternative branching and z_j', $j = 1,\ldots, m$ ($m \geq 0$) are the remaining branching

nodes (if any) in addition to y. Now, we add a constraint $y = \Sigma_{i=1,\ldots,k}\, z_i$. The model before decomposition implies (by induction on the number of decomposition steps) the constraint $x = y + \Sigma_{j=1,\ldots,m}\, z_j$' (this constraint is either explicitly in the model or there is a set of constraints equivalent to this constraint). The added and the implied constraints are equivalent to constraints $x = \Sigma_{i=1,\ldots,k}\, z_i + \Sigma_{j=1,\ldots,m}\, z_j$', $y = \Sigma_{i=1,\ldots,k}\, z_i$ which are used in the basic model to describe such a branching. Altogether, the modified constraint model of the nested P/A graph is equivalent to the original model (in terms of having the same set of feasible assignments). Note that the algorithm *DetectNested* reconstructs the decomposition process so we can define the nested constraint model for any given nested P/A graph. Figure 7 illustrates the process of building the nested constraint model and compares both nested and basic models.

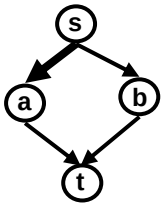
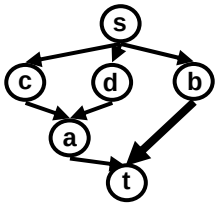
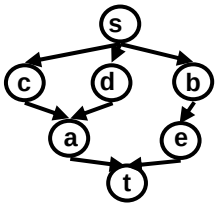
network				
basic model	s = t	s = a + b t = a + b	s = c + d + b a = c + d t = a + b	s = c + d + b a = c + d t = a + e b = e
nested model	s = t	s = t ***t = a + b***	s = t t = a + b ***a = c + d***	s = t t = a + b a = c + d ***b = e***

Fig. 7. Branching constraints as arithmetic formulas over 0-1 variables (all fan-in/fan-out sub-graphs are assumed to be alternative).

Proposition 5: The assignment problem for a nested P/A graph is tractable (can be solved in a polynomial time).

Proof: We shall show that the modified constraint model is Berge acyclic for which it is known that arc consistency implies global consistency [5]. The constraint model for the base nested graph consists of a single constraint so it is Berge acyclic. Any constraint added to the model after arc decomposition contains exactly one variable of the former model and new variable(s). Hence, it cannot introduce a Berge cycle. ■

6 Conclusions

The paper studies a recursive definition of temporal networks with alternative processes, so called nested temporal networks with alternatives [2], motivated by a structure of typical manufacturing processes. Though this structure is identical to TPN [7] and TAEMS framework [6], our approach to modeling branching relations

resembles more the work [4]. Surprisingly nobody so far paid attention to logical reasoning on these networks which is the main focus of this paper. We proposed an algorithm that can recognize nested TNAs, we proved its soundness, and showed its complexity. We also showed that the assignment problem for Nested TNAs is tractable by using a Berge acyclic constraint model of the problem. This constraint model applied to nested subparts of a TNA can even improve practical efficiency when solving the P/A graph assignment problem for general TNAs as showed in [3]. To solve real-life problems, the proposed model can be combined with other constraints such as the constraints modeling temporal and resource restrictions as demonstrated in [4,8,9]. Note finally, that despite a visual similarity to AND/OR graphs the concept of alternative branching differs from OR branching (the alternative branching allows two types of feasible assignment: either all nodes in the branching are invalid or the principal node is valid and exactly one branching node is valid). Moreover, opposite to AND/OR graphs, where only fan-out branching is marked by AND/OR label, P/A graphs specify both fan-in and fan-out branching for nodes.

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

References

1. Barták, R. and Čepek, O. Temporal Networks with Alternatives: Complexity and Model. In *Proceedings of the Twentieth International Florida AI Research Society Conference (FLAIRS)*, pp. 641–646, AAAI Press (2007)
2. Barták, R. and Čepek, O. Nested Temporal Networks with Alternatives: Recognition and Tractability. *Applied Computing 2008 - Proceedings of 23rd Annual ACM Symposium on Applied Computing*, Volume 1, pp. 156-157, ACM (2008)
3. Barták, R., Čepek, O., and Surynek, P. Discovering Implied Constraints in Precedence Graphs with Alternatives. Annals of Operations Research, Springer (2008), to appear.
4. Beck, J. Ch. and Fox, M. S. Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence* 121, 211–250 (2000)
5. Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. On the desirability of acyclic database schemes. *Journal of the ACM* 30, 479–513 (1983)
6. Horling, B., Leader, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., and Harvey, A. The Taems White Paper, University of Massachusetts (1999) http://mas.cs.umass.edu/research/taems/white/taemswhite.pdf
7. Kim, P. Williams, B. and Abramson, M. Executing Reactive, Model-based Programs through Graph-based Temporal Planning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 487 – 493, (2001)
8. Kuster, J., Jannach, D., and Friedrich, G. Handling Alternative Activities in Resource-Constrained Project Scheduling Problems. In *Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 1960–1965 (2007)
9. Nuijten, W., Bousonville, T., Focacci, F., Godard, D., and Le Pape, C. MaScLib: Problem description and test bed design. (2003) http://www2.ilog.com/masclib
10. Tsamardinos, I., Vidal, T., and Pollack, M. E. CTP: A New Constraint-Based Formalism for Conditional Temporal Planning. *Constraints*, 8(4), 365–388 (2003)