Nested Temporal Networks with Alternatives: Recognition and Tractability

CSP - 100

ABSTRACT

Temporal Networks with Alternatives were proposed to model alternative and parallel processes in planning and scheduling applications. However the problem of deciding which nodes can be consistently included in such networks is NP-complete. In this paper we study a tractable subclass of Temporal Networks with Alternatives that covers a wide range of real-life processes, while the problem of deciding node validity is solvable in polynomial time. We also present an algorithm that can effectively recognize whether a given network belongs to the proposed sub-class.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems – *routing and layout, sequencing and scheduling.*

General Terms

Algorithms, Performance, Theory.

Keywords

Temporal networks, alternatives, constraints, complexity.

1. INTRODUCTION

Temporal networks play a crucial role in modeling temporal relations in planning and scheduling applications. 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. 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 (TPN) [8] and TAEMS framework [7] also include conditional branching. These networks appear during Hierarchical Task Network Planning and they describe all alternative plans in a single graph. In other problems, such as log-based reconciliation [6], the logical dependencies between the nodes are modeled explicitly. 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 [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 [2] 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. Recall that nodes usually correspond to activities (their start and/or end time) and the task is to allocate activities to time (and to resources), and also to decide which activities will actually be present in the solution. Hence, these frameworks are appropriate for modeling and solving over-subscribed scheduling problems or problems with alternative activities. Temporal Networks with Alternatives (TNA) [1] introduced a different type of alternatives with so called parallel and alternative branching. This concept is similar to TPN – logical constraints are described as a part of branching in nodes – but TNA are more general than TPN. Unfortunately, the problem of deciding which nodes can be consistently selected, if some nodes are pre-selected, is NP-hard.

In this paper we propose a restricted form of TNA which we call Nested Temporal Networks with Alternatives. This restriction is motivated by real-life manufacturing scheduling problems where the network of alternatives has a specific topology similar to TPN. The main contribution is a new algorithm for recognizing and reconstructing the nested structure and a new constraint model describing the branching constraints. This model achieves global consistency in polynomial time which implies tractability of the above mentioned assignment problem. After a motivation example and recapitulation of TNA, we will formally define Nested TNA and present an algorithm that can recognize whether a given TNA is nested. The same algorithm (after a small extension) can also be used to design a new logical constraint model that achieves global consistency and hence solve the TNA assignment problem [1].

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

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

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 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.

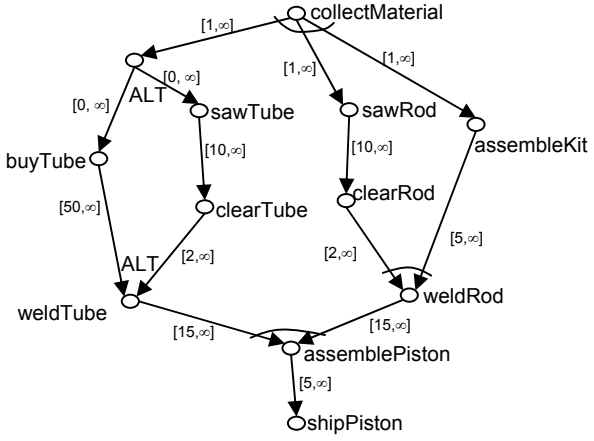


Figure 1. A manufacturing process with alternatives.

The manufacturing processes from the above problem can be described using a Simple 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). We can see that this TNA has a very specific topology that we will try to address in the rest of the paper.

Let us now formally define Simple 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 $\forall i = 1,\ldots,k: z \neq y_i$) 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: *Simple 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, $a \le Y\text{-}X \le b$.

Figure 1 shows an example of Simple 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*) each form a simple fan-in (or fan-out, it does not matter in this case) sub-graph. 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.

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). 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.

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.

3. NESTED P/A GRAPHS

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 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 (Figure 2), 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.

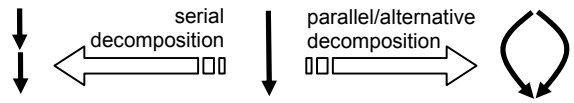


Figure 2. Possible decompositions of the process.

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

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.

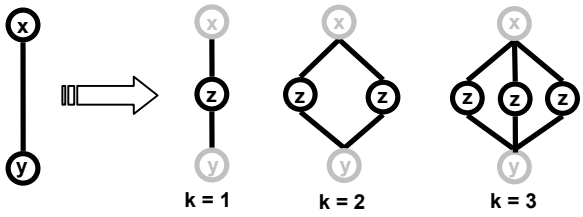


Figure 3. Arc decomposition in nested graphs.

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 4 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.

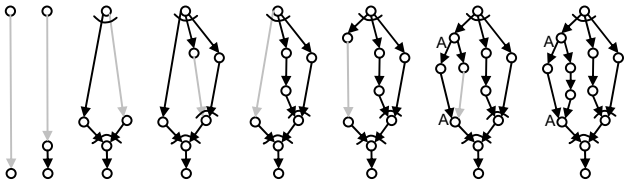


Figure 4. Building a labelled nested graph.

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 (the proof is omitted due to limited space).

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.

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.

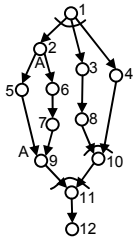
Proposition 2: A nested P/A graph is a P/A graph.

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 input label of x is non-empty then the arc belongs to a fan-out sub-graph with principal node x. Similarly, if the output 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. Hence, an interesting question is whether we can 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 6). 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 5). 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	-

Figure 5. Representation of a (nested) P/A graph.

The following algorithm *DetectNested* recognizes labeled nested graphs by reconstructing how they are built.

```
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
```

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.

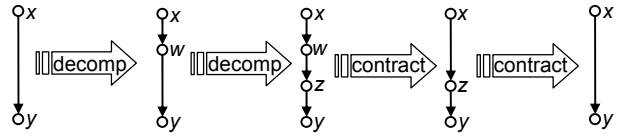
To prove (i) it is enough to realize that in any labeled nested graph constructed in accordance with Definition 4, the nodes added in the last decomposition step always fulfill the requirements on the set L in *DetectNested*. Thus if *DetectNested* fails to find a suitable set L then the input graph 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 the arc was already part of no branching or part of a parallel branching (see Definition 5) 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 either part of no branching or part of alternative branching. In the second case, without lost of generality let us assume that x was the principal node of that 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 a part of the model or there is a set of constraints equivalent to this constraint). Again, these 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 modified constraint model for any given nested P/A graph.

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 [3]. 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. □

4. CONCLUSIONS

The paper proposes a recursive definition of temporal networks with alternative processes, so called nested temporal networks with alternatives, motivated by a structure of typical manufacturing processes. Though this structure is very similar to TPN [8] and TAEMS framework [7] nobody so far paid attention to logical reasoning on these networks. We proposed an algorithm that can recognize nested TNAs and we showed that the assignment problem for Nested TNAs is tractable. For the proof of tractability we constructed a Berge acyclic constraint model that can be also used to solve the problem. Moreover, this model can be combined with other constraints such as the constraints modeling temporal and resource restrictions.

To justify the proposed nested constraint model we did a preliminary experiment comparing runtimes of the basic constraint model and the nested model using `clpfd` library of SICStus Prolog 4.0.1. We solved the assignment problem for randomly generated Nested TNAs with one pre-selected node. For both models a standard backtracking algorithm with maintaining arc consistency and Brélaz variable ordering was used. Figure 6 shows the distribution of runtimes (milliseconds; logarithmic scale; 1.1 GHz Pentium M) as a function of the number of nodes. Note that the runtime for the nested model includes the time to recognize the nested structure and to generate the constraints; the actual solving time is below 100 ms thanks to global consistency implying no backtracks during search (while the basic model requires search to find a solution). For completeness, we should say that a time limit of 10 minutes was used and many larger problems were not solved within this limit when using the basic constraint model. The graph shows that the overhead necessary to build the nested model pays off as soon as the problems become larger. The real-life problems are expected to contain thousands of nodes so the proposed nested constraint model is useful there.

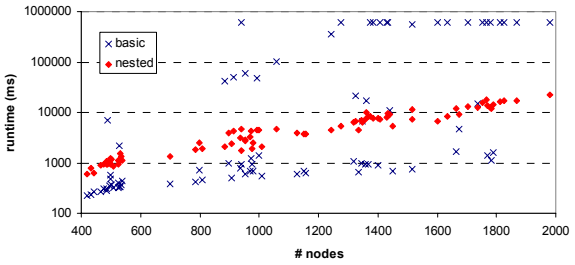


Figure 6. Runtimes for nested and basic constraint models.

5. 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)*, AAAI Press, 2007, pp. 641–646.

[2] Beck, J. Ch. and Fox, M. S. Scheduling Alternative Activities. In *Proceedings of AAAI-99*, AAAI Press, 1999, pp. 680-687.

[3] Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. On the desirability of acyclic database schemes. *Journal of the ACM* 30, 1983, pp. 479–513.

[4] Blythe, J. An Overview of Planning Under Uncertainty. *AI Magazine* 20(2), 1999, pp. 37–54.

[5] Dechter, R., Meiri, I., and Pearl, J. Temporal Constraint Networks. *Artificial Intelligence* 49, 1991, pp. 61–95.

[6] Hamadi, Y. Cycle-cut decomposition and log-based reconciliation. In *ICAPS Workshop on Connecting Planning Theory with Practice*, 2004, pp. 30–35.

[7] 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

[8] 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)*, 2001, pp. 487 – 493.

[9] 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)*, 2007, pp. 1960–1965.

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

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

[12] 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

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

[14] Tsamardinos, I., Vidal, T., and Pollack, M. E. CTP: A New Constraint-Based Formalism for Conditional Temporal Planning. *Constraints*, 8(4), 2003, pp. 365–388.