manuscript No.
(will be inserted by the editor)

Hyper Temporal Networks

A Tractable Generalization of Simple Temporal Networks and its relation to Mean Payoff Games

Carlo Comin · Roberto Posenato · Romeo Rizzi

Abstract Simple Temporal Networks (STNs) provide a powerful and general tool for representing conjunctions of maximum delay constraints over ordered pairs of temporal variables. In this paper we introduce Hyper Temporal Networks (HyTNs), a strict generalization of STNs, to overcome the limitation of considering only conjunctions of constraints but maintaining a practical efficiency in the consistency check of the instances. In a Hyper Temporal Network a single temporal hyperarc constraint may be defined as a set of two or more maximum delay constraints which is satisfied when at least one of these delay constraints is satisfied. HyTNs are meant as a light generalization of STNs offering an interesting compromise. On one side, there exist practical pseudo-polynomial time algorithms for checking consistency and computing feasible schedules for HyTNs. On the other side, HyTNs offer a more powerful model accommodating natural constraints that cannot be expressed by STNs like *"Trigger off exactly δ min before (after) the occurrence of the first (last) event in a set."*, which are used to represent synchronization events in some process aware information systems/workflow models proposed in the literature.

Keywords Simple Temporal Networks · Temporal Consistency · Hypergraphs · Pseudo-polynomial Time Algorithms · Mean Payoff Games · Disjunctive Temporal Problems · Workflows

1 Introduction

In many areas of Artificial Intelligence (AI), including planning, scheduling and workflow management systems, the representation and management of quantita-

Carlo Comin
Department of Mathematics, University of Trento, Trento, Italy *and* LIGM, Université Paris-Est in Marne-la-Vallée, Paris, France. Supported by Department of Computer Science, University of Verona under PhD grant "Computational Mathematics and Biology" on a co-tutelle agreement with Université Paris-Est in Marne-la-Vallée.
E-mail: carlo.comin@unitn.it

Roberto Posenato, Romeo Rizzi
Department of Computer Science, University of Verona, Verona, Italy
E-mail: {roberto.posenato, romeo.rizzi}@univr.it

tive temporal aspects is of crucial importance [4, 10, 11, 20, 37, 43]. Examples of possible quantitative temporal aspects are: constraints on the earliest start time and latest end time of activities, constraints over the minimum and maximum temporal distance between activities, etc.

In many cases these constraints can be represented as an instance of a *Simple Temporal Network (STN)* [18], a directed weighted graph where each node represents a time-point variable (timepoint), usually corresponding to the beginning or the end of an activity, and each arc specifies a binary constraint on the scheduling times to be assigned to its endpoints. In [18], each arc is labeled with a closed interval of real values: for example, the labeled arc $u \xrightarrow{[x,y]} v$ encodes the binary constraint $x \leq v - u \leq y$ over its endpoints u and v. A more uniform and elementary representation of an STN is provided by its *distance graph*[1] [18], a graph having the same set of nodes as the original one, but where each arc $u \xrightarrow{[x,y]} v$ is replaced by two arcs, each labeled with a single real value: arc $u \xrightarrow{y} v$ to express the constraint $v - u \leq y$, and arc $v \xrightarrow{-x} u$ to express the constraint $u - v \leq -x$, i.e., $x \leq v - u$.

An STN is said to be *consistent* if it is possible to assign a real value to each timepoint so that all temporal constraints are satisfied. The consistency property can be verified by searching for negative cycles in the distance graph and it is well known that the consistency check and the determination of the earliest/latest value for each timepoint can be done in polynomial time [18].

However, STNs do not allow the expression of constraints like *"trigger off an event exactly δ min after the occurrence of the last of its predecessors"*, which are a quite natural constraints to represent synchronization events in a process aware information system plan/workflow schema [25]. This is because in STNs, and in some of their natural extensions, (1) it is not possible to represent a single constraint involving more than two timepoints and (2) all constraints have to be satisfied in order to have the network consistent. On the contrary, the above constraint about a synchronization event can be represented as a set of distance constraints, each involving a different pair of timepoints, that is considered satisfied when at least one of set components is satisfied. In order to represent and analyze disjunctive constraints like the above one, it is then necessary to consider models like *Disjunctive Temporal Problem* (DTP) [44] where a constraint is a set of disjunctive difference constraints over the timepoints. The drawback of such model is that the consistency check problem is NP-complete [44].

1.1 Contribution

In this article we propose to generalize STN to *Hyper Temporal Network* (HyTN), which allows also the expression of constraints like the above one regarding synchronization events, but where the consistency check is amenable of effective solution algorithms.

Moreover, we show an interesting link between the consistency check of HyTNs and resolution in Mean Payoff Games (MPG), a family of perfect information

[1] Distance graph is also called *constraint graph* by other authors [17]. Moreover, Bellman [3] was the first to describe the relation between shortest paths and difference constraints in a constraint graph.

games played on graphs by two opponents [21], for which some pseudo-polynomial time algorithms for determining winning strategies are known [7, 47].

A preliminary version of this article appeared in the proceedings of TIME symposium [15]. Here we extend the presentation as follows: (1) the definition of HyTN has been extended in order to allow the presence of two kinds of hyperarcs; (2) the motivating example section has been revised to show how the new kind of hyperarc can be used; (3) some further issues and pertinent properties about HyTN have been introduced and proved; (4) several proofs have been expanded and clarified; (5) the experimental analysis of the consistency check algorithm has been improved considering more recent algorithms [7] for finding winning strategies for MPGs. This has improved the performances dramatically.

Organization The remainder of the article is organized as follows. In Section 2 we present a motivating example from the domain of the workflow-based process management to bring out HyTNs. Section 3 introduces some definitions and well-known results for STNs and introduces some definitions about hypergraphs. The generalization of STNs into HyTNs and the definition of consistency problem for HyTNs are presented in Section 4. In Section 5 we recall the main facts and results about Mean Payoff Games which are useful for the following sections. Section 6 presents the investigation into the link between the HyTN consistency problem and Mean Payoff Games deriving pseudo-polynomial time algorithms for checking the consistency of HyTNs and computing feasible schedules whenever they exist. Some empirical evaluations of the proposed algorithms are reported in Section 7. In Section 8, some related works are presented and discussed with respect to our approach. Section 9 summarizes the main facts brought to light in this article and presents a possible future development of the work we are currently carrying on.

2 Motivating Examples

In the introduction we have briefly recalled a kind of constraint that cannot be expressed within STNs. In this section, we describe in more detail two examples of temporal constraints that cannot be fully described in an STN in order to introduce and motivate the new expressive capability of our model. As a further motivation, at the end of the section we also spotlight how this new capability has been recently exploited to check the consistency of Conditional Simple Temporal Networks (CSTNs) [45] in a more efficient way.

Let us consider an example in the domain of the workflow-based process management, a domain concerned with the coordination and control of business processes using information technology. A *workflow* is a representation of a business process as the coordinated execution of activities by human or automatic executors (agents). A *Workflow management system* (WfMS) is a software system that supports the automatic execution of workflows [25]. In a WfMS, the management of temporal aspects is a critical component and in the literature there are many proposals on how to extend a workflow in order to represent and manage temporal constraints of a business process [4, 8, 10–13, 19, 20, 23]. In particular, in [4,8,13,19,20] authors show how to represent and manage some kinds of temporal constraints using specific algorithms, while in [10–12,23] authors show how it is

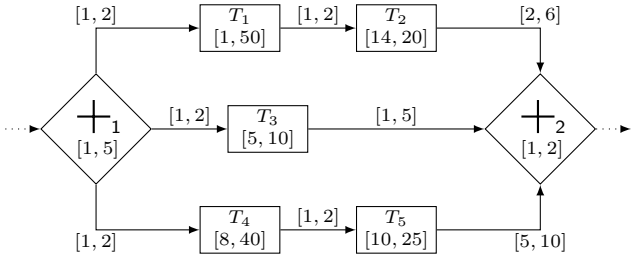


Fig. 1: A simple workflow schema excerpt with three parallel flows of execution.

possible to represent and manage a wider class of temporal constraints exploiting models like Time Petri Nets [35] or STNs/STNUs [36].

In this paper we consider an excerpt of the conceptual temporal model proposed by Combi et al. [12], where the specification of a temporal workflow is given by a *workflow schema*, a directed graph (also called workflow graph) where nodes correspond to activities and arcs represent control flows that define activity dependencies on the order of execution. Both nodes and arcs may be associated to temporal ranges to specify temporal constraints. There are two different types of activity: tasks and connectors. Tasks represent elementary work units that will be executed by external agents. Each task is graphically represented by a box containing a name and a temporal range that specifies the allowed temporal span for its execution. Connectors represent internal activities executed by the WfMS to achieve a correct and coordinated execution of tasks. They are graphically represented by diamonds and, as with tasks, each of them has a temporal range that gives the temporal span allowed to the WfMS for executing it. Every arc has a temporal property that gives the allowed times that can be spent by the WfMS for possibly delaying the consideration of the next activity after the end of the previous one. There are different kinds of connector that allow one to modify a control flow. *Split* connectors are nodes with one incoming arc and two or more outgoing arcs: after the execution of the predecessor, (possibly) several successors have to be considered for the execution. The set of nodes that can start their execution is given by the kind of split connector. A split connector can be: Parallel, Alternative or Conditional. *Join* connectors are nodes with two or more incoming arcs and one outgoing arc only. The types of activities considered in [12] are a subset of the possible activities specified by the Workflow Management Coalition [1, 25].

Fig. 1 shows a simple workflow schema where the Parallel connector $+_1$ splits the flow into three parallel flows of execution (one for the sequence of tasks T_1 and T_2, one for task T_3, and one for task T_4 and T_5) that have to be joined (synchronized) by the AND join connector $+_2$ before continuing the execution; all temporal ranges are in minutes.

Let us consider the connector $+_2$; according to the recommendations from the Workflow Management Coalition (WfMC) [25] and the temporal specification from [12], the execution of this connector requires to wait all incoming flows and, after the last incoming flow, to wait a time according to the connector temporal range before following the outgoing arc. In other words, the incoming flows can arrive at different instants but only when the last one arrives, the connector has to be activated in order to continue with the execution.

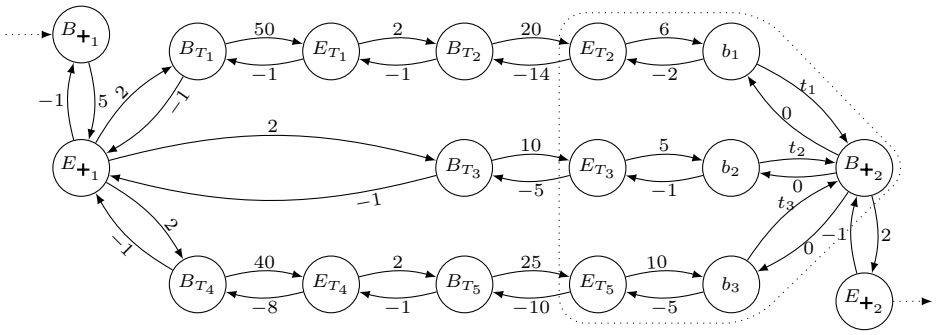


Fig. 2: An STN representing temporal aspects of the workflow depicted in Fig. 1. The dotted region emphasizes, within the workflow excerpt, the connections to an AND join connector.

Combi et al. [12] proposed a method to translate workflow schemata to STNs/ STNUs [36] in order to analyze and validate all temporal aspects in a rigorous way. As already noted in [9] and [10], such translation cannot specifically represent the behavior of an AND join connector, because the kind of constraints in an STN/ STNU is limited. Therefore, in [10], the authors proposed an adjustment of the translation of an AND join connector introducing for each incoming arc of the connector a *buffer* node connected with some determined new arcs and assuming a reasonable but fixed execution algorithm for the STN. In more detail, let us consider Fig. 2 that depicts the representation of workflow of Fig. 1 by means of an STN following partially the method described in [10] (without loss of generality, here we convert task constraints as STN arcs instead of STNU contingent ones because we are interested only in the AND join conversion). Each activity of the workflow is represented by two STN nodes, one to represent the begin timepoint, B_i, one for the end one, E_i, and temporal ranges in the workflow are represented by STN arc labels. Regarding the translation of the AND join node $+_2$, nodes representing the task endings on parallel flows, E_{T_2}, E_{T_3}, and E_{T_5}, are connected to *buffer* nodes b_1, b_2, and b_3 that allow the parallel flows to complete their execution following only their temporal constraints. Then, b_1, b_2, and b_3 are connected to node B_{+_2} (which represents the begin instant of the AND join connector) by temporal constraints $[0, t_1], [0, t_2]$ and $[0, t_3]$, where the values t_1, t_2, and t_3 are determined during the workflow-to-STN conversion as explained in [10].

Now, let us consider a possible execution scenario. If b_1, b_2, and b_3 occur all together at instant 20, then, following the proposed temporal semantics [12], the only possible instant value for B_{+_2} must be 20 while the updated STN allows any value in the range $[20, 20 + \min\{t_1, t_2, t_3\}]$. In [10], the authors showed that the right value is always the lower bound of such extended range and, therefore, it is sufficient to adopt an early execution strategy in order to choose the right value for timepoint B_{+_2}.

In other words, the proposed translation has two drawbacks: (1) it requires some preliminary computations for determining t_1, t_2, and t_3 values, and (2) the resulting STN admits some solutions that are not admissible by the semantics of the AND join connector.

To specifically represent the behavior of an AND join connector with respect to its predecessor time points without auxiliary conditions or analysis, it is necessary

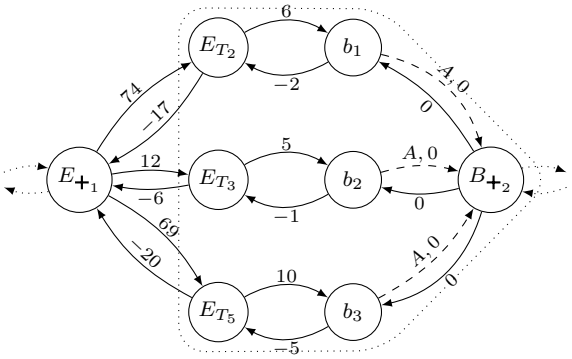
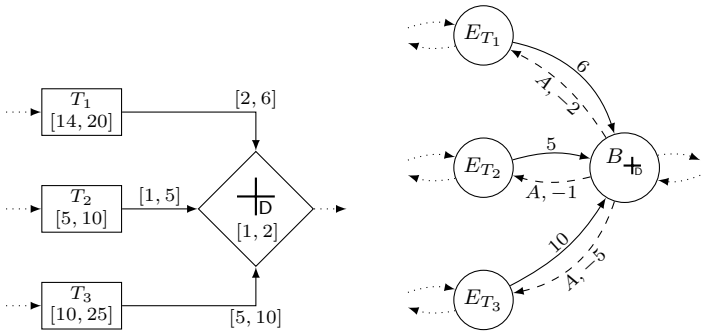


Fig. 3: An augmented STN, that we call HyTN, where dashed arcs represent components of hyperarcs, a new kind of constraint. This HyTN improves the representation of the STN of Fig. 2. To emphasize the changes, here we have summed all arcs outside the dotted region.

to introduce a new kind of constraint based on hyperarcs, as shown in Fig. 3. In the figure the *multi-tail hyperarc* A consists of three dashed arcs—called *components*—and replaces the arcs from b_i $(i = 1, 2, 3)$ to B_{+_2} of Fig. 2. We say that a multi-tail hyperarc is satisfied if at least one of its components is satisfied. In Fig. 3 dashed arcs define the hyperarc A that is satisfied if B_{+_2} is 0 distant from at least one time point among b_1, b_2, and b_3. Since B_{+_2} is constrained to occur at the same instant or after each time point b_1, b_2, and b_3 by the arcs between B_{+_2} and b_i, $i = 1, 2, 3$, the result is that to satisfy A it is necessary that B_{+_2} occurs at the same instant of the last time point among b_1, b_2, and b_3, as required originally. In more general, a multi-tail hyperarc is defined as a set of distance constraints (components) between some time points and a common end point.

The use of hyperarcs allows also the representation of temporal aspect of other advanced connectors as, for example, the Structured Discriminator [1]. The Structured Discriminator connector provides a means of merging two or more distinct flows in a workflow instance into a single subsequent. In particular, it triggers the subsequent flow as soon as the the first incoming flow arrives. The arrival of other incoming flows thereafter have no effect on the subsequent flow. As such, the Structured Discriminator provides a mechanism for progressing the execution of a process once the first of a series of concurrent tasks has completed and according to the connector temporal range. Fig. 4a depicts an excerpt of a workflow schema containing a structured discriminator connector, $\frac{1}{6}$, that joins three parallel flows.

At the best of our knowledge, currently there are no proposals for the representation of temporal constraints of a discriminator connector in any temporal workflow model or process-aware information system [32]. Even exploiting the methodology proposed in [10], it is easy to verify that it is not possible to represent such connector as an STN because in a consistent STN all constraints have to be satisfied while here it is necessary to allow the possibility that only one constraint of a set has to be satisfied in order to specifically represent a discriminator connector. A possible way for specifically managing a discriminator connector consists in following the approach suggested by [10] for representing activities and considering a variant of the multi-tail hyperarc, called *multi-head hyperarc*, for representing its temporal constraint, as depicted in Fig. 4b. In the figure there is a multi-head



(a) An excerpt of a workflow schema containing a structured discriminator connector, $+_{15}$.

(b) A possible representation of temporal aspects of a discriminator connectors by means of a multi-head hyperarc.

Fig. 4: A structured discriminator connector and a possible representation of its temporal aspects.

hyperarc A that connects the node representing the beginning instant of the discriminator activity to all nodes representing the end instant of the activities that precede the considered discriminator and are directly connected to it. In general a multi-head hyperarc is defined as a set of distance constraints (components) between one time point and some end points. We say that a multi-head hyperarc is satisfied if at least one of its components is satisfied. In Fig. 4b, dashed arcs define the hyperarc A that is satisfied if $B_{+_{15}}$ is at least 2 distant from E_{T_1} or 1 distant from E_{T_2} or 5 distant from E_{T_3}. It is sufficient that one of such previous nodes is executed and that the delay represented in the corresponding connecting arc is passed to execute $B_{+_{15}}$, as required by the structured discriminator connector semantics.

HyTNs are not only suitable for better representing temporal constraints originating from temporal workflow, but also for better representing more general temporal constraint networks like Conditional Simple Temporal Network [45].

A Conditional Simple Temporal Network (CSTN) is an enriched graph for representing and reasoning about temporal constraints in domains where some constraints may apply only in certain condition settings (scenarios). Each *condition* in a CSTN is represented by a propositional letter whose truth value is *observed* in real time as the outcome of the execution of an *observation time-point*. An execution strategy for a CSTN has to determine an execution time for each time-point guaranteeing that all temporal constrains that are significant in the resulting scenario are satisfied. An execution strategy can be dynamic in that its execution decisions can react to the information obtained from such observations. The Conditional Simple Temporal Problem (CSTP) consists in determining whether a given CSTN admits a dynamic execution strategy for any possible combination of propositional outcomes happens to be observed over time. If such a strategy exists, the CSTN is said to be dynamically consistent (DC).

Tsamardinos et al. [45] solved the CSTP by first encoding it as a meta-level Disjunctive Temporal Problem (DTP), then feeding it to an off-the-shelf DTP

solver. Although of theoretical interest, this approach is not practical because the CSTP-to-DTP encoding has exponential size, and the DTP solver itself runs in exponential time. To our knowledge, this approach has never been empirically evaluated [26].

In [16], Comin and Rizzi proposed a novel representation of CSTNs in terms of HyTNs allowing the determination of the first singly exponential-time algorithm for checking the dynamic consistency of Conditional Simple Temporal Networks. More precisely, a CSTN instance is represented as a suitable HyTN where each possible scenario is represented and connected to other scenarios in an appropriate way and, then, such HyTN instance is solved in pseudo-polynomial time by the algorithms analyzed in the present paper.

In summary, HyTNs allow the representation of temporal constraints that are more general of those represented in STNs [18], because they allow disjunctions involving more than two time points, but less general than those represented in DTPs [44] because all disjunctions related to a multi-head(tail) hyperarcs have to contain a common variable. Such kind of STN generalization not only allows the compact representation of some common temporal constraints in the domains like the workflow-based process management but also allows the determination of new interesting algorithm for checking dynamic-consistency in richer models like CSTN.

3 Background and Notation

In this section, we introduce some definitions, notations and well-know results about graphs and conservative graphs; moreover, we recall the relation between the consistency property of STNs and the conservative property of weighted graphs.

We consider graphs that are directed and weighted on the arcs. Thus, if $G = (V, A)$ is a graph, then every arc $a \in A$ is a triple (t_a, h_a, w_a) where $t_a \in V$ is the *tail* of a, $h_a \in V$ is the *head* of a, and $w_a \in \mathbf{R}$ is the *weight* of a. Moreover, since we use graphs to represent distance constraints, they do not need to have either loops (unary constraints are meaningless) or parallel arcs (two parallel constraints represent two different distance constraints between the same pair of node: only the most restrictive is meaningful). We also use the notations $h(a)$ for h_a, $t(a)$ for t_a, and $w(a)$ or $w(t_a, h_a)$ for w_a, when it helps. The *order* and *size* of a graph $G = (V, A)$ are denoted by $n \triangleq |V|$ and $m \triangleq |A|$, respectively. The size is a good measure for the encoding length of G.

A *cycle* of G is a set of arcs $C \subseteq A$ cyclically sequenced as $a_0, a_1, \ldots, a_{\ell-1}$ so that $h(a_i) = t(a_j)$ if and only if $j = (i + 1) \mod \ell$; it is called a *negative cycle* if $w(C) \leq 0$, where $w(C)$ stands for $\sum_{e \in C} w_e$. A graph is called *conservative* when it contains no negative cycle.

A *potential* is a function $p : V \mapsto \mathbf{R}$. The *reduced weight* of an arc $a = (u, v, w_a)$ with respect to a potential p is defined as $w_a^p \triangleq w_a - p_v + p_u$. A potential p of $G = (V, A)$ is called *feasible* if $w_a^p \geq 0$ for every $a \in A$. Notice that, for any cycle C, $w^p(C) = w(C)$. Therefore, the existence of a feasible potential implies that the graph is conservative as $w(C) = w^p(C) \geq 0$ for every cycle C.

The Bellman-Ford algorithm [17] can be used to produce in $O(nm)$ time:

- either a proof that G is conservative in the form of a feasible potential function;

– or a proof that G is not conservative in the form of a negative cycle C in G.

When the graph is conservative, the smallest weight of a walk between two nodes is well defined, and, fixed a root node r in G, the potentials returned by the Bellman-Ford algorithm are, for each node v, the smallest weight of a walk from r to v. Moreover, if all the arc weights are integral, then these potentials are integral as well. Therefore, the Bellman-Ford algorithm provides a proof to the following theorem.

Theorem 1 ([3, 17, 22]). *A graph admits a feasible potential if and only if it is conservative. Moreover, when all arc weights are integral, the feasible potential is an integral function.*

An STN can be viewed as a weighted graph whose nodes are timepoints that must be placed on the real line and whose arcs express mutual constraints on the allocations of their end points. An STN $G = (V, A)$ is called *consistent* if it admits a *feasible scheduling*, i.e., a scheduling $s : V \mapsto \mathbf{R}$ such that

$$s(v) \leq s(u) + w(u, v) \quad \forall \text{ arc } (u, v) \text{ of } G.$$

Corollary 1 ([3,17,18]). *An STN G is consistent if and only if G is conservative.*

Proof. A feasible scheduling is just a feasible potential. Therefore, this corollary is just a restatement of Theorem 1. □

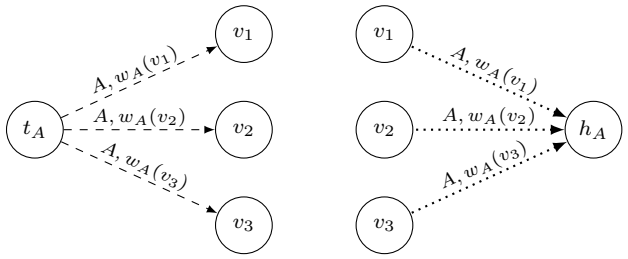
In this paper, we also deal with directed weighted hypergraphs.

Definition 1 (Hypergraph). A *hypergraph* $\mathcal{H}$ is a pair $(V, \mathcal{A})$, where V is the set of nodes, and $\mathcal{A}$ is the set of *hyperarcs*. Each hyperarc $A \in \mathcal{A}$ is either a *multi-head* or a *multi-tail* hyperarc.

A *multi-head* hyperarc $A = (t_A, H_A, w_A)$ has a distinguished node t_A, called the *tail* of A, and a nonempty set $H_A \subseteq V \setminus \{t_A\}$ containing the *heads* of A; to each head $v \in H_A$ is associated a *weight* $w_A(v) \in \mathbf{R}$. Fig. 5a depicts a possible representation of a multi-head hyperarc: the tail is connected to each head by a dashed arc labeled by the name of the hyperarc and the weight associated to the considered head.

A *multi-tail* hyperarc $A = (T_A, h_A, w_A)$ has a distinguished node h_A, called the *head* of A, and a nonempty set $T_A \subseteq V \setminus \{h_A\}$ containing the *tails* of A; to each tail $v \in T_A$ is associated a *weight* $w_A(v) \in \mathbf{R}$. Fig. 5b depicts a possible representation of a multi-tail hyperarc: the head is connected to each tail by a dotted arc labeled by the name of the hyperarc and the weight associated to the considered tail.

The *cardinality* of a hyperarc $A \in \mathcal{A}$ is given by $|A| \triangleq |H_A \cup \{t_A\}|$ if A is multi-head, and $|A| \triangleq |T_A \cup \{h_A\}|$ if A is multi-tail; if $|A| = 2$, then $A = (u, v, w)$ is a standard arc. The *order* and *size* of a hypergraph $(V, \mathcal{A})$ are denoted by $n \triangleq |V|$ and $m \triangleq \sum_{A \in \mathcal{A}} |A|$, respectively.



(a) Multi-Head Hyperarc $A = (t_A, H_A, w_A)$.

(b) Multi-Tail Hyperarc $A = (T_A, h_A, w_A)$.

Fig. 5: A graphical representation of the two kinds of hyperarcs.

4 Hyper Temporal Networks and Consistency Property

We introduce now *Hyper Temporal Networks* (HyTNs), a strict generalization of STNs to partially overcome the limitation of allowing only conjunctions of constraints. Compared to STN distance graphs, which they naturally extend, HyTNs allow a greater flexibility in the definition of temporal constraints.

A HyTN is a directed weighted hypergraph $\mathcal{H} = (V, \mathcal{A})$ where a node represents a time point variable (timepoint), and a multi-head/multi-tail hyperarc represents a set of temporal distance constraints between the tail/head and the heads/tails, respectively.

For example, the multi-tail hyperarc $A = (T_A, B_{+_2}, w_A)$ in Fig. 3, where $T_A = \{b_1, b_2, b_3\}$ and $w_A(b_i) = 0$ for $i = 1, 2, 3$, stands for the set of distance constraints $\{B_{+_2} - b_i \leq 0 \mid i = 1, 2, 3\}$.

In general, we say that a hyperarc is *satisfied* when at least one of its distance constraints is satisfied. Then, we say that a HyTN is *consistent* when it is possible to assign a value to each time-point variable so that all of its hyperarcs are satisfied.

More formally, in the HyTN framework the consistency problem is defined as the following decision problem.

Definition 2 (General-HyTN-Consistency)**.** Given a HyTN $\mathcal{H} = (V, \mathcal{A})$, decide whether there exists a scheduling $s : V \to \mathbf{R}$ such that, for every hyperarc $A \in \mathcal{A}$, the following holds:

- if $A = (t, h, w)$ is a standard arc, then

$$s(h) - s(t) \leq w;$$

- if $A = (t_A, H_A, w_A)$ is a multi-head hyperarc, then

$$s(t_A) \geq \min_{v \in H_A} \{s(v) - w_A(v)\};$$

- if $A = (T_A, h_A, w_A)$ is a multi-tail hyperarc, then

$$s(h_A) \leq \max_{v \in T_A} \{s(v) + w_A(v)\}.$$

Any such scheduling $s : V \to \mathbf{R}$ is called *feasible*. A HyTN that admits at least one feasible scheduling is called *consistent*.

Comparing the consistency of HyTNs with the consistency of STNs, the most important aspect of novelty is that, while in a distance graph of a STN each arc represents a distance constraint and all such constraints have to be satisfied by a feasible schedule, in a HyTN each hyperarc represents a set of one or more distance constraints and a feasible scheduling has to satisfy at least one such distance constraints for each hyperarc.

Let us show some interesting properties about the consistency problem for HyTNs.

The first interesting property is that any integral-weighted HyTN admits an integral feasible schedule when it is consistent, as proved in the following lemma.

Lemma 1. *Let $\mathcal{H} = (V, \mathcal{A})$ be an integral-weighted and consistent HyTN. Then $\mathcal{H}$ admits an integral feasible scheduling $s : V \to \{-T, -T+1, \ldots, T-1, T\}$, where $T = \sum_{A \in \mathcal{A}, v \in V} |w_A(v)|$.*

Proof. Since $\mathcal{H}$ is consistent, then there exists a feasible scheduling $\tilde{s} : V \to \mathbf{R}$. The idea in this proof is to project the HyTN $\mathcal{H}$ over a conservative graph and then, in that setting, to exploit the integrality properties of potentials as stated in Theorem 1. However, this projection is asked to resolve the non-determinism contained in the disjunctive nature of the hyperarcs; in order to sort out such non-determinism, the projection is built considering the given feasible scheduling $\tilde{s}$ as follows.

For each hyperarc $A \in \mathcal{A}$, a weighted directed arc e_A is defined as follows:

- if $A = (u, v, w)$ is a standard arc, then $e_A \triangleq (u, v, w)$. Note that $\tilde{s}(v) \leq \tilde{s}(u) + w$ follows by the feasibility of $\tilde{s}$;
- if $A = (t_A, H_A, w_A)$ is a multi-head hyperarc, then

$$e_A \triangleq (t_A, v_A, w_A(v)) \text{ where } v_A = \arg\min_{v \in H_A} \{\tilde{s}(v) - w_A(v)\}.$$

  Here, $\tilde{s}(v_A) \leq \tilde{s}(t_A) + w_A(v)$ follows by the feasibility of $\tilde{s}$;
- if $A = (T_A, h_A, w_A)$ is a multi-tail hyperarc, then

$$e_A \triangleq (v_A, h_A, w_A(v)) \text{ where } v_A = \arg\max_{v \in T_A} \{\tilde{s}(v) + w_A(v)\}.$$

  Here, $\tilde{s}(h_A) \leq \tilde{s}(v_A) + w_A(v)$ follows by the feasibility of $\tilde{s}$.

Now, a weighted directed graph $G = (V, E)$ with $E \triangleq \{e_A \mid A \in \mathcal{A}\}$ is defined. G is integral-weighted and conservative graph since it admits $\tilde{s}$ as a potential function. Therefore, G admits an integral potential function $s : V \to \{-T, -T+1, \ldots, T-1, T\}$. Indeed, such a function s is obtained by applying the Bellman-Ford algorithm on G. To conclude, we observe that s is also an integral feasible scheduling for $\mathcal{H}$. □

The following theorem states that General-HyTN-Consistency is NP-complete.

Theorem 2. General-HyTN-Consistency *is an NP-complete problem even if input instances $\mathcal{H} = (V, \mathcal{A})$ are restricted to satisfy $w_A(\cdot) \in \{-1, 0, 1\}$ and $|H_A|, |T_A| \leq 2$ for every $A \in \mathcal{A}$.*

Proof. If $\mathcal{H} = (V, \mathcal{A})$ is integral-weighted and consistent, then it admits an integral feasible scheduling $s : V \to \{-T, \ldots, T\}$ by Lemma 1. Moreover, any such feasible scheduling can be verified in polynomial time with respect to the size of the input; hence, General-HyTN-Consistency is in NP.

To show that the problem is NP-hard, we describe a reduction from 3-SAT.

Let us consider a boolean 3-CNF formula with $n \geq 1$ variables and $m \geq 1$ clauses:

$$\varphi(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m} (\alpha_i \vee \beta_i \vee \gamma_i)$$

where $\mathcal{C}_i = (\alpha_i \vee \beta_i \vee \gamma_i)$ is the i-th clause of φ and each $\alpha_i, \beta_i, \gamma_i \in \{x_j, \overline{x}_j \mid 1 \leq j \leq n\}$ is either a positive or a negative literal.

We associate to φ a HyTN $\mathcal{H}_\varphi = (V, \mathcal{A})$, where each boolean variable x_i occurring in φ is represented by two nodes, x_i and $\overline{x}_i$. V also contains node z that represents the reference initial node for the HyTN $\mathcal{H}_\varphi$, i.e., the first node that has to be executed. For each pair x_i and $\overline{x}_i$, $\mathcal{H}_\varphi$ contains a pair of hyperarc constraints as depicted in Fig. 6a: one with multi-head $\{x_i, \overline{x}_i\}$ and tail in z and the other multi-tail $\{x_i, \overline{x}_i\}$ and head in z. If $\mathcal{H}_\varphi$ is consistent, the pair of hyperarcs associated to $x, \neg x$ assures that $\mathcal{H}_\varphi$ admits a feasible scheduling s such that $s(x_i)$ and $s(\overline{x}_i)$ are coherently set with values in $\{0, 1\}$ (see Lemma 1). In this way, s is forced to encode a truth assignment on the x_i's. The HyTN $\mathcal{H}_\varphi$ contains also a node $\mathcal{C}_j$ for each clause $\mathcal{C}_j$ of φ; each node $\mathcal{C}_j$ is connected by a multi-tail hyperarc with head in $\mathcal{C}_j$ and tails over the literals occurring in $\mathcal{C}_j$ and by two standard and opposite arcs with node z as displayed in Fig. 6b. Such setting of arcs assures that if $\mathcal{H}_\varphi$ admits a feasible scheduling s, then s assigns value 1 at least to one of the node representing the literals connected with the hyperarc.

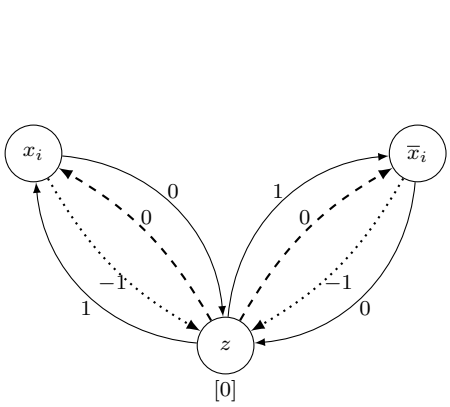
More formally, $\mathcal{H}_\varphi = (V, \mathcal{A})$ is defined as follows:

- $V = \{z\} \cup \{x_i \mid 1 \leq i \leq n\} \cup \{\overline{x}_i \mid 1 \leq i \leq n\} \cup \{\mathcal{C}_j \mid 1 \leq j \leq m\}$;
- $\mathcal{A} = \bigcup_{i=1}^{n} \mathrm{Var}_i \cup \bigcup_{j=1}^{m} \mathrm{Cla}_j$, where:
  - $\mathrm{Var}_i = \Big\{(z, x_i, 1), (x_i, z, 0), (z, \overline{x}_i, 1), (\overline{x}_i, z, 0),$
    $(\{x_i, \overline{x}_i\}, z, [w(x_i), w(\overline{x}_i)] = [-1, -1]),$
    $(z, \{x_i, \overline{x}_i\}, [w(x_i), w(\overline{x}_i)] = [0, 0])\Big\}.$
    This defines the variable gadget for x_i as depicted in Fig. 6a;
  - $\mathrm{Cla}_j = \Big\{(z, \mathcal{C}_j, 1), (\mathcal{C}_j, z, -1),$
    $(\{\alpha_j, \beta_j, \gamma_j\}, \mathcal{C}_j, [w(\alpha_j), w(\beta_j), w(\gamma_j)] = [0, 0, 0])\Big\}.$
    This defines the clause gadget for clause $\mathcal{C}_j = (\alpha_i \vee \beta_i \vee \gamma_i)$ as depicted in Fig. 6b.

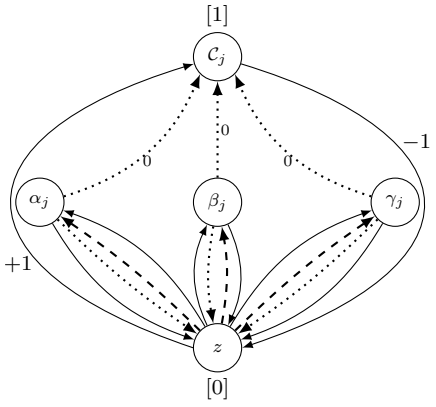
Notice that $|V| = 1 + 2n + m = O(m + n)$ and $m_{\mathcal{A}} = 8n + 5m = O(m + n)$; therefore, the transformation is linearly bounded in time and space.

We next show that φ is satisfiable if and only if $\mathcal{H}_\varphi$ is consistent.

Any truth assignment $\nu : \{x_1, \ldots, x_n\} \to \{\texttt{true}, \texttt{false}\}$ satisfying φ can be translated into a feasible scheduling $s : V \to \mathbf{Z}$ of $\mathcal{H}_\varphi$ as follows. For node z, let $s(z) = 0$, and let $s(\mathcal{C}_j) = 1$ for each $j = 1, \ldots, m$; then, for each $i = 1, \ldots, n$, let $s(x_i) = 1$ and $s(\overline{x}_i) = 0$ if the truth value of x_i, $\nu(x_i)$, is `true`, otherwise let $s(x_i) = 0$ and $s(\overline{x}_i) = 1$. It is easy to verify that, using this scheduling s, all the



(a) Gadget for a 3-SAT variable x_i.



(b) Gadget for a 3-SAT clause $\mathcal{C}_j = (\alpha_j \vee \beta_j \vee \gamma_j)$ where each $\alpha_j, \beta_j, \gamma_j$ is a positive or negative literal.

Fig. 6: Gadgets used in the reduction from 3-SAT to General-HyTN-Consistency.

constraints comprising each single gadget are satisfied and, therefore, the network is consistent.

Vice versa, assume that $\mathcal{H}_\varphi$ is consistent. Then, it admits an integral feasible scheduling s by Lemma 1. After the translation $s(v) = s(v) - s(z)$, we can assume that $s(z) = 0$. Hence, $s(\mathcal{C}_j) = 1$ for each $j = 1, \ldots, m$, as enforced by the two standard arcs incident at $\mathcal{C}_j$ in the clause gadget, and $\{s(x_i), s(\overline{x}_i)\} = \{0, 1\}$ for each $i = 1, \ldots, n$, as enforced by the constraints comprising the variable gadgets. Therefore, the feasible scheduling s can be translated into a truth assignment $\nu : \{x_1, \ldots, x_n\} \to \{\texttt{true}, \texttt{false}\}$ defined by $\nu(x_i) = \texttt{true}$ if $s(x_i) = 1$ (and $s(\overline{x}_i) = 0$); $\nu(x_i) = \texttt{false}$ if $s(x_i) = 0$ (and $s(\overline{x}_i) = 1$) for every $i = 1, \ldots, n$.

To conclude, we observe that any hyperarc $A \in \mathcal{A}$ of $\mathcal{H}_\varphi$ has weights $w_A(\cdot) \in \{-1, 0, 1\}$ and size $|A| \leq 3$. Since any hyperarc with three heads (tails) can be replaced by two hyperarcs each having at most two heads (tails), the consistency problem remains NP-Complete even if $w_A(\cdot) \in \{-1, 0, 1\}$ and $|A| \leq 2$ for every $A \in \mathcal{A}$. □

Theorem 2 motivates the study of consistency problems on hypergraphs having either only multi-head or only multi-tail hyperarcs. In the former case, the consistency problem is called Head-HyTN-Consistency, while in the latter it is called Tail-HyTN-Consistency. In the following theorem we observe that the two problems are inter-reducible, i.e., we can solve consistency for any one of the two models in $f(m, n, W)$ time whenever we have a $f(m, n, W)$ time procedure for solving consistency for the other one.

Theorem 3. Head-HyTN-Consistency *and* Tail-HyTN-Consistency *are inter-reducible by means of* log*-space, linear-time, local-replacement reductions.*

Proof. We show the reduction from multi-tail to multi-head hypergraphs; the other direction is symmetric. Informally, what we will do is to reverse all the arcs (so

that what was multi-tail becomes multi-head), and, contextually, we invert the time-axis (to account for the inversion of the direction of all arcs).

Let $\mathcal{H} = (V, \mathcal{A})$ be a multi-tail hypergraph, we associate to $\mathcal{H}$ a multi-head hypergraph $\mathcal{H}' = (V, \mathcal{A}')$ by reversing all multi-tail hyperarcs. Formally, we define

$$\mathcal{A}' = \{(v, S, w) \mid (S, v, w) \in \mathcal{A}\}.$$

For example, a standard arc $(t, h, w) \in \mathcal{A}$ is transformed into a reversed standard arc (h, t, w) in $\mathcal{A}'$ while a hyperarc with two weighted tails t_1 and t_2 becomes a hyperarc having t_1 and t_2 as its two weighted heads.

Now, $\mathcal{H}$ is consistent if and only if $\mathcal{H}'$ is consistent. To prove it, we note that each scheduling s for $\mathcal{H}$ can be associated, with a flip of the time direction, to the scheduling $s' \triangleq -s$. Then, it holds that s is feasible for $\mathcal{H}$ if and only if s' is feasible for $\mathcal{H}'$. Indeed, s satisfies the constraint represented by an hyperarc $A = (T_A, h_A, w_A) \in \mathcal{A}$, namely

$$s(h_A) \leq \max_{v \in T_A} \{s(v) + w_A(v)\},$$

or, equivalently

$$-s(h_A) \geq \min_{v \in T_A} \{-s(v) - w_A(v)\},$$

if and only if s' (that is, $-s$) satisfies the constraint represented by the reversed hyperarc $A' = (h_A, T_A, w_A)$, namely

$$s'(h_A) \geq \min_{v \in T_A} \{s'(v) - w_{A'}(v)\}.$$

□

In the remainder of this work we shall adopt the multi-head hypergraph as our reference model. Hence, when considering hypergraphs and HyTNs, we will be implicitly referring to multi-head hyperarcs. Notably, we consider the following specialized notion of consistency for HyTNs.

Definition 3 (HyTN-Consistency)**.** Given a (multi-head) HyTN $\mathcal{H} = (V, \mathcal{A})$, decide whether there exists a scheduling $s : V \to \mathbf{R}$ such that:

$$s(t_A) \geq \min_{v \in H_A} \{s(v) - w_A(v)\} \quad \forall A \in \mathcal{A}. \tag{1}$$

Remark 1. Notice that this notion of consistency for HyTNs is a strict generalization of STN one. In general, the feasible schedules of an STN are the solutions of a linear system and, therefore, they form a convex polytope. Since an STN may be viewed as a HyTN, the space of feasible schedules of an STN can always be described as the space of feasible schedules of a HyTN. The converse is not true because feasible schedules for a HyTN do not form a convex polytope. Let us consider, for example, a HyTN of just three nodes x_1, x_2, x_3 and one single hyperarc with heads $\{x_1, x_2\}$ and tail x_3 expressing the constraint $x_3 \geq \min\{x_1, x_2\}$; $(0, 2, 2)$ and $(-2, 0, 2)$ are both admissible schedules, but $(1, 1, 0) = \frac{1}{2}(0, 2, 2) - \frac{1}{2}(-2, 0, 2)$ is not an admissible schedule. In conclusion, the STN model is a special case of the Linear Programming paradigm, whereas the HyTN model is not.

In the rest of this section, we extend the characterization of STN consistency recalled in Section 3 to HyTNs.

Definition 4 (Reduced Slack Value $w_A^p(v)$)**.** With reference to a potential $p : V \to \mathbf{R}$, we define, for every arc $A \in \mathcal{A}$ and every $v \in H_A$, the *reduced slack value* $w_A^p(v)$ as $w_A(v) + p(t_A) - p(v)$ and the *reduced slack* w_A^p as

$$w_A^p \triangleq \max\{w_A^p(v) \mid v \in H_A\}.$$

A potential p is said to be *feasible* if and only if $w_A^p \geq 0$ for every $A \in \mathcal{A}$.

Again, as it was the case for STNs, a mapping $f : V \to \mathbf{R}$ is a feasible potential if and only if it is a feasible schedule. In order to better characterize feasible schedules, we introduce a notion of *negative cycle*.

Definition 5 (Negative Cycle)**.** Given a multi-head HyTN $\mathcal{H} = (V, \mathcal{A})$, a *cycle* is a pair $(S, \mathcal{C})$ with $S \subseteq V$ and $\mathcal{C} \subseteq \mathcal{A}$ such that:

1. $S = \bigcup_{A \in \mathcal{C}} (H_A \cup \{t_A\})$ and $S \neq \emptyset$;
2. $\forall v \in S$ there exists an unique $A \in \mathcal{C}$ such that $t_A = v$.

Moreover, we let $a(v)$ denote the unique arc $A \in \mathcal{C}$ with $t_A = v$ as required in previous item 2. Every infinite path in a cycle $(S, \mathcal{C})$ contains, at least, one *finite cyclic sequence* $v_i, v_{i+1}, \ldots, v_{i+p}$, where $v_{i+p} = v_i$ is the only repeated node in the sequence. A cycle $(S, \mathcal{C})$ is *negative* if for any finite cyclic sequence $v_1, v_2, \ldots, v_p$, it holds that

$$\sum_{t=1}^{p-1} w_{a(v_t)}(v_{t+1}) < 0.$$

There are two results about negative cycles as stated in the following lemmas.

Lemma 2. *A HyTN with a negative cycle admits no feasible schedule.*

Proof. By contraposition. Let $\mathcal{H}$ be a consistent HyTN and let p be a feasible potential for $\mathcal{H}$. Also, let $(S, \mathcal{C})$ be any cycle of $\mathcal{H}$; we will show that $(S, \mathcal{C})$ is not negative. For every $A \in \mathcal{C}$, let h_A be the head of A with maximum reduced slack value:

$$h_A \triangleq \arg\max_{v \in H_A} \{w_A^p(v)\}.$$

Let us consider the infinite path in $(S, \mathcal{C})$ built choosing, at each node v_t, $h_{a(v_t)}$ as the following node. As already seen, such a path contains at least one finite cyclic sequence $v_h, v_{h+1}, \ldots, v_k$ with $v_k = v_h$. The sum of weights of the finite cyclic sequence is given by

$$\sum_{t=h}^{k-1} w_{a(v_t)}(v_{t+1}) = \sum_{t=h}^{k-1} w_{a(v_t)}^p(v_{t+1})$$

for every potential p; since p is feasible, all terms of the last sum are non-negative. It follows that $(S, \mathcal{C})$ is not negative. □

At first sight, it may appear that checking whether $(S, \mathcal{C})$ is a negative cycle might take exponential time since one should check a possibly exponential number of cyclic sequences. The next lemma shows instead that it is possible to check the presence of negative cycle in polynomial time.

Lemma 3. *Let $(S, \mathcal{C})$ be a cycle in a HyTN. Then checking whether $(S, \mathcal{C})$ is a negative cycle can be done in polynomial time.*

Proof. Consider the weighted graph $G = (S, \cup_{t \in S} A_t)$ where each hyperarc $a(t)$, for every $t \in S$, is transformed into a set of standard arcs as follows:

$$a(t) \rightsquigarrow A_t \triangleq \{(t, v, -w_{a(t)}(v)) \mid v \in H_{a(t)})\}, \ \forall t \in S.$$

Notice that G is thus an STN. Checking whether $(S, \mathcal{C})$ is a negative cycle amounts to check whether all cycles in G have strictly positive weight. To do this, firstly, a potential function π for G is determined by Bellman-Ford algorithm. If the algorithm returns a negative cycle instead of π, then there is *no* negative cycle in $(S, \mathcal{C})$ and the check ends.

Otherwise, since $w(C) = w^\pi(C) \geq 0$ for every cycle C of G, it is necessary to verify that no cycle in G has $w^\pi(C) = 0$. This check can be done by verifying the acyclicity of the subgraph of G comprising only arcs a of G with $w^\pi(a) = 0$. The check that a graph is acyclic can be done in linear time by a depth first visit [17]. □

A hypergraph $\mathcal{H}$ is called *conservative* when it contains no negative cycle. In the next sections we will provide a pseudo-polynomial time algorithm that always returns either a feasible scheduling or a negative cycle, thus extending the validity of the classical good-characterization of STN consistency to general HyTN consistency. Here, we anticipate the statement of the main result in order to complete this general introduction of HyTNs.

Theorem 4. *A HyTN $\mathcal{H}$ is consistent if and only if it is conservative. Moreover, when all weights are integral, then $\mathcal{H}$ admits an integral scheduling if and only if it is conservative.*

Proof. If $\mathcal{H}$ is consistent, then it is conservative by Lemma 2. If $\mathcal{H}$ is not consistent, then there is a negative cycle as shown in Theorem 8-(3). The existence of an integral scheduling when all weights are integral is guaranteed by Lemma 1. □

5 Mean Payoff Games

In this section, we propose an introduction to Mean Payoff Games (MPGs) tailored to the needs of the present work. MPGs represent a well-studied model for representing some kinds of two-player dynamics and we will show in Section 6 that there is a substantial equivalence between the MPG and the HyTN model, which will allow us to exploit some important algorithmic and structural results.

An MPG is a weighted directed graph $G = (V_0 \uplus V_1, E)$ whose node set V is partitioned into two disjoint sets V_0 and V_1, where, for $p = 0, 1$, the nodes in V_p are those under control of Player p. Even with these graphs we have no loops and no parallel arcs. It is also assumed that every node has at least one outgoing arc. Notice that, in general, (V_0, V_1) does not need to be a bipartiton of G, i.e., E may contain arcs with both endpoints in V_0, or with both endpoints in V_1.

Each play starts with a pebble placed at some node $v_0 \in V_0 \uplus V_1$ and consists in a sequence of moves. Move t begins with the pebble placed in node v_{t-1} and is played by the Player p such that $v_{t-1} \in V_p$: the player chooses any arc $e \in E$ with tail $t_e = v_{t-1}$ and moves the pebble along e; at the end of the move the pebble is in node $v_t = h_e$. The game ends as soon as $v_t = v_{t'}$ for some $t > t'$, i.e., when the pebble comes back to an already visited node $v_{t'}$. At this point, the pebble has

traversed a cyclic sequence of arcs $e_{t'+1}, \ldots, e_t$ and Player 0 "pays" to Player 1 the average weight of the visited cycle:

$$\frac{1}{t-t'} \sum_{i=t'+1}^{t} w(e_i).$$

If this amount is negative, then Player 0 *wins* the game, otherwise the winner is Player 1.

A *strategy* for Player p is a mapping that, given all the previous visited nodes and the current node, returns which node has to be visited in the next move; a strategy is said to be *positional* (or *memoryless*) if it depends only on the current position v_t and does not take into account all the previous history. If $s \in V_0 \cup V_1$ and Player p has a strategy leading him to win any possible play starting at $v_0 = s$, then we say that s is a *winning start position* for Player p. We denote by W_p the set of winning start positions for Player p. A *winning strategy* for Player p leads Player p to win every play started from any node in W_p. Since these finite games are zero-sum, i.e., what won by a player is what lost by the other one, then they admit a *game value* ν: for each start position $s \in V$ of the game, there exists a $\nu_s \in \mathbf{R}$ such that Player 0 has a strategy ensuring payoff at most ν_s, while Player 1 has a strategy ensuring payoff at least ν_s.

It is worthwhile to consider an infinite variant of the model, in which the game does not stop, and continues for an infinite number of steps. In this model, Player 1 wants to maximize the limit inferior of the average weight:

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} w(v_{t-1}, v_t)$$

Symmetrically, Player 0 wants to minimize the limit superior of the same average weight:

$$\limsup_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} w(v_{t-1}, v_t)$$

In their *Determinacy Theorem*, Ehrenfeucht and Mycielski [21] proved that any infinite game admits a value ν^∞, and that this value equals the one of the finite counterpart game on every start position, i.e., $\nu^\infty_s = \nu_s$ for every $s \in V_0 \cup V_1$. Moreover, they proved the existence of positional strategies which are *optimal* for both variants of the model: when Player p limits himself to an optimal strategy π_p, i.e., when, for every $v \in V_p$, he disregards all arcs with tail in v except the one with head in $\pi_p(v)$, then he will secure himself the optimal payoff ν in every play, finite or infinite, however the adversary plays. The graph G_{π_p} obtained from G by dropping all arcs with tail in V_p not prescribed by π_p is called the *projection* of the game G on π_p, and is a solitaire game whose value can be easily computed by means of a simple variant of Bellman-Ford algorithm. Therefore, the Ehrenfeucht and Mycielski's results are already sufficient for determining a simple exponential time algorithm computing the node values and the two optimal positional strategies in an MPG: the algorithm consists in evaluating each possible strategy for one of the two players as a solitaire game for determining the optimal one. In the literature there are many local search algorithms that explore this space in a more efficient way [5, 6, 41, 42] and some of them have been proven to be practically efficient in

many settings by experiments [6, 41]. Moreover, the global optimization problem of computing the best strategies for one player, according to a given metric, has been shown to have the property that every local optimum is also a global one for many complete metrics [5].

As another line of research, Zwick and Paterson [47] proposed pseudo-polynomial time algorithms for computing values of games, as well as positional optimal strategies. In particular, they considered the following four algorithmic problems:

1. MPG-Decision(ν, s): given a real number ν and a start position s, decide whether $\nu_s \geq \nu$;
2. MPG-Threshold(T): given a real number T, determine for which nodes $s \in V$ it holds that $\nu_s \geq T$;
3. MPG-Values: compute the optimal values ν_s for all $s \in V$.
4. MPG-Synthesis: assuming $\nu_s \geq 0$ ($\nu_s < 0$) for every $s \in V$, *synthesize* a positional winning strategy for Player 1 (Player 0);

and they proved the following theorem:

Theorem 5 ([47]). *Let $G = (V, E)$ be a mean payoff game. Assume all weights are integers and let $W = \max_{e \in E} |w(e)|$. Then the following hold:*

1. MPG-Threshold(T) *can be solved in time $O(|V|^2|E|\,W)$ when $T \in \mathbf{Z}$, whereas it can be solved in time $O(|V|^3|E|\,W)$ when $T \in \mathbf{R}$;*
2. MPG-Values *can be solved in time $O(|V|^3|E|\,W)$;*
3. MPG-Synthesis *can be solved in time $O(|V|^4|E|\log(|E|/|V|)\,W)$.*

Then, they observed that MPG-Decision is the basic decision problem for MPGs in the sense that several natural questions for MPGs, like evaluating the value ν_s for every node s or constructing the optimal positional strategies, may all be Turing-reduced to it. They also pointed out that the existential results of Ehrenfeucht and Mycielski [21] already implies that MPG-Decision $\in \mathsf{NP} \cap$ co-NP and asked whether there might exist a strongly polynomial time decision procedure. Proving the existence of such algorithm is an open problem [7]. Finally, they showed how to reduce mean payoff games to other important families of games on graphs, like discounted payoff games and simple stochastic games.

The complexity status of MPG-Decision has been since updated by proving that it lays in $\mathsf{UP} \cap$ co-UP by Jurdziński in [27].

In recent years, some other interesting results have been proven. Notably, in 2007 Lifshits, Pavlov [33] proposed a *potential theory* for MPGs and in 2011 Brim et al. [7] obtained faster algorithms exploiting results obtained in the the fields of *Energy Games* and *energy progress measures*, which are intimately related to the potentials studied in [33].

Their algorithmic results are summarized in the following theorem.

Theorem 6 ([7]). *For MPGs in which all weights are integers and for $T \in \mathbf{Z}$, the Value Iteration Algorithm [7] solves* MPG-Threshold(T) *and* MPG-Synthesis *in time $O(|V|\,|E|\,W)$, where $W = \max_{e \in E} |w(e)|$.*

We remark that both the algorithm of Paterson and Zwick [47] and the Value Iteration Algorithm [7] prescribe well defined procedures even if the weights on the arcs are real values. What is lost in running these algorithms on real weights is only the pseudo-polynomial upper bound on their running time.

For our purposes, the family of pseudo-polynomial algorithms for MPGs is the best option. Indeed, in most of temporal workflow graphs all weights are expressed by integers of relatively small magnitude with respect to the intrinsic temporal granularity of the considered workflow. For example, in a workflow containing temporal distance constraints of days, the commonly adopted temporal granularity is the "minute" (m) and, therefore, all weights can be assumed to be less than 10^4 as order of magnitude. In such circumstances, Brim's algorithm offers the guarantee to terminate within short computation times. For these reasons we opted for integrating the procedures of Zwick and Paterson, as well as the faster procedures of Brim et al. [7], in order to efficiently solve instances of HyTN-Consistency and compute feasible schedules.

Furthermore, as will be discovered in the experimental section, if these algorithms are suitably adapted—so as to allow them to terminate earlier as soon as certain evidences of inconsistency have been collected—then their observed behavior outperforms by orders of magnitude what predicted by their theoretical pseudo-polynomial bounds even on input instances containing very large integer values.

Based on these findings, we think that these pseudo-polynomial algorithms are to be considered (and probably adopted) even for solving HyTN instances where weights are floating point values whose magnitudes may differ in a significant way. In case the running time results to be unacceptable for a real application, one could then consider the possibility to round the weights to integer values. This rounding would clearly require special care: a very accurate approximation might lead to very high computation times while a gross approximation might not represent the original instance in a correct way.

6 The Reductions

This section presents the direct connection and the computational equivalence between MPG-Threshold and HyTN-Consistency. The equivalence is formally proven by offering one reduction in each direction.

The reduction of HyTN-Consistency to MPG-Threshold allows to apply, in the context of HyTNs, any of the algorithms known for MPGs, included the exponential and subexponential ones.

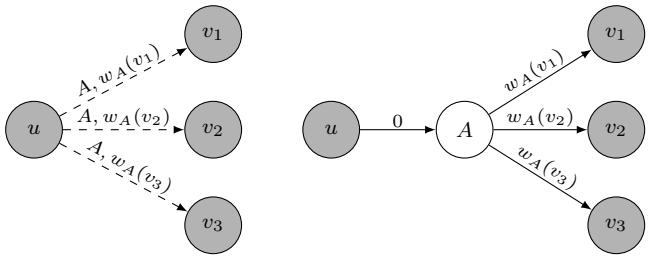
Vice versa, in consideration of the fact that the MPG-Decision $\stackrel{?}{\in} \mathsf{P}$ question is an open problem [5, 7, 27, 41, 47], the reduction of MPG-Decision to HyTN-Consistency confirms that HyTN-Consistency offers an algorithmically more ambitious and mathematically steeper generalization of STN-Consistency (see also Remark 1). Moreover, the reduction gives a further evidence that, within STNs, a new algorithmic approach is necessary in order to manage temporal aspects of event like the synchronization one presented in the Introduction.

Let us start considering the first reduction.

Theorem 7. *There exists a* log*-space*[2]*, linear-time, local-replacement*[3] *reduction from* HyTN-Consistency *to* MPG-Threshold.

[2] A strong and basic-form of reduction introduced by Papadimitriou in [38].

[3] A restricted kind of Karp reduction introduced in [24].



(a) Hyperarc A. (b) MPG representation of A.

Fig. 7: The conversion of a hyperarc into a white MPG node and its incident arcs.

Since this reduction plays a main role in the algorithmic solutions proposed in this paper, we firstly describe how it works and, secondly, we prove its correctness by means of two lemmas, Lemma 4 and Lemma 5.

The reduction goes as follows.

Let $\mathcal{H} = (V, \mathcal{A})$ be a HyTN. We assume that every $v \in V$ is the tail of some arc $A \in \mathcal{A}$. This assumption is not a restriction since, if $\mathcal{H}$ contains a *sink node* v, i.e., a node v with no arc $A \in \mathcal{A}$ having tail in it, then $\mathcal{H}$ is consistent if and only if so is $\mathcal{H}_v$, the HyTN obtained from $\mathcal{H}$ by removing node v and every hyperarc having v as an head. Indeed, any feasible scheduling $s : V \mapsto \mathbf{R}$ for $\mathcal{H}$, once projected onto $V \setminus \{v\}$, gives a feasible scheduling for $\mathcal{H}_v$ since every constraint involving v has been dropped and no constraint has been added; conversely, any feasible scheduling s for $\mathcal{H}_v$ can be easily extended to a feasible scheduling s for $\mathcal{H}$ by exploiting the property of v being a sink node: it is sufficient to set $s(v) \triangleq \min\{s(t_A) - w_A(v) \mid A \in \mathcal{A}, v \in H_A\}$.

Now, let us consider a mean payoff game $G_{\mathcal{H}} = (V_0 \uplus V_1, E)$ where: (1) $V_0 = V$, $V_1 = \mathcal{A}$, nodes in V_0 are colored by *black* while nodes in V_1 are colored by *white*, and (2) for each $A \in \mathcal{A}$, the following weighted arcs are added to E:

- an arc of weight 0 from the black node t_a to the white node A, i.e., arc $(t_A, A, 0)$;
- for each head node $h \in H_A$, an arc of weight $w_A(h)$ from the white node A to the black node h, i.e., arc $(A, h, w_A(h))$.

In short, $G_{\mathcal{H}} = (V_0 \cup V_1, \mathcal{A})$, with $V_0 = V$, $V_1 = \mathcal{A}$, $E = \{(t_A, A, 0) \mid A \in \mathcal{A}\} \cup \{(A, h, w_A(h)) \mid A \in \mathcal{A}, h \in H_A\}$. Fig. 7 depicts how a hyperarc is transformed into a MPG subnetwork while Fig. 8 reports a pseudocode for the whole construction process, i.e., Algorithm 1.

$G_{\mathcal{H}}$ has $|V| + |\mathcal{A}|$ nodes and $O(m)$ arcs and can be constructed in linear time. Moreover, $G_{\mathcal{H}}$ is a bipartite graph with bipartition (V_0, V_1) and it has been obtained from $\mathcal{H}$ by a simple local replacement rule: replace every hyperarc $A \in \mathcal{A}$ by a claw subgraph as depicted in Fig. 7. For each single object, it is necessary only to manage a constant number of indexes, each of them having a polynomial size; thus the reduction is log-space. Fig. 9 depicts an MPG obtained applying the reduction to the motivating example HyTN depicted in Fig. 3;

Now, let us introduce the formal proof of Theorem 7 by the following two lemmas.

Lemma 4. *If $\mathcal{H}$ is consistent then every node of $G_{\mathcal{H}}$ is a winning start position for Player* 1.

Algorithm 1: `makeACorrespondingGame`$(\mathcal{H})$

```
   // a HyTN H = (V, A)
1  V_0 ← V;
2  V_1 ← A;
3  E ← ∅;
4  foreach A ∈ A do
5  |   E ← E ∪ (t_A, A, 0);
6  |   foreach h ∈ H_A do
7  |   |   E ← E ∪ (A, h, w_A(h));
```

Output: The MPG $G_{\mathcal{H}} = (V_0 \uplus V_1, E)$

Fig. 8: The algorithm implementing the reduction from a HyTN to the corresponding MPG.

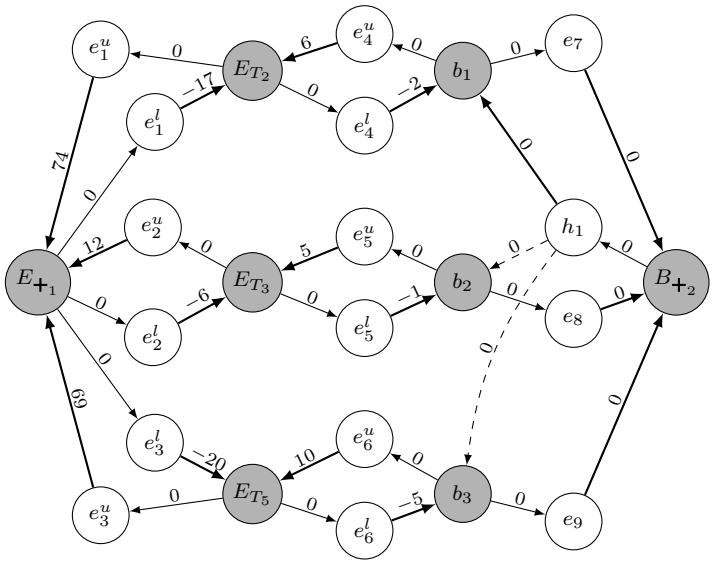


Fig. 9: The MPG equivalent to the HyTN depicted in Fig. 3, obtained by considering the (equivalent) multi-head HyTN transformation of the multi-tail HyTN shown in Fig. 3. A winning positional strategy π_1 for Player 1 is highlighted by thick arcs. The dashed arcs are those not prescribed by strategy π_1, i.e., they are removed when projecting the MPG on π_1.

Proof. Since $\mathcal{H}$ is consistent, there exists a feasible scheduling $s : V \to \mathbf{R}$ such that, for each hyperarc $A \in \mathcal{A}$, the reduced slack weight is non-negative $w_A^s \geq 0$. Consider the following positional strategy π_1 for Player 1: for each $A \in V_1$,

$$\pi_1(A) = \arg \min_{h \in H_A} \{s(h) - w_A(h)\}.$$

We claim that π_1 ensures Player 1 the win, wherever node the game starts from and however Player 0 moves. In order to show this, we prove that the projection graph G_{π_1} is conservative exhibiting a feasible potential p. Let $p : V_0 \cup V_1 \to \mathbf{R}$ be defined as follows:

$$p(v) \triangleq \begin{cases} s(v) & \text{if } v \in V_0, \\ s(t(v)) & \text{if } v \in V_1. \end{cases} \tag{2}$$

Now, let $a = (u, v, w)$ be any arc of G_{π_1}:
Case 1: if $v \in V_1$, then v is a hyperarc of $\mathcal{H}$ with $t(v) = u$ and $w = 0$; therefore, $p(v) = s(t(v)) = s(u) = p(u)$ since $u \in V_0$. Then $w^p(u, v) = w - p(v) + p(u) = 0 \geq 0$ follows;
Case 2: if $v \in V_0$, then $u \in V_1$ and $w = w_u(v)$. Moreover, $v = \pi_1(u)$, which implies that $v = \arg\min_{h \in H_u}\{s(h) - w_u(h)\}$. Therefore, recalling that $w^s_u \geq 0$, i.e., $s(t(u)) \geq \min_{h \in H_u}\{s(h) - w_u(h)\}$:

$$p(u) = s(t(u)) \geq \min_{h \in H_u}\{s(h) - w_u(h)\} = s(v) - w_u(v) = p(v) - w.$$

Hence, $w^p(u, v) = w - p(v) + p(u) \geq 0$.
In conclusion, G_{π_1} is conservative. Therefore, the positional strategy π_1 certifies that any node of G is a winning start position for Player 1. □

Lemma 5. *If every node of $G_{\mathcal{H}}$ is a winning start position for Player* 1 *then $\mathcal{H}$ is a consistent HyTN.*

Proof. If every node is a winning start position for Player 1, then there exists a positional strategy π_1 which is everywhere winning for Player 1. Notice that G_{π_1} must be conservative since Player 0 can clearly win any play starting from a node located on a negative cycle. Let $p : V_0 \cup V_1 \to \mathbf{R}$ be a feasible potential for G_{π_1}. We claim that the restriction of p onto V_0 is a feasible scheduling for $\mathcal{H}$. Indeed, for any hyperarc A of $\mathcal{H}$, $(t_A, A, 0)$ is an arc of G_{π_1}, whence $p(A) \leq p(t_A)$. Moreover, $(A, \pi_1(A), w_A(\pi_1(A)))$ is also an arc of G_{π_1}, whence $p(\pi_1(A)) \leq p(A) + w_A(\pi_1(A))$. Since $\pi_1(A) \in H_A$, then the following holds:

$$\begin{aligned} p(t_A) \geq p(A) &\geq p(\pi_1(A)) - w_A(\pi_1(A)) \\ &\geq \min_{h \in H_A}\{s(h) - w_A(h)\}. \end{aligned}$$

Hence, the restriction of p onto V_0 is a feasible scheduling for $\mathcal{H}$. Thus, $\mathcal{H}$ is consistent. □

In Fig. 10 the values under the nodes represent a feasible potential for the projection of the MPG depicted in Fig. 9. By Lemma 5, the restriction of such a feasible potential on the black nodes is also a feasible scheduling for the corresponding HyTN depicted in Fig. 3. Now, we have all the necessary results to prove the following theorem.

Theorem 8. *Let $\mathcal{H} = (V, \mathcal{A})$ be an integral-weighted HyTN, $m = \sum_{A \in \mathcal{A}} |A|$, and $W = \max_{A \in \mathcal{A}}\{\max_{h \in A} |w_A(h)|\}$ the maximal weight value present in $\mathcal{H}$. The following propositions hold:*

1. *There exists an $O((|V| + |\mathcal{A}|)mW)$ pseudo-polynomial time algorithm deciding* HyTN-Consistency *for $\mathcal{H}$;*
2. *There exists an $O((|V|+|\mathcal{A}|)mW)$ pseudo-polynomial time algorithm such that, given on input any consistent HyTN $\mathcal{H}$, it returns as output a feasible scheduling $s : V_{\mathcal{H}} \to \mathbf{Z}$ of $\mathcal{H}$;*
3. *There exists an $O((|V|+|\mathcal{A}|)mW)$ pseudo-polynomial time algorithm such that, given on input any not-consistent HyTN $\mathcal{H}$, it returns as output a negative cycle $(S, \mathcal{C})$ of $\mathcal{H}$.*

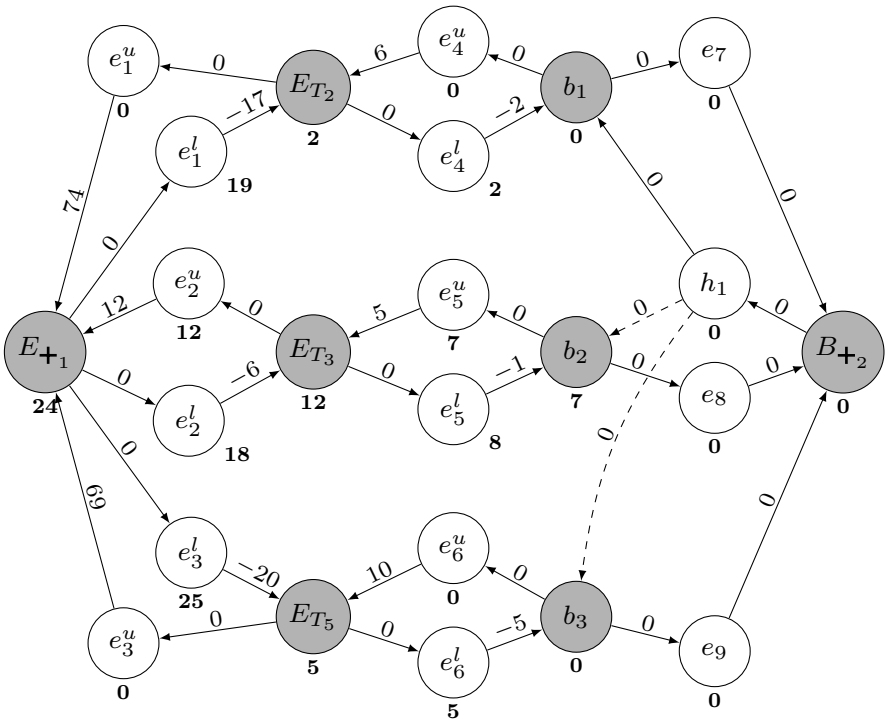


Fig. 10: The integer labels under the nodes are a feasible potential for the projection on π_1 of the MPG depicted in Fig. 9. The restriction of this potential on the black nodes (those in V_0) is a feasible scheduling for the HyTN depicted in Fig. 3 as explained in the proof of Lemma 5.

Algorithm 2: `isConsistent`($\mathcal{H}$)

```
   // a HyTN H = (V, A) of unknown consistency state
1  G_H ← makeACorrespondingGame(H); // See Algorithm 1
2  (W_0, W_1) ← solveMPG-Threshold(G_H, 0); // Brim's algorithm, see Theorem 6
3  if (W_0 = ∅) then Output: YES;
4  else Output: NO;
```

Fig. 11: Pseudocode of the algorithm for deciding HyTN-Consistency.

Proof. 1. The decision algorithm is sketched in Fig. 11. It takes in input a HyTN $\mathcal{H} = (V, \mathcal{A})$ and, in line 1, constructs the corresponding MPG $G_{\mathcal{H}}$ as described in Theorem 7. This first step takes $O(m)$ time and yields a graph with $|V|+|\mathcal{A}|$ nodes and $O(m)$ arcs. Then, in line 2, the instance of MPG-Threshold with $T = 0$ on graph $G_{\mathcal{H}}$ is solved in $O((|V|+|\mathcal{A}|)mW)$ time by the Value Iteration Algorithm (see Theorem 6). The output consists in a partition of $G_{\mathcal{H}}$ nodes into two sets: $W_1 = \{v \in V \cup \mathcal{A} \mid \nu_v \geq 0\}$ and $W_0 = \{v \in V \cup \mathcal{A} \mid \nu_v < 0\}$. If W_0 is empty, then $\mathcal{H}$ is consistent by Lemma 5, otherwise it is not consistent by Lemma 4.

2. In case W_0 is empty, a feasible scheduling is obtained as shown in Algorithm 3. First, in line 2, the algorithm computes a positional winning strategy π_1 for

Algorithm 3: `computeAFeasibleSchedule`$(\mathcal{H})$

// a consistent HyTN $\mathcal{H} = (V, \mathcal{A})$
1 $G \leftarrow$ `makeACorrespondingGame`$(\mathcal{H})$; // See Algorithm 1
2 $\pi_1 \leftarrow$ MPG-Synthesis (G); // Compute a positional winning strategy for Player 1; see Theorem 6
3 $G_{\pi_1} \leftarrow$ compute the subgraph of G induced by π_1;
// Recall $G_{\pi_1} = (V_0 \uplus V_1, E)$, where $V_0 = V$ and $V_1 = \mathcal{A}$.
4 $s \leftarrow$ a new node; // $s \notin V_0 \cup V_1$
5 Add s to V_1 and add an arc $(s, v, 0)$ for each $v \in V_0$;
6 $p \leftarrow$ `Bellman-Ford`(G_{π_1}, s); // compute a potential function p
Output: the restriction of p onto V

Fig. 12: Pseudocode of the algorithm for computing a feasible schedule.

Player 1. This takes $O((|V| + |\mathcal{A}|)mW)$ time by Theorem 6. Next, in line 3, it builds the graph G_{π_1} which is conservative since π_1 is a positional winning strategy for Player 1. Then, in lines 4-5, it adds a new node s to V_1 and a new arc $e_v = (s, v, 0)$ for each node $v \in V_0$ in G_{π_1}. Let $G'_{\pi_1} = (V_0 \uplus (V_1 \cup \{s\}), E')$ the graph thus obtained. Observe that every node of G'_{π_1} is reachable from s. Indeed, every node $A \in V_1 = \mathcal{A}$ can be reached by traversing two arcs: from s to t_A along the arc $e_{t_A} = (s, t_A, 0)$, which belongs to G'_{π_1} as $t_A \in V_0$, then from t_A to A along the arc $(t_A, A, 0)$, which belongs to G_{π_1} (and hence to G'_{π_1}) since $t_A \in V_0$.
Since the added node s is a source, then G'_{π_1} is conservative too. Therefore, in G'_{π_1}, the set of distances from node s, computed calling the Bellman-Ford algorithm in line 6, forms a feasible potential $p : V_0 \cup V_1 \cup \{s\} \to \mathbf{Z}$ and the restriction of p onto $V_0 = V$ is a feasible scheduling for $\mathcal{H}$.

3. In case W_0 is not empty, a negative cycle is determined by Algorithm 4. Let $G[W_0]$ be the subgraph of G induced by W_0, i.e., the graph obtained from G by removing all nodes not in W_0 and all the arcs incident into them. Notice that every node $v \in W_0$ is a winning start position for Player 0 in game $G[W_0]$ because v is a winning start position for Player 0 in game G, and no winning strategy for Player 0 in G can prescribe a move from a node in W_0 to a node in W_1; therefore, that same winning strategy remains valid on $G[W_0]$. This implies that, for every $u \in W_0$, there exists at least one arc (u, v) with $v \in W_0$. In particular, since (V_0, V_1) is a bipartition of G, then $\overline{W}_0 \triangleq W_0 \cap V_0 \neq \emptyset$. In line 3, a positional winning strategy π_0 for Player 0 on $G[W_0]$ is determined. By Theorem 6, this computation takes time $O((|V| + |\mathcal{A}|)mW)$. Consider the set of hyperarcs $\mathcal{C} = \{\pi_0(v)\}_{v \in \overline{W}_0}$; the pair $(\overline{W}_0, \mathcal{C})$ returned by the algorithm is a negative cycle. Indeed, for any $v \in \overline{W}_0$, $\pi_0(v) \in V_1$ is a hyperarc of $\mathcal{H}$. Thus the head set $H_{\pi_0(v)} \subseteq V_0$. Also, $H_{\pi_0(v)} \subseteq W_0$, since v is a winning start position for Player 0 and π_0 is a winning strategy for Player 0. Combining, $H_{\pi_0(v)} \subseteq \overline{W}_0$ determining that $(\overline{W}_0, \mathcal{C})$ is a negative cycle.

□

Remark 2. In Theorem 8 Item 2), a set of feasible potentials may be obtained without executing the Bellman-Ford algorithm. Actually, if the partition (W_0, W_1) is computed by the Value Iteration Algorithm [7], then a feasible scheduling for $\mathcal{H}$ can be directly derived from the *progress measure* computed within the algorithm.

Algorithm 4: `computeANegativeCycle`$(\mathcal{H}, W_0)$

// a HyTN $\mathcal{H} = (V, \mathcal{A}) = (V_0 \uplus V_1, \mathcal{A})$ which is not consistent
// the non-empty set $W_0 = \{v \in V \mid \nu_v < 0\}$
1 $G \leftarrow$ `makeACorrespondingGame`$(\mathcal{H})$; // See Algorithm 1
2 $G[W_0] \leftarrow$ compute the subraph of G induced by W_0;
3 $\pi_0 \leftarrow$ MPG-Synthesis $(G[W_0])$; // Compute a positional winning strategy for Player 0; see Theorem 6
4 $\overline{W}_0 \leftarrow W_0 \cap V_0$;
5 $\mathcal{C} \leftarrow \{\pi_0(v)\}_{v \in \overline{W}_0}$;
Output: $(\overline{W}_0, \mathcal{C})$

Fig. 13: Pseudocode of the algorithm for computing a negative cycle.

Algorithm 5: `computeAFeasibleSchedule-Remark2`(H)

// a consistent HyTN $\mathcal{H} = (V, \mathcal{A}) = (V_0 \uplus V_1, \mathcal{A})$
// ref. Remark 2 and Theorem 6 [7]
1 $G \leftarrow$ `makeACorrespondingGame`$(\mathcal{H})$; // ref. Algorithm 1
2 $f \leftarrow$ `Value-Iteration`(G); // compute an energy progress measure for G as in Theorem 6
Output: f

Fig. 14: Pseudocode of the algorithm of Remark 2 for computing a feasible schedule.

In more detail, let $G = (V_0 \uplus V_1, E)$ be an MPG weighted by $w : E \to \mathbf{Z}$. An *energy progress measure* is a function $f : V_0 \cup V_1 \to \mathbf{N} \cup \{+\infty\}$ such that: if $v \in V_0$, then for every $(v, v', w) \in E$ it holds $f(v) \geq f(v') - w$; otherwise, $v \in V_1$ and there exists $(v, v', w) \in E$ such that $f(v) \geq f(v') - w$. An energy progress measure $f : V_0 \cup V_1 \to \mathbf{N} \cup \{+\infty\}$ such that $0 \leq f(v) < +\infty$ for every $v \in V_0 \cup V_1$ is provided by the resolution algorithm of Theorem 6 in time $O((|V| + |\mathcal{A}|)mW)$.

The progress measure f is already a feasible scheduling for $\mathcal{H}$: in fact, for every hyperarc $A \in \mathcal{A}$, it holds $(t_A, A, 0) \in E$ and $(A, v, w_A(v)) \in E$, for every $v \in H_A$; combining these two last facts, it follows that:

$$f(t_A) \geq f(A) \geq \min_{v \in H_A} \{f(v) - w_A(v)\},$$

i.e., f is a scheduling satisfying all constrains $A \in \mathcal{A}$. This allow us to employ the algorithm depicted in Fig. 14 instead of the one depicted in Fig. 12 in the case that $W_1 = V$.

The computational equivalence between MPG-Decision problem and HyTN-Consistency can be now determined by showing that also MPG-Decision can be reduced to HyTN-Consistency.

Theorem 9. *There exists a* log*-space, linear-time, local-replacement reduction from* MPG-Decision *to* HyTN-Consistency.

Proof. Let $G = (V_0 \uplus V_1, E)$ be an MPG. For each node $u \in V_0 \cup V_1$, let $N_G(u)$ denote the outgoing neighborhood of u in G, i.e., $N_G(u) \triangleq \{v \in V_0 \cup V_1 \mid (u, v) \in E\}$.

A corresponding HyTN $\mathcal{H} = (V, \mathcal{A})$, where $V = V_0 \uplus V_1$, is constructed from G as follows. For every $u \in V_1$, a hyperarc $A_u \in \mathcal{A}$ is added to $\mathcal{H}$, where:

$$A_u \triangleq (u, N_G(u), w_{A_u}),$$

with weight $w_{A_u}(v) \triangleq w(u,v)$ for every $v \in N_G(u)$. Moreover, for every $u \in V_0$ and every $v \in N_G(u)$, a hyperarc $A_{uv} \in \mathcal{A}$ is added to $\mathcal{H}$, where:

$$A_{uv} \triangleq (u, v, w(u,v)).$$

This construction requires a log-space and linear-time computation.

Now, we firstly prove that if $\mathcal{H}$ is consistent then every node of G is a winning start position for Player 1.

Indeed, let $s : V \to \mathbf{R}$ be a feasible scheduling for $\mathcal{H}$. Thus, $w_A^s \geq 0$ for every hyperarc $A \in \mathcal{A}$. Notice that, by construction, for each $u \in V_1$ there exists a unique hyperarc $A_u \in \mathcal{A}$ with tail $t_{A_u} = u$; moreover, it holds that $H_{A_u} \triangleq N_G(u)$. Hence, for each $u \in V_1$, we can define a positional strategy π_1 for Player 1 as follows:

$$\pi_1(u) \triangleq \arg \min_{h \in H_{A_u}} \{s(h) - w_{A_u}(h)\}.$$

Now, consider the potential function $p : V \to \mathbf{R}$ defined as: $p(u) \triangleq s(u)$ for every $u \in V$. We argue that p is a feasible potential for G_{π_1}.

In fact, let $a = (u, v, w) \in E$ be any arc of G:

Case 1: Assume that $u \in V_0$. Then, by construction, $a = A_{uv}$. Hence, from $p \triangleq s$ and the feasibility of s, we have:

$$\begin{aligned} p(u) = s(u) &\geq \min_{h \in H_{A_{uv}}} \{s(h) - w_{A_{uv}}(h)\} \\ &= s(v) - w_{A_{uv}}(v) \\ &= p(v) - w \end{aligned}$$

Hence, $w^p(u,v) = w - p(v) + p(u) \geq 0$;

Case 2: Assume that $u \in V_1$. Then, by construction, $A = (u, N_G(u), w_A)$, where $w_A(z) = w(u,z)$ for every $z \in N_G(u)$; moreover, notice that $v = \pi_1(u) \in N_G(u) = H_A$. Hence, from $p \triangleq s$, the feasibility of s, and the definition of π_1, we have:

$$\begin{aligned} p(u) = s(u) &\geq \min_{h \in H_u} \{s(h) - w_{A_u}(h)\} \\ &= s(\pi_1(u)) - w_{A_u}(\pi_1(u)) \\ &= s(v) - w_{A_u}(v) \\ &= p(v) - w \end{aligned}$$

Hence, $w^p(u,v) = w - p(v) - p(u) \geq 0$.

Thus, G_{π_1} is conservative. This implies that every node of G is a winning start for Player 1.

Secondly, we prove that if every node of G is a winning start position for Player 1, then H is consistent.

Let π_1 be a positional winning strategy for Player 1. It follows that G_{π_1} is conservative and, therefore, it admits a feasible potential $p : V \to \mathbf{R}$. Now, consider the scheduling function $s : V \to \mathbf{R}$ for $\mathcal{H}$ defined as: $s(u) \triangleq p(u)$ for every $u \in V$. We argue that s is a feasible scheduling of $\mathcal{H}$.

In fact, let $A = (t_A, H_A, w_A) \in \mathcal{A}$ be any hyperarc of $\mathcal{H}$:

Case 1: assume $t_A \in V_0$. Then, by construction, $A = (u, v, w)$ for some $v \in N_G(u), w \in \mathbf{R}$ and $u = t_A$. Hence, from $s \triangleq p$ and the feasibility of p, we have:

$$\begin{aligned} s(t_A) = p(u) &\geq p(v) - w \\ &= s(v) - w_A(v) \\ &= \min_{h \in H_A}\{s(h) - w_A(h)\} \end{aligned}$$

Hence, s satisfies A, i.e., $w_A^s \geq 0$;

Case 2: assume $t_A \in V_1$. Then, by construction, $A = (u, N_G(u), w_A)$ for $u = t_A$ and $w_A(v) = w(u, v) \in \mathbf{R}$ for every $v \in N_G(u)$; moreover, if $v \triangleq \pi_1(u)$, then $v \in N_G(u) = H_A$. Hence, from $s \triangleq p$ and the feasibility of p, we have:

$$\begin{aligned} s(t_A) = p(u) &\geq p(v) - w \\ &= s(v) - w_A(v) \\ &\geq \min_{h \in H_A}\{s(h) - w_A(h)\} \end{aligned}$$

Hence, s satisfies A, i.e., $w_A^s \geq 0$.

This proves that s satisfies every hyperarc $A \in \mathcal{A}$. Then s is a feasible scheduling of $\mathcal{H}$, which is thus consistent. □

7 Computational Experiments

This section describes our empirical evaluation of the proposed consistency checking algorithms to evaluate the performances and the general applicability of the proposed HyTN model. Both Algorithm 3 and Algorithm 4 consist of one single call to Algorithm 2, plus some extra computation of lower asymptotic complexity. Since the cost of these further computations was confirmed to be practically negligible in some preliminary experiments, we report on the results of our experimental investigations only for Algorithm 2.

All algorithms and procedures employed in this empirical evaluation have been implemented in C/C++ and executed on a Linux machine having the following characteristics:

- 2 CPU AMD Opteron 4334;
- 64GB RAM;
- Ubuntu server 14.04.1 Operating System.

The source code and all HyTNs used in the experiments are freely available [14].

The main goal of this empirical evaluation was to determine the average computation time of Algorithm 2, with respect to randomly-generated HyTNs following different criteria, in order to give an idea of the practical behavior of the algorithm. According to Theorem 8, the worst-case time complexity of Algorithm 2 is $O((n + m')mW)$, where $n = |V|$, $m' = |\mathcal{A}|$, $m = \sum_{A \in \mathcal{A}} |A|$, and $W = \max_{A \in \mathcal{A}}\{\max_{h \in A} |w_A(h)|\}$. Hence, we implemented different experiments with respect to the parameters n, m', m, and W. Here we propose a summary of the obtained results presenting a brief report about four tests, Test 1, Test 2, Test 3 and Test 4.

In Test 1 the average computation time was determined for different HyTN orders n to emphasize the practical computation time dependency on n. In Test 2

the average computation time was determined for different HyTN maximal edge-weights W to understand how much the practical computation time is dependent on W. In Test 3 we investigated how some execution times affect the value of the standard deviation, with the goal to determine how many instances require a significant greater computation time with respect to the average time of a data set. Finally, in Test 4 the average computation time was determined with respect to different values of the number of possible strategies of Player 1 $\prod_{A \in \mathcal{A}} |H_A|$ in order to give an idea about the possible practical relation between execution time and number of possible strategies.

The generation of random HyTN instances was carried out exploiting two generators. The first generator was the random workflow schema generator provided by ATAPIS toolset [31]: it produces random workflow graphs according to different input parameters that allow to control the minimal and maximal number of activities, probability for having parallel branches, the minimal and maximal probability of inter-task temporal constraints, etc. on the generated graphs. We verified that this tool allows the determination of graphs that are not only a closer approximation to real-world examples, but also more difficult to check than those generated at random without particular criteria.

We generated benchmarks as follows:

1. First, temporal workflow graphs were generated by fixing the probability for parallel branches to 10% and maximal value for each activity duration or delay between activities to a value W, where W was chosen accordingly to the test type;
2. Then, each workflow graph was translated into an equivalent HyTN $\mathcal{H}$ by the simple transformation algorithm exemplified in Section 2.

It is worth noting that different random workflow graphs all having the same number of activities may translate into HyTNs having different orders n because the original workflow graphs may have different number of connector nodes. Considering the transformation algorithm exemplified in Section 2, it is easy to verify that a workflow with N activities can translate into a CSTN having between $2N + 2$ nodes (when the workflow is a simple sequence) and $5N + 2$ nodes (when the workflow is a sequence of groups of two parallel activities).

ATAPIS toolset has been designed to generate graphs with strongly connected components (Andreas Lanz, personal communication, October 6, 2015). In particular, it has been optimized for small graphs with up to 50 activities. This design choice was motivated by the widely accepted seven process modeling guidelines [34] which suggests to always "decompose a (workflow) model with more than 50 elements (activities)". Therefore, we used the tool for generating random workflow graphs with 100 activities at maximum and, consequently, obtaining HyTNs having 502 nodes at most.

In Table 1 we report the orders of the smallest and the largest HyTN determined from each set of random generated workflow graphs having N activities for $N \in \{10, 20, \ldots, 100\}$.

In order to study the scalability of the algorithm with respect to the number of nodes, we had to rely on a second generator of random HyTN graphs. Our choice has been to use the `randomgame` procedure of `pgsolver` suite [39], that can produce parity games instances for any given number of nodes. In particular, we exploited `randomgame` in the following way:

Table 1: Orders of the smallest and biggest HyTN determined for each set of random generated workflows having N activities.

N	Order of smallest HyTN	Order of biggest HyTN
10	26	50
20	48	94
30	78	138
40	104	196
50	136	236
60	164	268
70	196	306
80	222	350
90	262	394
100	292	410

Table 2: Comparison between different kinds of queue implementation in the Value-Iteration procedure. All values are in seconds.

	FIFO Queue	LIFO Queue	LIFO Queue + Stopping-Criterion	Max-Priority Queue
μ	90.55	11.77	6.98	184.69
σ	487.69	64.10	34.61	653.26

1. First, `randomgame` was used to generate random directed graphs with out-degree fixed to 3;
2. Then, the resulting graphs were translated into MPGs by weighting each arc with an integer randomly chosen in the interval $[-W, W]$, where W was chosen accordingly to the test type;
3. Finally, each MPG G was translated into a HyTN $\mathcal{H}_G$ by the reduction algorithm of Theorem 9. During the translation from MPG to HyTN, only 10% of the hyperarcs were maintained having multiple heads, while 90% of hyperarcs were transformed into standard arcs. This 10%-rule stems from the fact that we are considering workflow based applications where the percentage of (multi-headed) hyperarcs is less than 10% compared to standard arcs in general.

With such settings, the resulting HyTNs are characterized by $m, m' \in \Theta(n)$.

Before presenting the summary of results, it is worthwhile to present some implementation choices about Algorithm 2 that we had to adopt. The core of the algorithm consists of calls to algorithms `makeACorrespondingGame(`$\mathcal{H}$`)`, that transforms the given HyTN $\mathcal{H}$ into a MPG $G_{\mathcal{H}}$, and `solveMPG-Threshold(`$G_{\mathcal{H}}, 0$`)` (Value Iteration algorithm), that determines for which game nodes s it holds that $v_s \geq 0$. The `makeACorrespondingGame()` implementation didn't require significant choices thanks to the simple structure of the algorithm. On the contrary, in the implementation of `solveMPG-Threshold()` we introduced some further ideas in order to speed-up the algorithm and avoid unnecessary computations. In particular, it is not necessary for `solveMPG-Threshold()` to continue to determine other potential value $v_{s'}$ as soon as it determines a value $v_s < 0$: at this point we can already conclude that the network is not consistent and, with a lower computational cost, we can yield a generalized negative circuit assessing this fact (Lemmas 4 and 5).

Moreover, we verified that there is an important data structure in the original Value Iteration algorithm, a queue, that is not further specified by the authors and that different implementations of it affect the performance of the algorithm.

Therefore, we decided to verify whether `solveMPG-Threshold()` performance could be appreciably improved adding a suitable stopping criterion and a proper queue implementation. Table 2 reports the obtained results, mean execution time μ and its standard deviations σ, determined running the following different versions of `solveMPG-Threshold()` on the same data set of 10^3 not consistent HyTNs[4] having $|V| = 10^6$ and $W = 10^3$:

1. **FIFO Queue**: the original queue is implemented as a FIFO queue;
2. **LIFO Queue:** the original queue is implemented as a a LIFO queue (stack);
3. **LIFO Queue+Stopping Criterion:** the queue is implemented as stack and the computation is halted either when all potential values are stable or when any of them is negative;
4. **Max-Priority Queue:** the original queue is implemented as a Fibonacci's heap.

The results show that, in general, `solveMPG-Threshold()` performance becomes better if the original queue is implemented as a stack and, in particular, a further improvement can be obtained if the stopping criterion is also considered. Nevertheless, such improvements can only partially reduce the statistics variability of the running time, as it is shown in the following experimental results.

As mentioned above, the goal of Test 1 was to determine the average computation time of Algorithm 2 implementation for different values of n to study the practical computation time dependency on such parameter.

The instances in Test 1 come from the `randomgame` generator, except those for the first row of the table in Fig. 15a which have been built by the ATAPIS workflow random generator. In particular, for each $n \in \{1 \cdot 10^5, 2 \cdot 10^5, \ldots, 10 \cdot 10^5\}$, 1600 HyTN instances with maximum weight $W := 1000$ and unknown consistency state were generated by `randomgame`, whereas 1600 HyTNs of unknown consistency state and order n around 400 were generated by ATAPIS. The results of the test are summarized in Fig. 15, where each execution mean time is depicted as a point with a vertical bar representing its confidence interval determined according to its standard deviation.

The depicted function interpolating the mean values shows that the practical performance of the algorithms is definitely better than the theoretical worst-case bound of $O((n + m')mW)$; in our case this last is $O(n^2)$ since in the generated data sets W is constant and $m, m' \in \Theta(n)$. Fig. 15c depicts the interpolating function of experimental execution times and, in red, the function $n^2/10^{10}$ as a reasonable surrogate for the worst-case execution time. The comparison shows that the algorithm performs very well in real case executions.

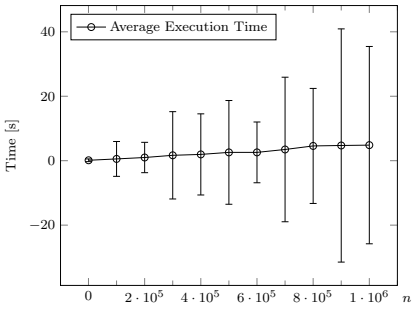
However, since the standard deviation observed in the experiment is not negligible, we further investigated the behavior of the algorithm and we discovered that there is a correlation between the execution time of the algorithm and the consistency state of the input HyTN. Therefore, μ and σ were recalculated considering two kind of HyTN sets: one having all consistent HyTNs, and the other having all not consistent HyTNs.

Fig. 16 depicts average execution times obtained in Test 1 calculated considering samples of either all consistent or all not consistent HyTNs obtained from

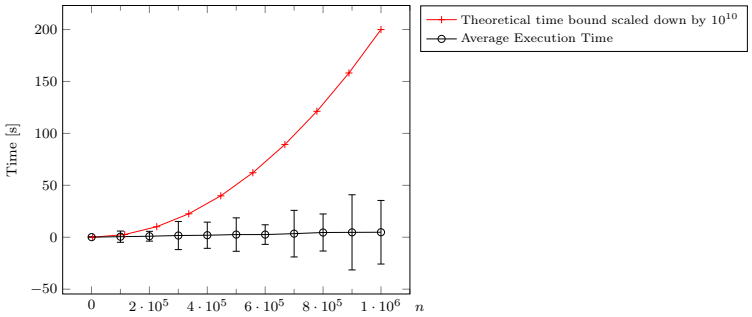
[4] We considered not consistent HyTNs because they practically required more time to be solved.

n	μ (sec)	σ
$< 4 \cdot 10^2$	0.13	0.42
$1 \cdot 10^5$	0.55	5.41
$2 \cdot 10^5$	0.99	4.71
$3 \cdot 10^5$	1.67	13.55
$4 \cdot 10^5$	1.95	12.59
$5 \cdot 10^5$	2.58	16.10
$6 \cdot 10^5$	2.58	9.43
$7 \cdot 10^5$	3.48	22.43
$8 \cdot 10^5$	4.58	17.85
$9 \cdot 10^5$	4.72	36.19
$10 \cdot 10^5$	4.83	30.62

(a) Test 1 results.



(b) Interpolation of average execution times of Test 1.



(c) Comparison between theoretical computation times and experimental ones.

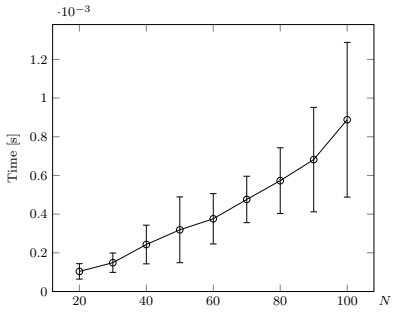
Fig. 15: Results of Test 1: average execution times (μ) and relative standard deviations (σ) over a range of different HyTN orders n. Times are in seconds. Each data set comprised of 1600 HyTN instances of unknown consistency state.

workflow graphs. Fig. 17 offers the same view but for HyTNs obtained from MPG graphs. In general, the mean execution times for consistent HyTNs are smaller than the corresponding ones for not consistent HyTNs; furthermore, they also exhibit a negligible standard deviation. However, for samples of consistent HyTNs obtained from workflows, the standard deviation is not negligible even for samples with size $N = 20$. Part of the reasons for this behavior is given by the structure of the data sets: in each data set HyTNs can differ a lot with respect to their order and, therefore, they may require very different execution times. For example, the data set relating to workflow graphs with 20 activities contains HyTNs with order in range $[48, 94]$. Since the number of activities is usually considered as main parameter in workflow community, we wanted to maintain such structure of data set and experiment results to emphasize the dependency of execution time with respect to such number.

On the other side, for consistent HyTNs determined from MPGs, the observed standard deviation σ is always less that the 10% of the average execution time μ with 99% level of confidence, while for not consistent HyTNs it has not been possible to determine any confidence level because the observed standard deviation σ resulted to be always high due to some hard instances.

N	μ (sec)	σ
10	$6.42 \cdot 10^{-5}$	$1.22 \cdot 10^{-5}$
20	$1.05 \cdot 10^{-4}$	$4.85 \cdot 10^{-5}$
30	$1.50 \cdot 10^{-4}$	$5.7 \cdot 10^{-5}$
40	$2.43 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$
50	$3.20 \cdot 10^{-4}$	$1.78 \cdot 10^{-4}$
60	$3.77 \cdot 10^{-4}$	$1.38 \cdot 10^{-4}$
70	$4.77 \cdot 10^{-4}$	$1.28 \cdot 10^{-4}$
80	$5.73 \cdot 10^{-4}$	$1.80 \cdot 10^{-4}$
90	$6.82 \cdot 10^{-4}$	$2.79 \cdot 10^{-4}$
100	$8.89 \cdot 10^{-4}$	$4.10 \cdot 10^{-4}$

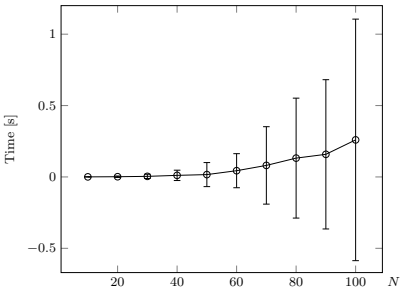
(a) Average execution times for consistent HyTNs obtained from workflow graphs with N activities.



(b) Interpolation of average execution times of Table 16a.

N	μ (sec)	σ
10	$4.45 \cdot 10^{-4}$	$1.38 \cdot 10^{-3}$
20	$1.50 \cdot 10^{-3}$	$5.10 \cdot 10^{-3}$
30	$4.04 \cdot 10^{-3}$	$1.48 \cdot 10^{-2}$
40	$1.10 \cdot 10^{-2}$	$3.62 \cdot 10^{-2}$
50	$1.64 \cdot 10^{-2}$	$8.42 \cdot 10^{-2}$
60	$4.36 \cdot 10^{-2}$	$1.20 \cdot 10^{-1}$
70	$8.08 \cdot 10^{-2}$	$2.71 \cdot 10^{-1}$
80	$1.31 \cdot 10^{-1}$	$4.20 \cdot 10^{-1}$
90	$1.59 \cdot 10^{-1}$	$5.22 \cdot 10^{-1}$
100	$2.59 \cdot 10^{-1}$	$8.46 \cdot 10^{-1}$

(c) Average execution times for not consistent HyTNs obtained from random workflow graphs with N activities.



(d) Interpolation of average execution times of Table 16c.

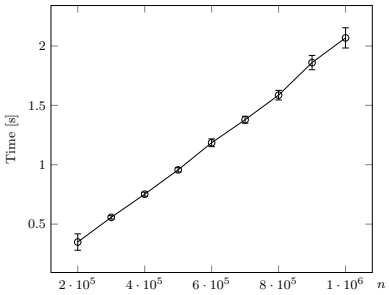
Fig. 16: Average execution times obtained in Test 1 calculated considering samples of either all consistent or all not consistent HyTNs obtained from workflow graphs.

Even though procedure `solveMPG-Threshold()` could require up to $\Theta(W)$ updates according to the theoretical worst-case bound, our experiments suggest that, in practice, some kind of dependency of the running time on W is appreciable only for a few MPG games, all associated to not consistent HyTN instances.

The goal of Test 2 was to determine the average computation time of Algorithm 2 for different values of W, in order to understand how much the practical computation time is dependent on W. Therefore, we considered three possible edge weight ranges, $[10^2, 10^3]$, $[10^5, 10^6]$, and $[10^8, 10^9]$, and for each of them two data sets have been built using the `randomgame` generator, one comprising only consistent HyTNs, and the other only not consistent ones. Each data set comprised of 800 HyTNs instances having $|V| = 10^5$ nodes, $m, m' \in \Theta(n)$ and edge weights in the corresponding weight range. Fig. 18 depicts the results on these six data sets. Applying the worst-case analysis for these data sets, it results that the time complexity should be $O(W)$ since n, m and m' are constants. On the contrary, the determined interpolation functions representing the experimental execution times do not show any clear dependence on W. This result suggests that, in practice, uniform random weighted instances are decided very quickly with respect to

n	μ (sec)	σ
$1 \cdot 10^5$	0.16	0.04
$2 \cdot 10^5$	0.35	0.07
$3 \cdot 10^5$	0.56	0.01
$4 \cdot 10^5$	0.75	0.02
$5 \cdot 10^5$	0.96	0.02
$6 \cdot 10^5$	1.18	0.03
$7 \cdot 10^5$	1.38	0.03
$8 \cdot 10^5$	1.59	0.04
$9 \cdot 10^5$	1.86	0.06
$10 \cdot 10^5$	2.07	0.08

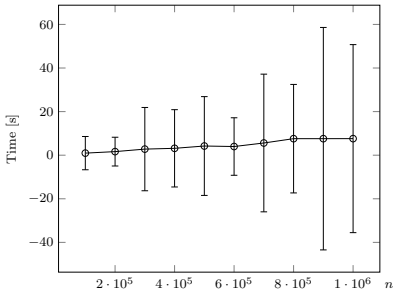
(a) Average execution times for consistent HyTNs obtained from MPGs.



(b) Interpolation of average execution times of Table 17a.

n	μ (sec)	σ
$1 \cdot 10^5$	0.95	7.63
$2 \cdot 10^5$	1.64	6.60
$3 \cdot 10^5$	2.79	19.11
$4 \cdot 10^5$	3.15	17.73
$5 \cdot 10^5$	4.21	22.67
$6 \cdot 10^5$	3.98	13.19
$7 \cdot 10^5$	5.60	31.58
$8 \cdot 10^5$	7.58	24.89
$9 \cdot 10^5$	7.58	51.03
$10 \cdot 10^5$	7.60	43.14

(c) Average execution times for not consistent HyTNs obtained from MPGs.

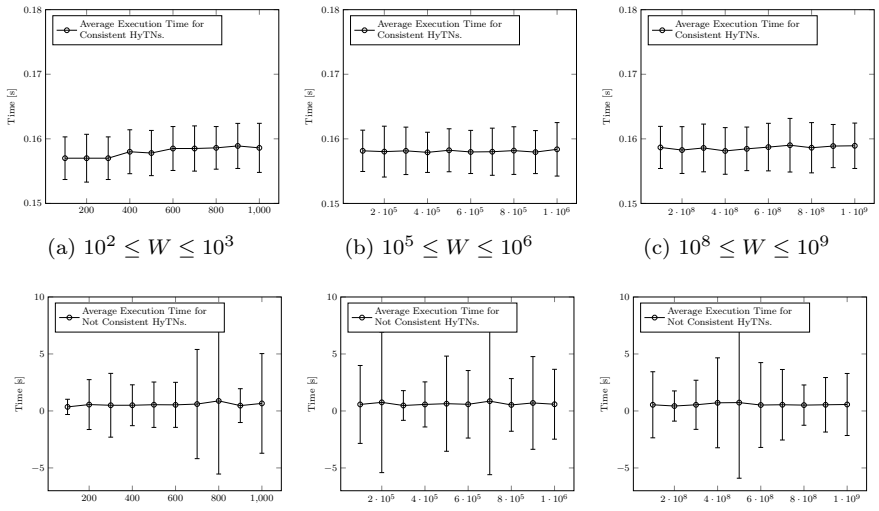


(d) Interpolation of average execution times of Table 17c.

Fig. 17: Average execution times obtained in Test 1 calculated for samples of either all consistent or all not consistent HyTNs obtained from MPG graphs.

the magnitude of their weights and that the algorithm does not seem to exhibit the worst-case pseudo-polynomial behavior predicted in the asymptotic analysis. Moreover, the average execution times for each data set comprising only consistent HyTNs are less than those for the corresponding data of only not consistent HyTNs. Only for consistent HyTNs data sets the standard deviation was below 7% than the average execution time with a confidence of 99%.

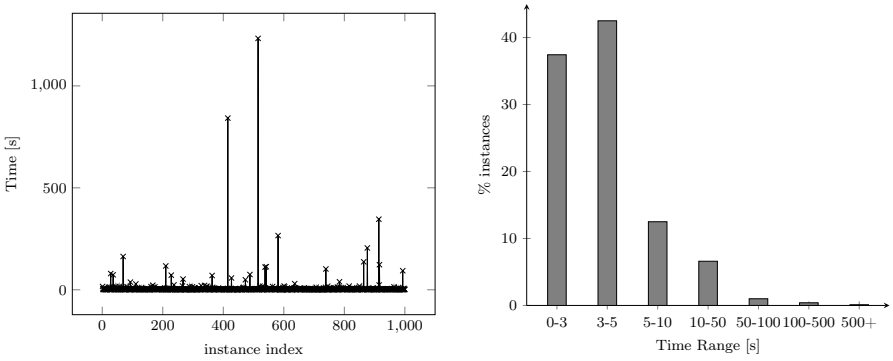
In order to better understand how some execution times affect the value of the standard deviation, we conducted a third experiment, Test 3, with the goal to visualize the distribution of the instances with computation times significantly above the average. Procedure `solveMPG-Threshold()` has been executed on 10^3 randomly generated not consistent HyTNs, each having order $n = 10^6$ and $W \approx 10^3$. The determined running times are depicted in Fig. 19a: most of the instances are decided very quickly, i.e., in a time between 0 and 10 seconds, while a smaller portion of the HyTNs required a time between 10 and 500 seconds. In more details, in repeated tests we verified that, approximately, 1% of the HyTN instances required a time between 50 and 100 seconds to be decided, 0.4% required a time between 100 and 500 seconds, and, finally, only 0.1% required more than 500 seconds. These results are shown in Fig. 19b.



(a) $10^2 \leq W \leq 10^3$ (b) $10^5 \leq W \leq 10^6$ (c) $10^8 \leq W \leq 10^9$

(d) $10^2 \leq W \leq 10^3$ (e) $10^5 \leq W \leq 10^6$ (f) $10^8 \leq W \leq 10^9$

Fig. 18: Average execution times in Test 1 calculated considering samples of either all consistent or all not consistent HyTNs.

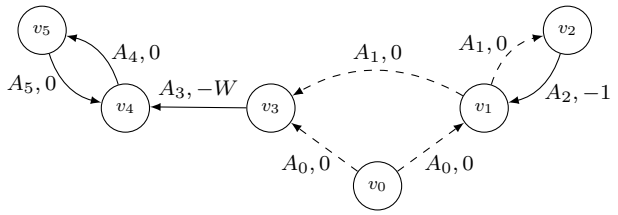


(a) Execution times of 10^3 not consistent HyTNs instances with $n = 10^6$ and $W = 10^3$.

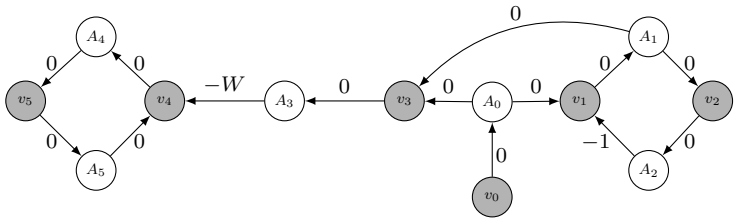
(b) Instance classification with respect to `solveMPG-Threshold()` execution time.

Fig. 19: `solveMPG-Threshold()` execution times obtained in Test 3 determined considering samples of all not consistent HyTN instances.

Such behavior has been confirmed in other tests with different graph orders and maximum edge weight values. In several experiments we conducted, we observed that the maximum execution time of the algorithm keeps growing as we enlarge the size of the dataset. This explains why the standard deviation can't be reduced. If we could characterize such hard instances in general, we would be making a major progress in understanding the computational complexity of MPGs. We didn't find any pattern or property that characterizes the found hard instance. Here we can only show a simple family of HyTNs instances in which the execution time grows



(a) A HyTN $\mathcal{H}_W$ for which Algorithm 2 takes $\Theta(W)$ time.

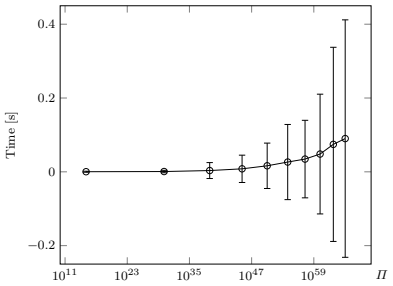


(b) The corresponding MPG $G_{\mathcal{H}_W}$ computed by Algorithm 1.

Fig. 20: A HyTN which requires $\Theta(W)$ computation time by Algorithm 2.

Π	μ (sec)	σ
$1.13 \cdot 10^{15}$	$3.02 \cdot 10^{-4}$	$1.36 \cdot 10^{-3}$
$1.27 \cdot 10^{30}$	$7.78 \cdot 10^{-4}$	$3.34 \cdot 10^{-3}$
$8.08 \cdot 10^{38}$	$3.45 \cdot 10^{-3}$	$2.15 \cdot 10^{-2}$
$1.43 \cdot 10^{45}$	$8.13 \cdot 10^{-3}$	$3.70 \cdot 10^{-2}$
$1.00 \cdot 10^{50}$	$1.63 \cdot 10^{-2}$	$6.15 \cdot 10^{-2}$
$9.10 \cdot 10^{53}$	$2.65 \cdot 10^{-2}$	$1.02 \cdot 10^{-1}$
$2.02 \cdot 10^{57}$	$3.46 \cdot 10^{-2}$	$1.05 \cdot 10^{-1}$
$1.61 \cdot 10^{60}$	$4.82 \cdot 10^{-2}$	$1.62 \cdot 10^{-1}$
$5.80 \cdot 10^{62}$	$7.44 \cdot 10^{-2}$	$2.63 \cdot 10^{-1}$
$1.13 \cdot 10^{65}$	$9.01 \cdot 10^{-2}$	$3.21 \cdot 10^{-1}$

(a) Average execution times obtained for different values of the product of the heads of hyperarcs Π.



(b) Interpolation of average execution times in Table 21a.

Fig. 21: Average Execution Times obtained in Test 4.

linearly with W. The family is given by just one single HyTN graph where only W changes, as depicted in Fig. 20a. The corresponding MPG is shown in Fig. 20b and provides a clear example where Brim's Value Iteration algorithm [7] performs poorly. It is worth noting that in the context of MPGs this example can be reduced down to 6 nodes.

Finally, in order to show how much the running time is dependent on the number of different positional strategies of one player, in Test 4 the average computation time has been calculated with respect to different values of the product of the heads of hyperarcs (i.e., $\prod_{A \in \mathcal{A}} |H_A|$) in a HyTN. In particular, for each $\Pi \in \{10^{15}, 10^{30}, \ldots, 10^{65}\}$, 2500 HyTNs instances $(V, \mathcal{A})$ each having $|V| = 50$ nodes, $\prod_{A \in \mathcal{A}} |H_A| \approx \Pi$, and $W = 10^3$ have been generated by means of `randomgame` generator. The results of the evaluation are depicted in Fig. 21, where Π values are drawn in logarithmic scale. Analyzing the diagram in the fig-

ure it is possible to say that, experimentally, the average execution time increases only logarithmically with respect to the number of different positional strategies of one player. This results is quite interesting because, considering the HyTN in Fig. 20a, it is evident that the time for checking a HyTN is more dependent on the edge weight magnitude than on the number of different positional strategies of one player.

8 Related Work

In the literature there are some extension proposals of the STN model to augment the capability to represent temporal constraints.

In the STN seminal paper [18], Dechter et al. firstly proposed to consider the Temporal Constraint Satisfaction Problem (TCSP). A binary constraint in a TCSP is represented using a set of intervals rather than a single interval as in an STN. In particular, a binary constraint $C_{ij} = \{[a_1, b_1], [a_2, b_2], \ldots, [a_l, b_l]\}$ between time points x_i and x_j represents the disjunction $a_1 \leq x_j - x_i \leq b_1 \vee a_2 \leq x_j - x_i \leq b_2 \vee a_l \leq x_j - x_i \leq b_l$. The problem of verifying consistency of a TCSP is NP-complete as the same authors showed in the paper; hence, they finally propose to consider STNs as a tractable simplified model.

A similar kind of generalization considering disjunction of temporal distance constraints was proposed by Stergiou and Koubarakis [44] defining the Disjunctive Temporal Problem (DTP). A DTP consists of a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ having continuous domains and representing time points and a set of disjunctive difference constraints between the time points in the form: $a_1 \leq x_{i_1} - x_{j_1} \leq b_1 \vee a_2 \leq x_{i_2} - x_{j_2} \leq b_2 \vee \ldots \vee a_k \leq x_{i_k} - x_{j_k} \leq b_k$; where $x_{i_1}, x_{j_1}, \ldots, x_{i_k}, x_{j_k}$ are time points from X and $a_1, b_1, \ldots, a_k, b_k$ are real numbers. A DTP is consistent if there exists an instantiation of variables X to real numbers satisfying all the constraints. Since DTPs are a generalization of TCSPs, also for DTPs the consistency check problem is NP-complete. In [44] the authors presented some of the theoretical results characterizing the possible backtracking algorithms that solve the consistency problem in terms of search nodes visited and consistency checks performed.

In 2005, Kumar proposed to consider a restricted class of DTP in order to maintain some of the expressive power of DTPs but, at the same time, allowing an efficient consistency check. In particular, in [40], RDTPs (restricted DTPs) is defined as a disjunctive temporal problem where a constraint is one of the following types: (Type 1) $(l \leq x_i - x_j \leq u)$, (Type 2) $(l_1 \leq x_i \leq u_1) \vee (l_2 \leq x_i \leq u_2) \ldots (l_j \leq x_i \leq u_j)$, (Type3) $(l_1 \leq x_i \leq u_1) \vee (l_2 \leq x_j \leq u_2)$, where x_i and x_j represent a timepoint variable, and l_i, u_i real values. An RDTP instance can be solved in strongly polynomial-time deterministic algorithm transforming it into a binary Constraint Satisfiability Problem (CSP) over meta variables representing constraints of Type 2 or Type 3 and, then, showing that such binary constraints are also *connected row-convex (CRC)* constraints, and, then, exploiting the properties of CRC constraints. An instantiation of a consistency check algorithm for RDTPs that further exploits the structure of CRC constraints has a running time complexity of $O((|TP_2| + |TP_3|)^3 d_{\max}^2 + (|TP_2| + |TP_3|)^2 (NM + d_{\max}^2))$, where TP_2 is the set of Type 2 constraints, TP_3 is the set of Type 3 ones, $d_{\max}$ is the maximum number of disjuncts in any constraint, and N/M is the number of the

nodes/arcs of the instance, respectively. In the same paper, Kumar presented also a simpler and faster, but *randomized*, algorithm for the same class RDTP.

An attempt to model some aspects of STNs similar to those addressed by HyTNs was lead in [2], where fun-in and fun-out subgraphs much resembling our multi-tail and multi-head hyperarcs were considered. However, since the problem 1-in-3-SAT is NP-complete even when all the literals comprising the clauses are positive, it readily follows that their models lead to NP-complete problems even when fun-out subgraphs (or fun-in subgraphs) are banned. As such, the opportunity for tractability spotlighted in this paper is missed in those models.

Another approach to extend STN is represented by the proposal of Khatib et al. [28, 29]. They introduced the characterization of *hard* and *soft* constraints. STNs are able to model just hard temporal constraints, i.e., they can represent instances where all constraints have to be satisfied, and that the solutions of a constraint are all equally satisfying. However, such assumption can be too much restrictive in some real-life scenarios. For example, it may be that some solutions are preferred with respect to others and, hence, the main problem is to find a way to satisfy them optimally, according to the preferences specified. To address these kind of problems, in [28] the authors introduced a framework in which each temporal constraint is associated with a preference function specifying the preference for each distance or duration; a *soft* simple temporal constraint is a 4-tuple $\langle (X, Y), I, A, f \rangle$ consisting of (1) an ordered pair of variables (X, Y) over the integers, called the scope of the constraint; (2) an interval $I = [a, b]$, where a and b are integers such that $a \leq b$; (3) a set of preferences A; (4) a preference function f, where $f : [a, b] \mapsto A$ is a mapping of the elements belonging to interval I into preference values, taken from set A. An assignment v_x and v_y to the variables X and Y is said to satisfy the constraint $\langle (X, Y), I, A, f \rangle$ if and only if $a \leq v_y - v_x$. In such a case, the preference associated to the assignment by the constraint is $f(v_y - v_x)$. Using soft simple temporal constraint, a new model of temporal constraint network has been introduce: the *Simple Temporal Problem with Preferences (STPP)*. In general, each solution of a STPP has a global preference value, obtained by combining in a suitable way the preference levels at which the solution satisfies the constraints. The optimal solutions of an STPP are those solutions which are not dominated by any other solution in terms of global preference. It was shown in [28] that, in general, STPPs belongs to the class of NP-hard problems. When the preference functions are semi-convex and some other side conditions are observed, then the problem to find an optimal solutions of an STPP is tractable [29].

Finally, another kind of possible extension is represented by the use of $\neq$ operator instead of $\leq$ in the binary constrains of STNs. Koubarakis [30] showed that if in a STN temporal constraints are used together with disequations in the form $x - y \neq r$, where r is a real constant, then the problem of deciding consistency is still tractable. This extension does not allow the specification of alternative constraints but it is interesting because it allows to exclude some solutions maintaining the consistency problem tractable.

9 Conclusions and Future Work

In the literature, there are different frameworks and approaches aimed to extend the STN model allowing the representation of disjunctive temporal constraints [18,

44], but at cost of an exponential-time consistency check procedure. The only extension with a polynomial time consistency check procedure we are aware of is the one of Kumar [40] mentioned in Section 8.

In this paper, we proposed a novel extension, called Hyper Temporal Network (HyTN), where it is possible to represent a new kind of disjunctive constraint, hyper constraint, and to check the consistency of a network in pseudo-polynomial time. A hyper constraint is a suitable set of STN distance constraints and it is satisfied if at least one distance constraint is satisfied. There could be two kinds of hyperarc: multi-head and multi-tail. In a multi-head hyperarc, its distance constraints are between a common source timepoint and different destination timepoints. In a multi-tail hyperarc, its distance constraints are between different source timepoints and a common destination timepoint.

A HyTN is said consistent if it is possible to determine an assignment for all its timepoints such that all hyperarcs are satisfied. The computational complexity of the consistency problem of a HyTN is NP-complete when instances contain both kinds of hyperarc.

On instances containing either only multi-tail hyperarcs, or only multi-head hyperarcs, the consistency problem can be solved by reducing it, in a very efficient way, to the search of a winning strategy in an equivalent Mean Payoff Game (MPG), and exploiting the known winning-strategy search algorithms for MPGs.

Moreover, we presented an empirical analysis of the efficiency of the resulting consistency check algorithm. The empirical analysis shows that the proposed algorithm can be effectively used in real cases and confirms the general robustness of our approach.

As future work we are investigating the frontier of practical efficient consistency checking for possible generalizations of the HyTN model as, for example, those including contingent constraints [46] or conditional ones [45].

References

1. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A. (2003): Workflow patterns. Distributed and Parallel Databases **14**(1), 5–51. DOI 10.1023/A:1022883727209
2. Barták, R., Čepek, O.: Temporal networks with alternatives: Complexity and model. In: D. Wilson, G. Sutcliffe (eds.) Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, May 7-9, 2007, Key West, Florida, USA., pp. 641–646. AAAI Press (2007)
3. Bellman, R. (1958): On a routing problem. Quarterly of Applied Mathematics **16**(1), 87–90
4. Bettini, C., Wang, X.S., Jajodia, S. (2002): Temporal reasoning in workflow systems. Dist. & Paral. Data. **11**(3), 269–306. DOI 10.1023/A:1014048800604
5. Björklund, H., Vorobyov, S. (2007): A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Applied Mathematics **155**(2), 210 – 229. DOI 10.1016/j.dam.2006.04.029
6. Brim, L., Chaloupka, J. (2012): Using strategy improvement to stay alive. Int. J. Found. Comput. Sci. **23**(3), 585–608. DOI 10.1142/S0129054112400291
7. Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., Raskin, J.F. (2011): Faster algorithms for mean-payoff games. Formal Methods in System Design **38**(2), 97–118. DOI 10.1007/s10703-010-0105-x
8. Chinn, S.J., Madey, G.R. (2000): Temporal representation and reasoning for workflow in engineering design change review. IEEE Transactions on Engineering Management **47**(4), 485–492. DOI 10.1109/17.895343

9. Combi, C., Gambini, M., Migliorini, S., Posenato, R.: Modelling temporal, data-centric medical processes. In: Proc. of the 2nd ACM SIGHIT Int. Health Informatics Symp., IHI '12, pp. 141–150. ACM, New York, NY, USA (2012). DOI 10.1145/2110363.2110382
10. Combi, C., Gambini, M., Migliorini, S., Posenato, R. (2014): Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. Systems, Man, and Cybernetics: Systems, IEEE Transactions on **44**(9), 1182–1203. DOI 10.1109/TSMC.2014.2300055
11. Combi, C., Gozzi, M., Posenato, R., Pozzi, G. (2012): Conceptual modeling of flexible temporal workflows. ACM Trans. Auton. Adapt. Syst. **7**(2), 19:1–19:29. DOI 10.1145/2240166.2240169
12. Combi, C., Posenato, R.: Controllability in temporal conceptual workflow schemata. In: BPM 2009 - Proc. of the 7th Business Process Management Conference, pp. 64–79 (2009). DOI 10.1007/978-3-642-03848-8_6
13. Combi, C., Pozzi, G.: Architectures for a temporal workflow management system. In: Proc. of the 2004 ACM Symp. on Applied Computing, SAC '04, pp. 659–666. ACM, New York, NY, USA (2004). DOI 10.1145/967900.968040
14. Comin, C.: A HyTN Consistency Check Algorithm Implementation in C++. `http://profs.scienze.univr.it/~posenato/software/hytn/2015_v1_Code.tgz` (2015)
15. Comin, C., Posenato, R., Rizzi, R.: A tractable generalization of simple temporal networks and its relation to mean payoff games. In: 21st International Symposium on Temporal Representation and Reasoning (TIME 2014), pp. 7–16. IEEE CPS (2014). DOI 10.1109/TIME.2014.19
16. Comin, C., Rizzi, R.: Dynamic consistency of conditional simple temporal networks via mean payoff games: a singly-exponential time DC-Checking. In: 22nd International Symposium on Temporal Representation and Reasoning (TIME 2015), pp. 19–28. IEEE CPS (2015). DOI 10.1109/TIME.2015.18
17. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press (2001)
18. Dechter, R., Meiri, I., Pearl, J. (1991): Temporal constraint networks. Artificial Intelligence **49**(1–3), 61–95. DOI 10.1016/0004-3702(91)90006-6
19. Eder, J., Gruber, W., Panagos, E.: Temporal modeling of workflows with conditional execution paths. In: M. Ibrahim, J. Küng, N. Revell (eds.) Database and Expert Systems Applications (DEXA 2000), *LNCS*, vol. 1873, pp. 243–253. Springer Berlin Heidelberg (2000). DOI 10.1007/3-540-44469-6_23
20. Eder, J., Panagos, E., Rabinovich, M.: Time constraints in workflow systems. In: M. Jarke, A. Oberweis (eds.) Advanced Information Systems Engineering, *LNCS*, vol. 1626, pp. 286–300. Springer Berlin Heidelberg (1999). DOI 10.1007/3-540-48738-7_22
21. Ehrenfeucht, A., Mycielski, J. (1979): Positional strategies for mean payoff games. Int. Journal of Game Theory **8**(2), 109–113. DOI 10.1007/BF01768705
22. Ford Jr., L.R., Fulkerson, D.R.: Flows in networks, vol. 3. Princeton University Press (1962)
23. Gonzalez del Foyo, P.M., Reinaldo Silva, J.: Using time Petri Nets for modeling and verification of timed constrained workflow systems. In: ABCM Symposium Series in Mechatronics, pp. 471–478. Dept. Of Mechatronics, University of São Paulo, Brazil (2008)
24. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
25. Hollingsworth, D.: The workflow reference model. http://www.wfmc.org/standards/model.htm (1995)
26. Hunsberger, L., Posenato, R., Combi, C.: A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In: F. Grandi, M. Lange, A. Lomuscio (eds.) 22st International Symposium on Temporal Representation and Reasoning (TIME 2015), pp. 4–18. IEEE CPS (2015). DOI 10.1109/TIME.2015.26
27. Jurdziński, M. (1998): Deciding the winner in parity games is in UP ∩ co-UP. Information Processing Letters **68**(3), 119–124. DOI 10.1016/S0020-0190(98)00150-1
28. Khatib, L., Morris, P., Morris, R., Rossi, F.: Temporal constraint reasoning with preferences. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'01, pp. 322–327. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
29. Khatib, L., Morris, P., Morris, R., Rossi, F., Sperduti, A., Venable, K.B. (2007): Solving and learning a tractable class of soft temporal constraints: Theoretical and experimental results. AI Communications **20**(3), 181–209

30. Koubarakis, M. (1997): From local to global consistency in temporal constraint networks. Theoretical Computer Science **173**(1), 89 – 112. DOI 10.1016/S0304-3975(96)00192-2
31. Lanz, A., Reichert, M.: Enabling time-aware process support with the atapis toolset. In: L. Limonad, B. Weber (eds.) Proceedings of the BPM Demo Sessions 2014, *CEUR Workshop Proceedings*, vol. 1295, pp. 41–45. CEUR (2014)
32. Lanz, A., Weber, B., Reichert, M. (2012): Time patterns for process-aware information systems. Requirements Engineering **19**(2), 113–141. DOI 10.1007/s00766-012-0162-3
33. Lifshits, Y., Pavlov, D. (2007): Potential theory for mean payoff games. Journal of Mathematical Sciences **145**(3), 4967–4974. DOI 10.1007/s10958-007-0331-y
34. Mendling, J., Reijers, H.A., van der Aalst, W.M.P. (2010): Seven process modeling guidelines (7PMG). Information and Software Technology **52**(2), 127–136. DOI 10.1016/j.infsof.2009.08.004
35. Merlin, P., Farber, D.J. (1976): Recoverability of communication protocols–implications of a theoretical study. Communications, IEEE Transactions on **24**(9), 1036–1043. DOI 10.1109/TCOM.1976.1093424
36. Morris, P., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'01, pp. 494–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
37. Pani, A., Bhattacharjee, G. (2001): Temporal representation and reasoning in artificial intelligence: A review. Mathematical and Computer Modelling **34**(1–2), 55–80. DOI 10.1016/S0895-7177(01)00049-8
38. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
39. pgsolver: The pgsolver collection of parity game solvers. https://github.com/tcsprojects/pgsolver (2013)
40. Satish Kumar, T.K.: On the tractability of restricted disjunctive temporal problems. In: ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling, pp. 110–119 (2005)
41. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: M. Kaminski, S. Martini (eds.) Computer Science Logic, *LNCS*, vol. 5213, pp. 369–384. Springer (2008). DOI 10.1007/978-3-540-87531-4_27
42. Schewe, S., Trivedi, A., Varghese, T.: Symmetric strategy improvement. In: M.M. Halldórsson, K. Iwama, N. Kobayashi, B. Speckmann (eds.) Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 9135, pp. 388–400. Springer (2015). DOI 10.1007/978-3-662-47666-6_31
43. Smith, D., Frank, J., Jónsson, A. (2000): Bridging the gap between planning and scheduling. Knowledge Engineering Review **15**(1), 47–83. DOI 10.1017/S0269888900001089
44. Stergiou, K., Koubarakis, M. (2000): Backtracking algorithms for disjunctions of temporal constraints. Artificial Intelligence **120**(1), 81–117. DOI 10.1016/S0004-3702(00)00019-9
45. Tsamardinos, I., Vidal, T., Pollack, M.E. (2003): Ctp: A new constraint-based formalism for conditional, temporal planning. Constraints **8**(4), 365–388. DOI 10.1023/A:1025894003623
46. Vidal, T., Fargier, H. (1999): Handling contingency in temporal constraint networks: from consistency to controllabilities. Journal of Experimental and Theoretical Artificial Intelligence **11**(1), 23–45. DOI 10.1080/095281399146607
47. Zwick, U., Paterson, M. (1996): The complexity of mean payoff games on graphs. Theoretical Computer Science **158**(1–2), 343–359. DOI 10.1016/0304-3975(95)00188-3