Appl Intell
DOI 10.1007/s10489-010-0246-z

Conditional and composite temporal CSPs

Malek Mouhoub · Amrudee Sukpan

Abstract Constraint Satisfaction Problems (CSPs) have been widely used to solve combinatorial problems. In order to deal with dynamic CSPs where the information regarding any possible change is known *a priori* and can thus be enumerated beforehand, *conditional constraints* and *composite variables* have been studied in the past decade. Indeed, these two concepts allow the addition of variables and their related constraints in a dynamic manner during the resolution process. More precisely, a *conditional constraint* restricts the participation of a variable in a feasible scenario while a composite variable allows us to express a disjunction of variables where only one will be added to the problem to solve. In order to deal with a wide variety of real life applications under temporal constraints, we present in this paper a unique temporal CSP framework including numeric and symbolic temporal information, conditional constraints and composite variables. We call this model, a Conditional and Composite Temporal CSP (or CCTCSP). To solve the CCTCSP we propose two methods respectively based on Stochastic Local Search (SLS) and constraint propagation. In order to assess the efficiency in time of the solving methods we propose, experimental tests have been conducted on randomly generated CCTCSPs. The results demonstrate the superiority of a variant of the Maintaining Arc Consistency (MAC) technique (that we call MAX+) over the other constraint propagation strategies, Forward Checking (FC) and its variants, for both consistent and inconsistent problems. It has also been shown that, in the case of consistent problems, MAC+ outperforms the SLS method Min Conflict Random Walk (MCRW) for highly constrained CCTCSPs while both methods have comparable time performance for under and middle constrained problems. MCRW is, however, the method of choice for highly constrained CCTCSPs if we decide to trade search time for the quality of the solution returned (number of solved constraints).

Keywords Temporal Reasoning · Constraint Satisfaction · Stochastic Local Search

M. Mouhoub (✉) · A. Sukpan
Univ. of Regina, Regina, Canada
e-mail: mouhoubm@uregina.ca

A. Sukpan
e-mail: sukpan1a@uregina.ca

Published online: 30 July 2010

1 Introduction

1.1 Problem definitions

A well-known approach to managing the numeric and symbolic aspects of time is to view them as Constraint Satisfaction Problems (CSPs). We talk then about temporal constraint networks [2, 7, 12, 20, 30, 46, 49]. Here, a CSP involves a list of variables defined on discrete domains of values and a list of relations constraining the values that the variables can simultaneously take [11, 21, 27].

In a temporal constraint network, variables, corresponding to temporal objects, are defined on a set of time points or time intervals while constraints can either restrict the domains of the variables and/or express the relative position between the temporal objects these variables represent. The relative position between temporal objects can be expressed via qualitative or quantitative relations. Quantitative relations are temporal distances between temporal variables while qualitative relations represent incomplete and less specific symbolic information between variables. Constraint propagation techniques and backtrack search are then used to check the consistency of the temporal network and

to infer new temporal information. While a considerable research work has been concerned with reasoning on the metric or the symbolic aspects of time (respectively through metric or qualitative networks), little work such as [18, 25, 29, 43] has been developed to manage both types of information. In [33, 34], we have developed a temporal model, TemPro, based on Allen's interval algebra [2] and a discrete representation of time, to express numeric and symbolic time information in terms of qualitative and quantitative temporal constraints. More precisely, TemPro translates an application involving numeric and symbolic temporal information into a binary CSP[1] called Temporal CSP (or TCSP[2]) where variables are temporal events defined on domains of numeric intervals and binary constraints between variables correspond to disjunctions of Allen primitives [2]. The resolution method for solving the TCSP is based on constraint propagation and requires two stages. In the first stage, local consistency is enforced by applying the arc consistency on variable domains and the path consistency on symbolic relations. A backtrack search algorithm using constraint propagation is then performed in the second stage to check the consistency of the TCSP by looking for a feasible scenario. A feasible scenario (or solution to the TCSP) is a complete assignment of intervals to all the events such that all the constraints are satisfied. Note that for some TCSPs local consistency implies the consistency of the TCSP network as indicated in [29]. The backtrack search phase can be avoided in this case.

When solving real life applications under (temporal) constraints we often need to deal with uncertain information. These latter are the result of uncontrollable external factors. Variables, domains and constraints in CSPs and temporal CSPs can thus be uncertain and some research work in this regard has been addressed. In a Mixed-CSP [13], variables are divided into two categories. One includes controllable variables (decision variables) that are totally under control of users and the other includes uncontrollable variables (environmental variables) that are not under control. The uncertainty is reflected by uncontrollable variables. In the stochastic CSP [50], a probability distribution is associated with the domain of each variable. In probabilistic CSPs [13], the uncertain factors are the probabilities of the existence of various constraints. Simple Temporal Networks [12], very popular for representing quantitative temporal information, have been augmented to model the uncertain duration between time points. One of the resulting frameworks is called Simple Temporal Network with Uncertainty (STNU) [47]. Solving algorithms and tractable classes for the STNU have been proposed in [32, 48]. Another resulting framework is the Fuzzy Temporal Constraint Network [28]. This latter model extends the STN with fuzzy durations using the possibilistic distributions. Qualitative temporal networks handling uncertainty include the fuzzy extension of Allen's Interval Algebra [3] and the Probabilistic Temporal Interval Network (PTI) [35]. Both of these two formalisms are based on the possibility theory used to model the uncertain symbolic relations by assigning a preference degree to every basic Allen's primitive [2]. A path consistency algorithm has been proposed in each model. Finally, a framework for reasoning about qualitative and metric temporal relations between vague time periods has been proposed in [40].

1.2 Motivation

Representing and reasoning about numeric and/or symbolic aspects of time is crucial in many real world applications [14] such as scheduling and planning [1, 4, 8, 18, 26], natural language processing [22, 42], molecular biology [19] and temporal database [9, 44]. In all these applications we have to deal with both the numeric and the symbolic aspects of time in a dynamic and evolving world. While many efforts have been dedicated to tackle constraint problems in a dynamic environment [10, 16, 23, 24, 31, 37, 38, 45], none of these works manages both the numeric and the symbolic aspects of time in a very expressive way as we will show in the next section. This has motivated us to develop such a model based on our TemPro framework.

1.3 Proposed solution and contributions

We present in this paper an extension of the modeling framework TemPro in order to deal with a large variety of dynamic real world temporal applications and where the information regarding any possible change can be enumerated beforehand. This extension includes the following:

Variable status. Each variable has either active or inactive status. Only active variables require an assignment from their domain of values. Inactive variables will not be considered during the resolution of the temporal network until they are activated. A variable can be activated by default (in the initial problem), through an activity constraint or a composite variable.

Composite temporal variables. Composite temporal variables are variables whose values are temporal events.[3] During the resolution process, an active composite variable will be assigned (replaced with) one event from its

[1] In a binary CSP, constraints can only be unary or binary relations.

[2] Note that the acronym TCSP was used in [12]. The well known TCSP, as defined by Dechter et al., is a quantitative temporal network used to represent only numeric temporal information. Nodes represent time points while arcs are labeled by a set of disjoint intervals denoting a disjunction of bounded differences between each pair of time points.

[3] An event is defined here as a preposition that holds over a time interval.

domain. This latter event will then be activated. For instance, if we have a list of events $Evt_1, \ldots, Evt_n$ where only one Evt_i should be present in the problem to solve, then we can create a composite variable X defined on the domain $D_X = \{Evt_1, \ldots, Evt_n\}$.

Activity (or conditional) constraints. An activity constraint has the following form: $X_1 \wedge \cdots \wedge X_p \overset{condition}{\rightarrow} Y$ where $X_1, \ldots, X_p$ and Y are temporal variables (composite or events). This activity constraint will activate Y if $X_1, \ldots, X_p$ are active and *condition* holds on these variables. *condition* corresponds here to the assignment of particular values to the variables $X_1, \ldots, X_p$.

The main difference between an activity constraint and a composite variable is the fact that:

- An activity constraint requires a condition (over a set of active variables) to be satisfied in order to activate a given variable.
- A composite variable represents the fact that among a list of events only one should be activated (without a specific condition).

Note that we could as well express the disjunction of events $Evt_1, \ldots, Evt_n$ we mentioned above (when describing composite variables) using conditional constraints. In this case we need to do the following:

1. Create a variable X defined on $D_X = \{e_1, \ldots, e_n\}$ where each value e_i corresponds to the activation of event Evt_i.
2. Create n activity constraints as follows:
   - $X \overset{X=e_1}{\rightarrow} Evt_1$
   - $\ldots$,
   - $X \overset{X=e_n}{\rightarrow} Evt_n$

Using the above, each time X is assigned a particular value, the corresponding event is activated. As we can see, it is natural and easier to represent a disjunction of events with a composite variable rather than a set of activity constraints. Also, from a conceptual point of view it is better to distinguish between the events that are unconditionally selected (from a list) and activated through composite variables and those activated via activity constraints.

We call a Composite TCSP (CTCSP) a TCSP including composite temporal variables. A CTCSP represents a finite set of possible TCSPs where each TCSP corresponds to a complete assignment of values (temporal events) to composite variables. Solving a CTCSP consists of finding a feasible scenario for one of its possible TCSPs. Solving a TCSP requires a backtrack search algorithm with exponential complexity in time $O(D^N)$ where N is the total number of temporal events and D the domain size of each event (number of values in the domain). The possible number of TCSPs the CTCSP involves is d^M where M is the number of composite variables and d their domain size. Thus, solving a CTCSP requires a backtrack search algorithm of complexity $O(d^M \times D^{N+M})$ in the worst case. We call Conditional CTCSP (CCTCSP) a CTCSP augmented by activity constraints. Solving a CCTCSP can be seen like solving a CTCSP dynamically i.e. when some of the variables (events or composites) and their corresponding constraints are added or removed dynamically during the resolution of the CSP. To overcome this difficulty in practice, we propose in this paper two methods respectively based on constraint propagation and Stochastic Local Search (SLS) for solving efficiently CCTCSPs. Constraint propagation includes Arc Consistency (AC) [27] as well as forward check and MAC strategies [21]. On the other hand, the SLS methods we use are Min-Conflict-Random-Walk (MCRW), Steepest-Descent-Random-Walk (SDRW) and Tabu [39].

In order to assess the efficiency in time of the solving methods we propose, experimental comparative tests have been conducted on random CCTCSPs generated using the model RB [51] as this latter has the ability to generate asymptotically hard instances. The test results demonstrate the superiority of a variant of the MAC technique (that we call MAX+) over the other constraint propagation strategies (FC and its variant FC+) for both consistent and inconsistent problems. It has also been shown that, in the case of consistent problems, MAC+ outperforms the SLS method MCRW for highly constrained CCTCSPs while both methods have comparable time performance for under and middle constrained problems. MCRW is, however, the method of choice for highly constrained CCTCSPs if we decide to trade search time for the quality of the solution returned (number of solved constraints).

1.4 Plan of the paper

The rest of the paper is structured as follows. In the next section we introduce the CCTCSP framework through an example. Sections 3 and 4 are respectively dedicated to the constraint propagation technique and the SLS methods for solving CCTCSPs. Section 5 describes the experimental comparative tests we have conducted on random CCTCSPs. Finally, concluding remarks are covered in Sect. 6.

2 Conditional and composite temporal constraint satisfaction problems (CCTCSPs)

Managing conditional, composite and dynamic CSPs has already been reported in the literature [10, 16, 23, 24, 31, 37, 38, 45]. Mittal and Falkenkainer [31] introduced the notion of *Dynamic Constraint Satisfaction Problems* for configuration problems (renamed *Conditional Constraint Satisfaction Problems (CCSPs)* later). In contrast with the standard CSP paradigm, in a CCSP the set of variables requiring

assignment is not fixed by the problem definition. A variable has either *active* or *nonactive* status. An activity constraint enforces the change of the status of a given variable from *nonactive* to *active*. In [37], Freuder and Sabin have extended the traditional CSP framework by including the combination of three new CSP paradigms : *Meta CSPs, Hierarchical Domain CSPs, and Dynamic CSPs*. This extension is called *composite CSP*. In a composite CSP, the variable values can be entire sub CSPs. A domain can be a set of variables instead of atomic values (as it is the case in the traditional CSP). The domains of variable values can be hierarchically organized. The participation of variables in a solution is dynamically controlled by activity constraints. Jónsson and Frank [23] proposed a general framework using procedural constraints for solving dynamic CSPs. This framework has been extended to a new paradigm called Constraint-Based Attribute and Interval Planning (CAIP) for representing and reasoning about plans [24]. CAIP and its implementation, the EUROPA system, enable the description of planning domains with time, resources, concurrent activities, disjunctive preconditions and conditional constraints. The main difference, comparing to the formalisms we described earlier, is that in this latter framework [23] the set of constraints, variables and their possible values do not need to be enumerated beforehand which gives a more general definition of dynamic CSPs. Note that the definition of dynamic CSPs in [23] is also more general than the one in [10] since in this latter work variable domains are predetermined. Finally, in [45], Tsamardinos et al. propose the Conditional Temporal Problem (CTP) formalism for Conditional Planning under temporal constraints. This model extends the well known qualitative temporal network proposed in [12] by adding instantaneous events (called observation nodes) representing conditional constraints.

We adopt both the CCSP [31] and the composite CSP [37] paradigms and extend the modeling framework TemPro [34] by including conditional temporal constraints and composite temporal events as shown in introduction. TemPro will then have the ability to transform constraint problems involving numeric information, symbolic information, conditional constraints and composite variables into the CCTCSP we have described in introduction. Comparing to the formalisms we mentioned above and those we mentioned in introduction on hybrid temporal constraints [18, 25, 29, 43], ours has the following specifics.

1. Our work focuses on temporal constraints while the previous literature is on general constraints, if we exclude the work in [45] and [24]. Both these two latter formalisms, however, handle only quantitative time information while ours combines both quantitative and qualitative temporal constraints.
2. Our model is application independent and is not restricted to a particular area such as planning or scheduling. It can thus be used in a large variety of applications involving symbolic and|or numeric temporal constraints. Moreover, the qualitative constraints are based on the whole Allen Algebra [2] which offers more expressiveness. Altough this will lead to NP-hard problems, the solving techniques that we will present in the next two sections overcome this difficulty, in practice, as demonstrated by the test results in Sect. 5.
3. Our model is based on a discrete representation of time. Thus, events are defined on discrete values (numeric intervals). This offers an easier way to handle numeric temporal information with different granularities. It will also enable the constraint propagation techniques and approximation methods to be applied in a straight forward manner.
4. Numeric and symbolic temporal constraints as well as conditional constraints and composite variables, are managed within the same constraint graph and not, for example, like in [25] where numeric and symbolic temporal constraints are managed in two separate systems.
5. Our model is more expressive then IxTeT [18] since this latter is restricted to the Point Algebra (PA) while ours is based on the full Allen Interval Algebra (IA). Also, when using a discrete representation of time, temporal problems represented by our model include those represented by the well known formalisms proposed in [29] and [43] as we will show later in example 2.
6. As we will see in Sect. 3, unlike the work in [38] where constraint propagation is performed during search (using forward checking and MAC strategies) through activity constraints, in our model the propagation is performed through compatibility constraints.
7. Unlike the model in [37] where a variable can unfold into sub CSPs in a recursive manner, in our framework a composite variable is defined on a set of events and can thus be replaced by only one event from this set.

In the following we will define the CCTCSP model and its corresponding network (graph representation) through an example.

Definition

A CCTCSP is a tuple $\langle E, D_E, X, D_X, IV, C, Act\rangle$, where:

$E = \{e_1, \ldots, e_n\}$ is a finite set of temporal variables that we call events. In the same way as in [2], we define an event e as the proposition that occurs over the smallest possible time interval I for it to occur:

$$OCCURS(e, I) \wedge I' \subset I \Rightarrow \neg OCCURS(e, I')$$

where I' is a subinterval of I. For the sake of notation simplicity, an event is used in this paper to denote its temporal qualification (time interval during which the event occurs).

$D_E = \{D_{e_1}, \ldots, D_{e_n}\}$ is the set of domains of the events. Each domain D_{e_i} is the finite and discrete set of numeric and continuous time intervals the event e_i can take. D_{e_i} is expressed by the fourfold [$EarliestStart_{e_i}$, $LatestEnd_{e_i}$, $Duration_{e_i}$, $Step_{e_i}$] where $EarliestStart_{e_i}$ and $LatestEnd_{e_i}$ are respectively the earliest start time and the latest end time of e_i, $Duration_{e_i}$ is its duration and $Step_{e_i}$ defines the distance (number of time units) between the starting time of two adjacent intervals within D_{e_i}. The discretization step $Step_{e_i}$ allows us to handle temporal information with different granularities as in [5, 6]. Note that $EarliestStart_{e_i}$, $LatestEnd_{e_i}$, $Duration_{e_i}$ and $Step_{e_i}$ are natural numbers expressed as constants, variables defined over specific domain or as arithmetic expressions including natural variables and arithmetic operators ('+', '−', '∗' or '/'). See Fig. 3 for an example.

$X = \{x_1, \ldots, x_m\}$ is the finite set of composite variables.

$D_X = \{D_{x_1}, \ldots, D_{x_m}\}$ is the set of domains of the composite variables. Each domain D_{x_i} is the set of possible events the composite variable x_i can take.

IV is the set of initial variables. An initial variable can be a composite variable or an event. $IV \subseteq E \cup X$.

$C = \{C_1, \ldots, C_p\}$ is the set of *compatibility constraints*. Each compatibility constraint is a qualitative temporal relation between two variables in case the two variables are events, or a set of qualitative relations if at least one of the two variables involved is composite. A qualitative temporal relation is a disjunction of Allen primitives [2].

Act is the set of *activity constraints*. Each activity constraint has the following form: $X_1 \wedge \cdots \wedge X_p \overset{condition}{\rightarrow} Y$ where $X_1, \ldots, X_p$ and Y are temporal variables (composite or events). This activity constraint will activate Y if

$X_1, \ldots, X_p$ are active and *condition* holds on these variables. *condition* corresponds here to the assignment of particular values to the variables $X_1, \ldots, X_p$.

Let us illustrate the CCTCSP through the following example.

Example 1

John, Mike and Lisa are going to see a movie on Friday. John will pick Lisa up and Mike will meet them at the theatre. If John arrives at Lisa's before 7:30, then they will stop at a convenient store to get some snacks and pops. It will take them 35 minutes to reach the theater if they stop at the store and 15 minutes otherwise. There are three different shows playing: $movie_1$, $movie_2$ and $movie_3$. If they finish the movie by 9:15, they will stop at a Pizza place 10 minutes after the end of the movie and will stay there for 30 minutes. John leaves home between 7:00 and 7:20. Lisa lives far from John (15 minutes driving). Mike leaves home between 7:15 and 7:20 and it takes him 20 minutes to go to the theater. $movie_1$, $movie_2$ and $movie_3$ start at 7:30, 7:45 and 7:35 and finish at 9:00, 9:10 and 9:20 respectively.

Table 1 Allen primitives

Relation	Symbol	Inverse	Meaning
X Precedes Y	B	Bi	X Y
X Equals Y	E	E	X / Y
X Meets Y	M	Mi	X Y
X Overlaps Y	O	Oi	X Y
X During Y	D	Di	X Y
X Starts Y	S	Si	X Y
X Finishes Y	F	Fi	Y X

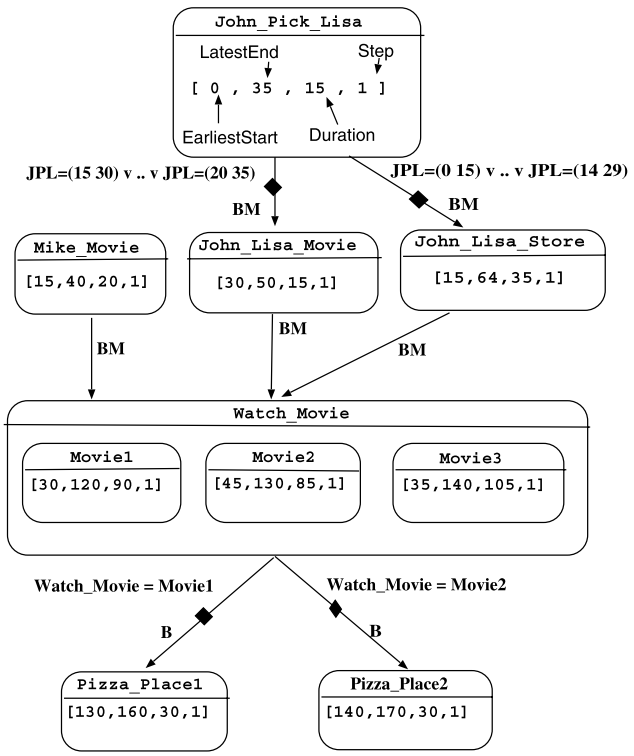


Fig. 1 CCTCSP of example 1

The goal here is to check if this story is consistent (has a feasible scenario). The story can be represented by the CCTCSP in Fig. 1. Each event domain is represented by the fourfold [*EarliestStart*, *LatestEnd*, *Duration*, *Step*].

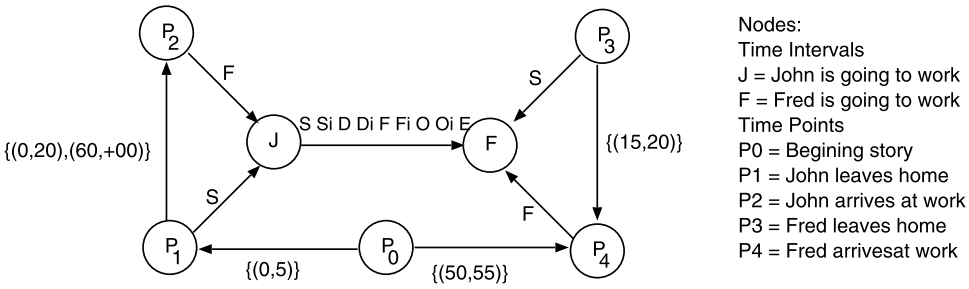


Fig. 2 Meiri's Temporal Network [29] of example 2

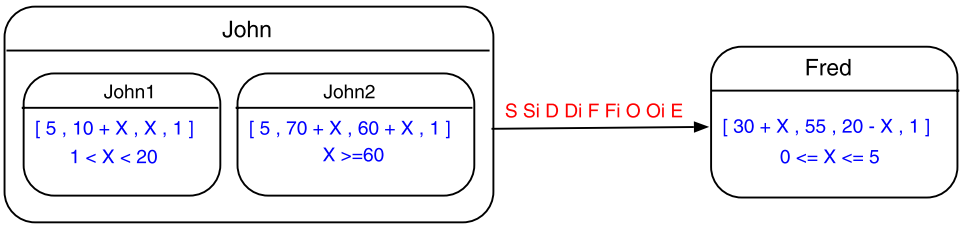


Fig. 3 CCTCSP of example 2

In the case of the event *John_Pick_Lisa*, the domain is [0, 35, 15, 1] where 0 (the time origin corresponding to 7:00) is the earliest start time, 35 is the latest end time, 15 is the duration, and 1 (corresponding to 1 min) is the discretization step. This domain corresponds to the following set of intervals: {[0, 15], [1, 16], . . . , [20, 35]}. For the sake of simplicity all the events in this story have the same step. Arcs represent either a compatibility constraint or an activity constraint (case of arcs with diamond) between variables. The compatibility constraint is denoted by one or more qualitative relations. The activity constraint shows the condition to be satisfied and the qualitative relation between the two variables in case the condition is true. Each qualitative relation is a disjunction of some Allen primitives [2]. For example, the relation *BM* between *John_Pick_Lisa* and *John_Lisa_Store* denotes the disjunction *Before* $\vee$ *Meets*. The JPL (*John_Pick_Lisa*) related activity constraint shown in Fig. 1 expresses the fact that if JPL ending time is before or equal 30 (7:30) then the event *John_Lisa_Store* is activated, otherwise (if JPL starting time is after 20) *John_Lisa_Movie* is activated. This activity constraint is a representation of the sentence "*If John arrives at Lisa's before 7:30, then they will stop at a convenient store to get some snacks and pops*".

In order to compare our model to Meiri's general framework for representing qualitative and quantitative temporal constraints [29] we will see how the following example (taken from [29]) is represented by both models.

Example 2

John and Fred work for a company that has local and main offices in Los Angeles. They usually work at the local office, in which case it takes John less than 20 minutes and Fred 15–20 minutes to get to work. Twice a week John works at the main office, in which case his commute to work takes at least 60 minutes. Today John left home between 7:05–7:10 a.m., and Fred arrived at work between 7:50–7:55 a.m. We also know that Fred and John met at a traffic light on their way to work.

Using Meiri's model the above story can be represented by the constraint network of Fig. 2 and taken from [29]. In the graph, nodes represent time instants (case of P_0, P_1, P_2, P_3 and P_4) or time intervals (J and F) while arcs are labeled with qualitative or quantitative constraints. A qualitative constraint is a binary relation between two nodes each of which may correspond to a time instant or an interval. Indeed, Meiri has defined three types of qualitative constraints: Interval–Interval (II) relations (case of the constraint (J, F)), Point–Interval PI) relations (case of the constraint (P_2, J)) and Point–Point (PP) relations. A quantitative constraint (case of the relation (P_1, P_2)) represents the distance between time points and is represented by an interval set in the same way as in the well known TCSP [12]. Using our CCTCSP we represent the story in example 2 with the graph in Fig. 3. Note that, on the graph, the variable X is a natural number. Thus, the two temporal networks respectively in Figs. 2 and 3, are not completely equivalent. Indeed, metric information are continuous in Meiri's model and discrete in ours.

From example 2 we can see that our CCTCSP can represent any temporal problem that can be modeled with Meiri's framework. In addition, CCTCSP has the ability to deal with temporal information that are added to the problem dynamically via activity constraints and composite variables.

3 Constraint propagation for solving CCTCSPs

Different methods for solving conditional CSPs have been reported in the literature [15, 16, 31, 38]. In [16], all possible CSPs are first generated from the CCSP to solve. CSP techniques are then used on the generated CSPs in order to look for a possible solution. Dependencies between the activity constraints are considered in order to generate a directed acyclic graph (DAG), where the root node corresponds to the set of initially active variables. Activity constraints are applied during the derivation of one total order from the partial order given by the resulting DAG. In [31, 38] resolution methods have been proposed and are directly applied on CCSPs. Maintaining arc consistency (MAC) is used to prune inconsistent branches by removing inconsistent values during the search [38]. The solving method starts by instantiating the active variables. For each active variable instantiation, the algorithm first checks the compatibility constraints and then activates the activity constraints. The method will then enforce look-ahead consistency (through arc consistency) along the compatibility constraints and prunes inconsistent values from the domains of future variables. When activity constraints come into play, newly activated variables are added to the set of future variables. MAC is then applied to the set of all active variables. In [15, 38], a CCSP is reformulated into an equivalent standard CSP. A special value "null" is added to the domains of all the variables which are not initially active. A variable instantiation with "null" indicates that the variable does not participate in the problem resolution. The CCSP is transformed into a CSP by including the "null" values. The disadvantage is that, in a large constraint problem, all variables and all constraints are taken into account simultaneously even if some are not relevant to the problem at hand. In the above methods, backtrack search is used for both the generation of possible CSPs and the search for a solution in each of the generated CSPs. Thus, these methods require an exponential time for generating the different CSPs and an exponential time for searching a solution in each generated CSP. Moreover these methods perform the propagation through activity constraints (while it is done through compatibility constraints in our case). The other problem of the above methods (since the goal is to look for all solutions) is the redundant work done when checking at each time the consistency of the same set of variables (subset of a given generated CSP). Our aim in this paper is to check for the consistency of the CCTCSP and to return one solution in case this latter is consistent.

The goal of the constraint propagation method we propose for solving CCTCSPs is to overcome, in practice, the difficulty due to the exponential search space of the possible TCSPs generated by the CCTCSP to solve and also the search space we consider when solving each TCSP. In the same way as reported in [31, 38], we use constraint propagation in order to detect earlier later failure. This will allow us to discard at the early stage any subset containing conflicting variables. The description of the method we propose is as follows.

1. The method starts with an initial problem containing a list of initially activated temporal events and composite variables. Arc consistency is applied on the initial temporal events and composite variables in order to reduce some inconsistent values which will reduce the size of the search space. If the temporal events are not consistent (in the case of an empty domain) then the method will stop. The CCTCSP is inconsistent in this case.
2. Following the Forward Check (FC) principle [21], pick an active variable v, assign a value to it and perform arc consistency between this variable and each of the unassigned active variables. If one domain of the non assigned variables becomes empty then assign another value to v or backtrack to the previously assigned variable if there are no more values to assign to v. Activate any variable v' resulting from this assignment and perform arc consistency between v' and all the active variables. If arc inconsistency is detected then deactivate v' and choose another value for v (since the current assignment of v leads to an inconsistent CCTCSP). If v is a composite variable then assign an event to it (from its domain). Basically, this consists of replacing the composite variable with one event *evt* of its domain. We then assign a value to *evt* and proceed as shown before except that we do not backtrack in case all values of *evt* are explored. Instead, we will choose another event from the domain of the composite variable v or backtrack to the previously assigned variable if all values (events) of v have been explored. This process will continue until all the variables are assigned in which case we obtain a solution to the CCTCSP. The arc consistency in the above two steps is enforced as shown in the four cases below. We will assume in the following that evt_1 and evt_2 are two events while x_1 and x_2 are two composite variables.

   (a) The constraint is (evt_1, evt_2). Arc consistency [27] is applied here i.e. each interval a of evt_1 should have a support in the domain of evt_2.
   (b) The constraint is (evt_1, x_1). Each interval a, from the domain of evt_1, should have a support in at least one domain of the variables within x_1.
   (c) The constraint is (x_1, evt_1). Each interval a, from the domain of every event *evt* within x_1, should have a support in the domain of evt_1.
   (d) The constraint is (x_1, x_2). Apply case (b) between each interval *evt* within x_1, and x_2.

Using the above rules, we have implemented a new arc consistency algorithm for CCTCSPs as shown in Fig. 4.

```
REVISE(D_i, D_j)
    REVISE ← false
    for each value a ∈ D_i do
        if not compatible(a, b) for any value b ∈ D_j then
                remove a from D_i
                REVISE ← true
        end if
    end for
    return REVISE

REVISE_COMP (D_i, D_j)
  REVISE_COMP ← false
  if i is an event and j is a composite variable (case b)
    D ← ∅
    D_tmp ← D_i
    for each event k ∈ D_j do
      REVISE(D_i, D_k)
      D ← D ∪ D_i
      D_i ← D_tmp
    end for
    D_i ← D
    if D_i ≠ D_tmp
      REVISE_COMP ← true
    end if
  end if
  if i is a composite variable and j is an event (case c)
    for each event k ∈ D_i do
      REVISE_COMP ← REVISE(D_k, D_j)
    end for
  end if
  if both i and j are composite variables (case d)
    for each event k ∈ D_i do
      REVISE_COMP(D_k, D_j)
    end for
  end if
  return REVISE_COMP

AC − 3 − CCTCSP
  Given a CCTCSP ⟨E, D_E, X, D_X, IV, C, Act⟩
  i, j and k are variables defined on D_i, D_j and D_k
  respectively Q ← {(i, j)|(i, j) ∈ C}
  while Q ≠ Nil do
    Q ← Q − {(i, j)}
    if i or j is a composite variable (case b, c or d)
      if REVISE_COMP(D_i, D_j) then
        Q ← ∪{(k, i)|(k, i) ∈ C and k ≠ j}
      end if
    else (both i and j are events (case a))
      if REVISE(D_i, D_j) then
        Q ← ∪{(k, i)|(k, i) ∈ C and k ≠ j}
      end if
    end if
  end while
```

Fig. 4 AC-3 for CCTCSPs

This algorithm is an extension of the well known AC-3 procedure [27]. We call it AC-3-CCTCSP. Like AC-3, AC-3-CCTCSP starts with a list of pairs of variables to revise (all the pair of variables sharing a constraint) and goes through this list until this latter is empty. Each pair (i, j) is then processed (revised) according to the above 4 cases as follows.

Case (a) Both i and j are events. Here we apply the traditional REVISE function of AC-3 [27].

Case (b) i is an event and j is a composite variable. As we stated in rule (b) above, each value a of D_i (where D_i is the domain of event i) should have a support in at least one domain D_k (where k is an event within the composite variable j). In other words, a is removed from D_i if it does not have any support in any domain D_k. This is implemented by computing the union of the sets respectively obtained by revising D_i with each of the events within j.

Case (c) i is a composite variable and j is an event. The function REVISE is applied here on each pair of events (k, j) where k is an event within i.

Case (d) Both i and j are composite variables. Here we apply case (b) on each event within i, and j.

Like for general CSPs, variable and value ordering, during search, has a significant impact on the size of the explored space in the case of CCTCSPs. For variable selection, we will follow the idea of choosing the most constrained variable first in the hope of triggering early failure. In the case of value selection, we start with the value that leads to an easiest to solve CCTCSP first since our goal here is to find the first solution and that there is no preference on the solution obtained. More precisely, our variable and value selection policy works as follows.

1. The variables (simple and composite variables) are selected by decreasing order of the number of constraints they share with other variables. For a given variable x, this number (that we call degree of a variable) corresponds to the node degree (number of edges connected to the node) of the node corresponding to x in the constraint graph.
2. The degree of a composite variable x is equal to the minimum variable degree of all the variables (events) within its domain D_x.
3. If a variable x can activate other variables then we add to its variable degree, the minimum number of constraints it can generate (activate) through the activity constraints.
4. For value selection, in the case of a composite variable x, select the simple variables, within the domain of x, by decreasing number of their degrees.
5. For an event, select the least constrained value first (the value that causes the activation of the minimum number of constraints).

Note that preliminary tests we have conducted on randomply generated CCTCSPs clearly demonstrated the importance of using the above policy. Indeed, for CCTCSPs with more than 40 variables the constraint propagation

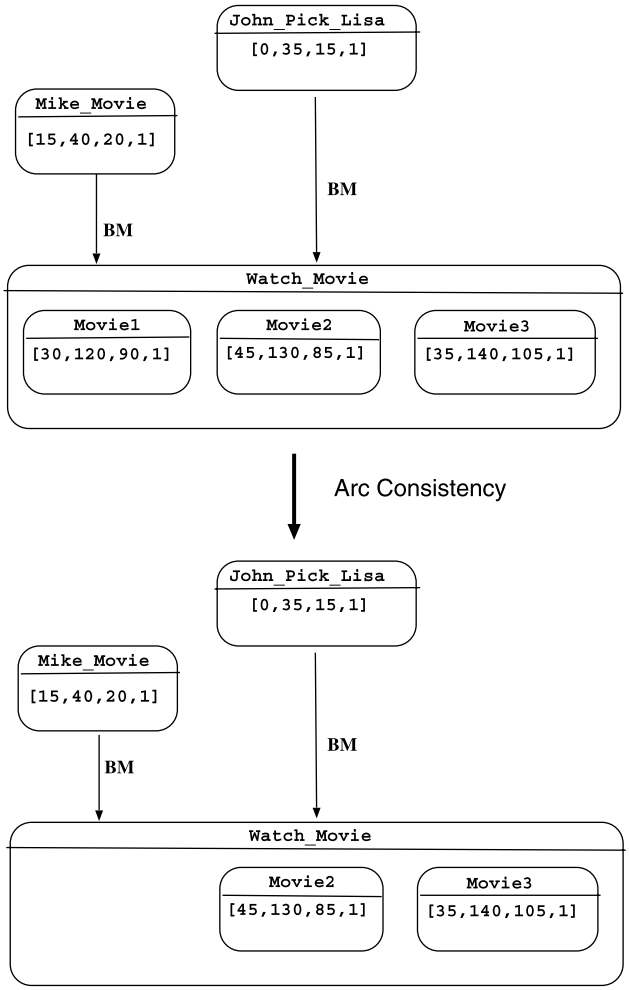


Fig. 5 Arc consistency on the initial CCTCSP of example 1

method always fails to solve the problem, while the solution is returned within a reasonable time (as shown in Sect. 5) in case the variable|value policy is considered.

Example 3

In the following we will apply our constraint propagation method to solve the problem presented in example 1. The initial problem containing the initially active variables is presented in the top graph of Fig. 5. When applying the arc consistency algorithm to this initial CCTCSP (phase 1 of our method) the event $Movie_1$ of the composite variable *Watch_Movie* will be removed as it does not have any interval that is arc consistent with the values of the event *Mike_Movie*. The bottom graph of Fig. 5 shows the initial CCTCSP after enforcing the arc consistency. *John_Pick_Lisa* will then be selected and assigned the first interval from its domain: (0 15). This will activate *John_Lisa_Store* as shown in the top left graph of Fig. 6. Applying the arc consistency between this newly activated event and all the other active variables will result in an empty domain of the composite variable *Watch_Movie*. Indeed none of the two events of this latter composite variable has an interval that is arc consistent with the values of the event *John_Lisa_Store* (since the earliest end time of *John_Lisa_Store* is greater than the earliest start time of both $Movie_2$ and $Movie_3$). *John_Lisa_Store* will then be deactivated and the next value to assign to *John_Pick_Lisa* is (1 16). This latter assignment and all the others until (14 29) will fail in the same way. (15 30), the first interval that does not activate *John_Lisa_Store*, will then be chosen. This will result in the activation of the event *John_Lisa_Movie*. When applying arc consistency between this new event and the other active variables, the event $Movie_3$ will be removed as illustrated in the top right graph of Fig. 6. After assigning the values (30 45), (15 35) and $Movie_2$ respectively to *John_Lisa_Movie*, *Mike_Movie* and *Watch_Movie* we will obtain the CCTCSP in the bottom right of Fig. 6. Finally the consistent solution is obtained as shown in the bottom left graph of Fig. 6.

Note that, in addition to the Forward Check (FC) principle we described earlier in phase 2 of our constraint propagation method, we have explored other propagation strategies. More precisely we are considering the following four techniques.

1. FC. As we mentioned earlier, this strategy consists of checking the arc consistency, during the search, between the current variable (the variable that we are assigning a value) and each of the future active variables (active variables that are not assigned yet) sharing a constraint with the current variable.
2. Maintaining Arc Consistency (MAC). This strategy maintains a full arc consistency on the current and future active variables.
3. FC+. Same as FC except that the applicability of arc consistency is extended to inactive variables as well. This means that the arc consistency is performed between the current variable and each of the future active and non active variables sharing a constraint with the current one.
4. MAC+. Same as MAC except that the applicability of the arc consistency is extended to inactive variables as well. This means that the full arc consistency is enforced on the current and the future variables (active and inactive).

4 Approximation methods for CCTCSPs

The method we presented in the previous section is an exact technique that guarantees a complete solution. In many real-life applications where the execution time is an issue, an alternative will be to trade the execution time for the

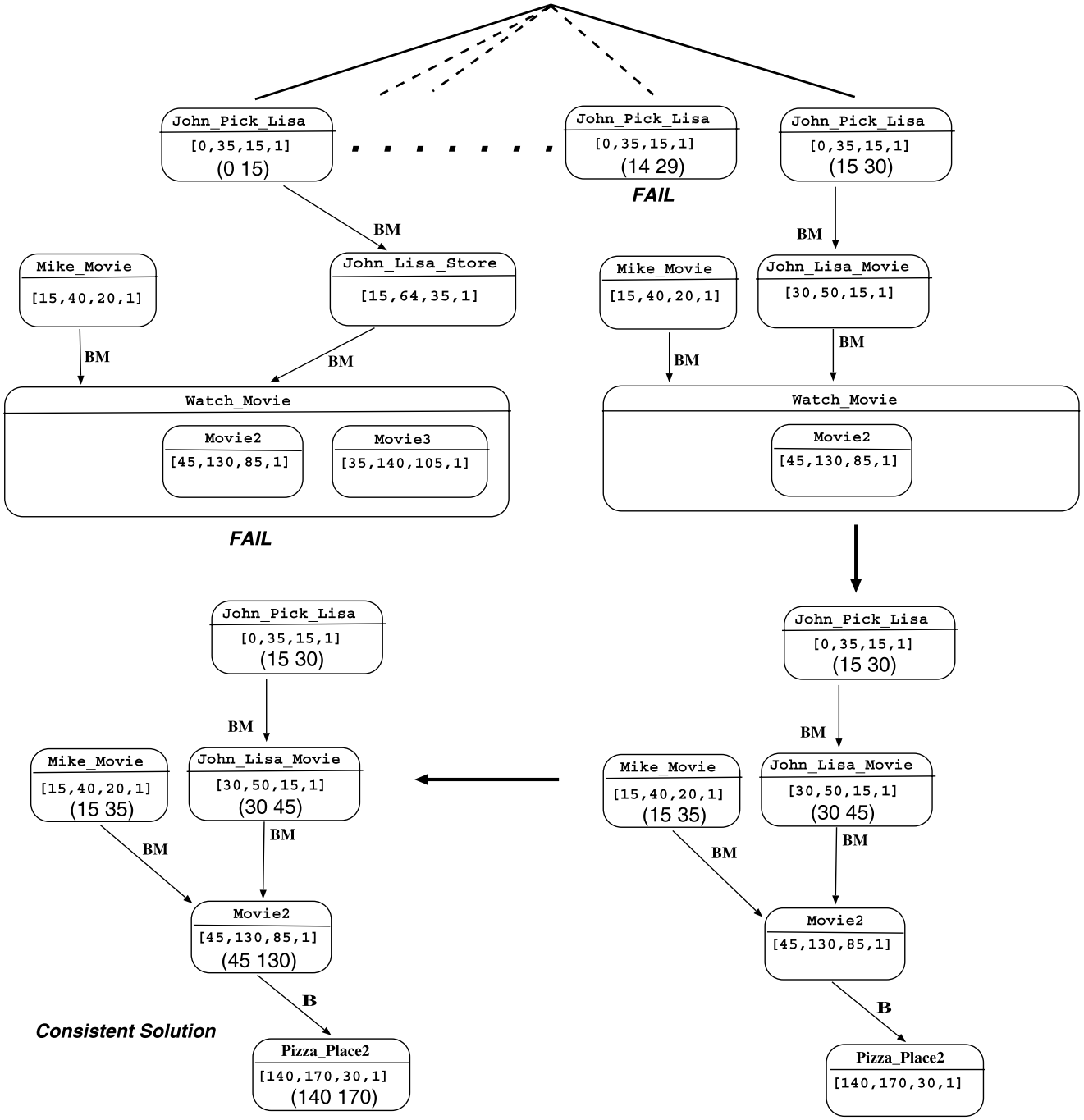


Fig. 6 Constraint propagation for the CCTCSP of example 1

quality of the solution returned (number of solved constraints). This can be done by applying approximation methods such as local search and where the quality of the solution returned is proportional to the running time (assuming that the local search method will not be trapped in a local optimum). In this section we will study the applicability of the following local search techniques for solving CCTCSPs: Min-Conflict-Random-Walk (MCRW), Steepest-Descent-Random-Walk (SDRW) and Tabu Search (TS) [39]. In the following we will show how each of these methods is applied on CCTCSPs.

4.1 MCRW-CCTCSP

MCRW has already been applied to solve TCSPs [34]. Basically, this method starts from a complete assignment of temporal intervals to events and iterates by improving at each

step the quality of the assignment (number of solved constraints) until a complete solution is found or a maximum number of iterations is reached. Given the dynamic aspect of CCTCSPs (some variables are added|removed dynamically during the resolution process) we propose the following algorithm based on MCRW for solving CCTCSPs. This algorithm is called MCRW-CCTCSP.

1. The algorithm starts with a random assignment of values to the initial variables. If the initial variable is an event then it will be randomly assigned a temporal interval from its domain. In the case where the initial variable is composite then it will be replaced by one variable selected randomly from its domain. This latter variable will then be randomly assigned a temporal interval from its domain.
2. Activate any variable where the activating condition is true and randomly assign to it a value from its domain as shown in the previous step.
3. If a complete solution is not found and the maximum number of iterations is not reached, randomly select an active variable v involved in a conflict and proceed with one of the following cases:
   - If v belongs to the domain of a given composite variable X then select the pair $\langle v_i, int_{v_i} \rangle$ that increases the quality of the current solution (number of solved constraints). v_i belongs here to the domain of X and int_{v_i} is a value of v_i's domain,
   - otherwise, assign to v a value that increases the quality of the solution.

   In the above two cases, if there is no value to increase the quality of the solution then the method picks randomly one value that does not increase the number of violated constraints (the current value of the event is picked only if all the other values increase the number of violated constraints).
4. Deactivate any variable activated by the old assignment of v and goto 2.

The problem of the above method is that it can easily be trapped in a local-minimum. In order to prevent that we use the random-walk strategy as follows. In step 3 of the method, the best value is chosen for the event with probability $1 - pWalk$ while a random value is assigned to the event with probability $pWalk$. Note that, to improve the performance of MCRW-CCTCSP, if we have more than one assignment that increases the quality of the solution (in step 3 above) then we choose the one that does not cause the deactivation of variables. Indeed, as demonstrated by the test results of the next section, variable deactivation affects dramatically the performance of the search as it will result in a change of the CCTCSP which forces MCRW to restart the search with a new one. Figure 7 illustrates the pseudo-code of MCRW-CCTCSP.

```
MCRW − CCTCSP(maxMoves, pWalk)
1.    Given a CCTCSP ⟨E, D_E, X, D_X, IV, C, Act⟩
2.    S ← randomAssign(IV)
3.    activate non active variables if the condition is true
4.    newActiveVar = list of new active variables
5.    IV ← IV ∪ newActiveVar
6.    newS ← randomAssign(newActiveVar)
7.    S ← S ∪ newS
8.    nbMoves ← 0
9.    while eval(S) > 0 and nbMoves < maxMoves
10.   choose randomly an active variable v involved
       in a conflict
11.     randomly pick rWalk from [0...1]
12.     if rWalk ≤ pWalk then
13.       assignment ← randomAssign({v})
14.     else
15.       assignment ← bestAssign({v})
16.     end if
17.   insert(S, assignment)
18.   nbMoves ← nbMoves + 1
19.   end while

randomAssign(listVar)
1.    assignment ← NIL
2.    while (listVar ≠ NIL)
3.      pick v from listVar
4.      if v ∈ E (v is an event)
5.        randomly pick intv from D_v (D_v ∈ D_E)
6.        assignment ← assignment ∪ ⟨v, intv⟩
7.      else (v is a composite variable)
8.        randomly pick vx from D_v (D_v ∈ D_E)
9.        randomly pick intv from D_vx (D_vx ∈ D_E)
10.       assignment ← assignment ∪ ⟨vx, intv⟩
11.     end if
12.   end while
13.   return assignment

bestAssign(listVar)
1.    assignment ← NIL
2.    while (listVar ≠ NIL)
3.      pick v from listVar
4.      if v ∈ E (v is an event)
5.        pick intv from D_v (D_v ∈ D_E) that minimizes
          the number of conflicts for v
6.        assignment ← assignment ∪ ⟨v, intv⟩
7.      else (v is a composite variable)
8.        pick the pair ⟨vx, intv⟩ that minimizes
          the number of conflicts for v
9.        (vx ∈ D_v where D_v ∈ D_E
10.       (intv ∈ D_vx where D_vx ∈ D_E
11.       assignment ← assignment ∪ ⟨vx, intv⟩
12.     end if
13.   end while
14.   return assignment
```

Fig. 7 MCRW-CCTCSP

Example 4

Let us illustrate our MCRW-CCTCSP method through the example 1. The first step is to randomly assign values to

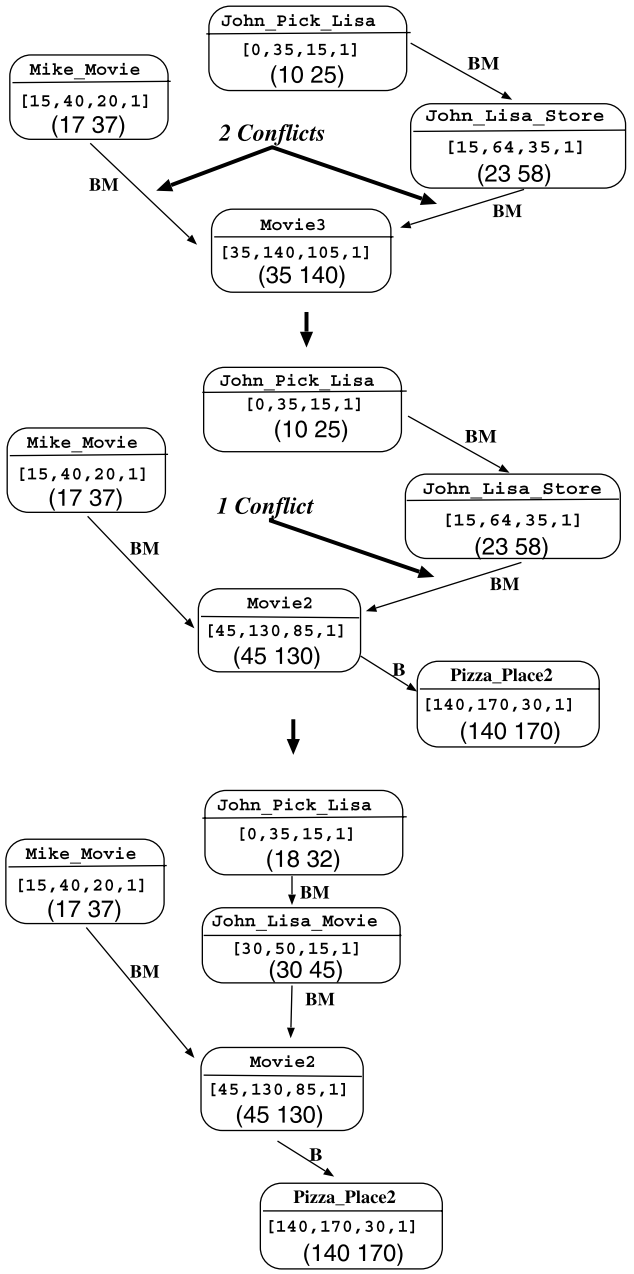


Fig. 8 MCRW for the CCTCSP of example 1

the initial variables and then activate (and randomly assign a value to) any variable where the activating condition is true. The result is shown in the top graph of Fig. 8. There are two conflicts here and the composite variable *Watch_Movie* is selected and assigned the value $Movie_2$ since this will reduce the number of conflicts to 1 as we can see in the middle graph of Fig. 8.[4] The variable *John_Pick_Lisa* is then selected and assigned the value (18 32) since this will reduce the number of conflicts to 0 and a complete solution is obtained in this case as shown in the bottom graph of Fig. 8.

[4]Note that we could as well select one of the other 2 variables in conflicts: *Mike_Movie* and *John_Lisa_Store*. However this latter choice will not reduce the number of conflicts.

4.2 SDRW-CCTCSP

SDRW-CCTCSP is similar to MCRW-CCTCSP with the following difference. Instead of selecting the variable in conflict randomly as it is the case in step 3 of MCRW-CCTCSP (see Fig. 8), SDRW-CCTCSP algorithm explores the whole neighborhood of the current configuration and selects the best neighbor (neighbor with the best quality). A neighbor of a given configuration (or complete assignment) is obtained from this latter by changing one variable value. The neighborhood of a given configuration is the set of all its possible neighbors. SDRW-CCTCSP is randomized by using the random-walk strategy in the same manner as for MCRW-CCTCSP to avoid getting stuck at local optima. The pseudo-code of SDRW-CCTCSP is the same as the one of MCRW-CCTCSP illustrated in Fig. 7 with the following slight difference. In line 15 of $MCRW - CCTCSP$ function, instead of passing the variable v to the function $bestAssign()$ we pass the entire set of active variables. More precisely line 15 will be as follows.

15. $ListActive \leftarrow$ list of active variables in conflict.
15bis $assignment \leftarrow bestAssign(\{ListActive\})$

4.3 Tabu-CCTCSP

Tabu-CCTCSP is based on the notion of Tabu list used to maintain a selective history, composed of previously encountered configurations in order to prevent Tabu from being trapped in short term cycling and allows the search process to go beyond local optima. In each iteration of the algorithm, a couple $\langle variable, value \rangle$ that does not belong to the Tabu list and corresponding to the best performance is selected and considered as an assignment of the current configuration. $\langle variable, value \rangle$ will then replace the oldest move in the Tabu list. More precisely, the pseudo-code of the function Tabu-CCTCSP is described in Fig. 9.

5 Experimentation

5.1 Comparison criteria

The goal of the tests reported in this section is to evaluate and compare the performance for solving randomly generated CCTCSPs using the methods we presented in the previous two sections. More precisely we will first compare the following four constraint propagation strategies we described in Sect. 3 (FC, FC+, MAC and MAC+).

Tabu − CCTCSP(*maxMoves*)
1. Given a CCTCSP $\langle E, D_E, X, D_X, IV, C, Act\rangle$
2. $S \leftarrow randomAssign(IV)$
3. activate non active variables if the activation condition is true
4. $newActiveVar =$ list of new active variables
5. $IV \leftarrow IV \cup newActiveVar$
6. $newS \leftarrow randomAssign(newActiveVar)$
7. $S \leftarrow S \cup newS$
8. $nbMoves \leftarrow 0$
9. $tabuList \leftarrow nil$
10. while $eval(S) > 0$ and $nbMoves < maxMoves$
11. $listActive \leftarrow$ list of active variables
12. do
13. $assignment \leftarrow bestAssign(listActive)$
14. while *assignment* contains a variable $\in listActive$
15. $insert(S, assignment)$
16. $nbMoves \leftarrow nbMoves + 1$
17. remove the oldest assignment from *tabuList*
18. $tabuList \leftarrow tabuList \cup assignment$
17. end while

Fig. 9 Pseudo-code of the Tabu Search method

In the second type of experiments we will compare the best of the above complete methods with the three local search methods covered in Sect. 4.

All the experiments are performed on a PC Pentium 4 computer running Linux. All the procedures are coded in C/C++.

5.2 CCTCSP instances

CCTCSPs are build from TCSPs randomly generated using the model RB proposed in [51]. This model is a revision to the standard Model B [17, 41], has exact phase transition and the ability to generate asymptotically hard instances. Following the model RB, we generate each TCSP instance in two steps as shown below and using the parameters n, p, α and r where:

- n is the number of events,
- p $(0 < p < 1)$ is the constraint tightness which can be measured, as shown in [36], as the fraction of all possible pairs of intervals from the domain of two events that are not allowed by the constraint,
- and r and α $(0 < r,\ \alpha < 1)$ are two positive constants.

1. Select with repetition $rn\ln n$ random constraints.[5] Each random constraint is formed by selecting without repetition 2 of n events.
2. For each constraint we uniformly select without repetition pd^2 incompatible pairs of intervals from the domains of the pair of events involved by the constraint. $d = n^\alpha$ is the domain size of each event.

Each CCTCSP instance is then generated as follows using the parameters N, D, I and a which respectively denote the number of composite variables, their domain size (number of events within each composite variable), the percentage of variables that are initially active and the density of activity constraints.

1. Randomly generate a TCSP with the parameters n, p, α and r as shown above.
2. Generate N composite variables each containing D events.
3. Select with repetition $r[(n+N)\ln(n+N) - n\ln n]$ new random constraints (between the $n+N$ composite variables and events), each formed by selecting without repetition 2 of the $n+N$ variables. This will guarantee that the total number of constraints is $r(n+N)\ln(n+N)$ (as per the requirements of the RB model). Each selected constraint C_{ij} involving two variables X_i and X_j is then generated following one of the procedures below.
   (a) If both X_i and X_j are events then we uniformly select without repetition pd^2 incompatible pairs of intervals from the domains of X_i and X_j.
   (b) If X_i is composite and X_j is an event (or vise versa) then the constraint will be a disjunction of D relations between the event X_j and each event within X_i's domain. Each of these D relations will be generated as shown above in (a).
   (c) If both X_i and X_j are composite then the constraint will be a disjunction of D^2 relations between the pair of events from X_i and X_j domains. Each of these D^2 relations will then be generated as shown above in (a).
4. Select $I(n+N)$ initial variables from $n+N$ $(0 < I < 1)$.
5. Select $a * nbActivity$ activity constraints where $0 < a < 1$. Here *nbActivity* is the number of possible activity constraints. For simplicity, the generated activity constraints have the following form $(X_i = val) \rightarrow X_j$ (where *val* belongs to X_i's domain) which is less general then the definition we have provided in Sect. 2. Thus, the total number of possible activity constraints is equal to $(nd + ND) * (n + N - I(n+N))$.

As demonstrated in [51], when the number of variables approaches infinity the phase transition occurs when the constraint tightness $p = 1 - e^{-\frac{\alpha}{r}}$. Thus the phase transition is an asymptotic phenomenon since, only for infinite number of variables, we can have sharp phase transitions. In addition, the number of variables and constraints of the possible CSPs, each CCTCSP contains, is slightly different from the one of the CCTCSP they are generated from.

[5] According to the RB model, in order to be able to compute accurately the phase transition, the number of constraints should be equal to $rn\ln n$ [51].

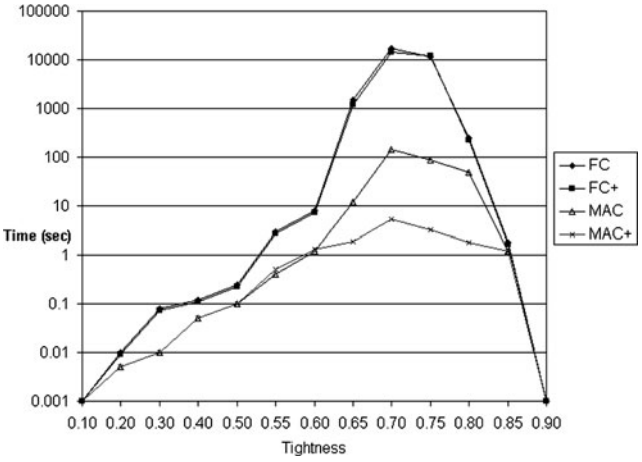


Fig. 10 Comparative tests when varying the tightness p

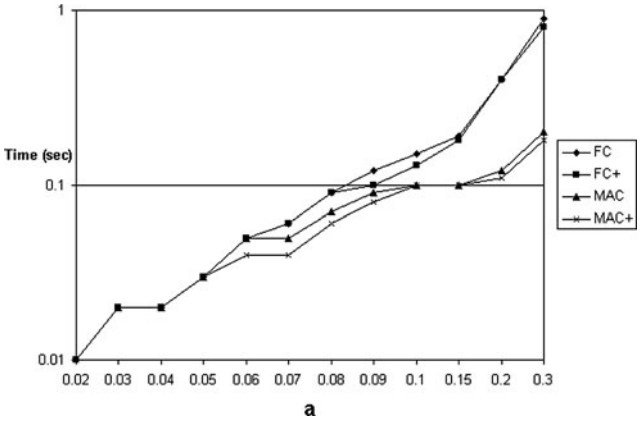


Fig. 11 Comparative tests when varying a (% of possible activity constraints)

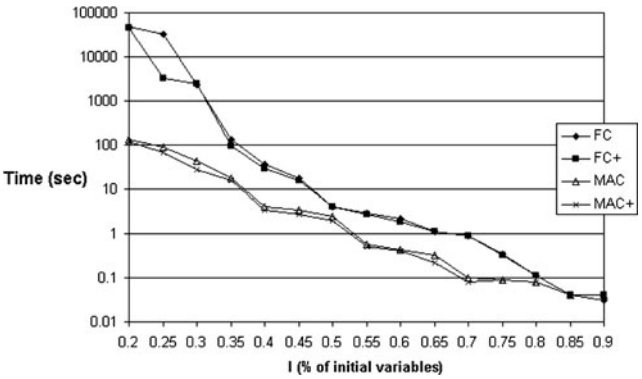


Fig. 12 Comparative tests when varying I

5.3 Comparing complete methods

In the following we will present the results of tests performed by each of the four strategies described in Sect. 3 on several CCTCSP sets. Each set is generated by varying one of the following parameters: p, a, I, N, D, α and r. For each test, each of the four methods is executed on 100 instances and the average running time in seconds is taken.

5.3.1 Easy versus hard problems: varying p

Here the parameters are set as follows. $n = 140$, $N = 10$, $D = 5$, $\alpha = 0.8$, $I = 0.8$, $a = 0.2$ and $r = 0.6$. p varies from 0.2 to 0.9 (in order to consider under and over constrained problems as well as those near the phase transition). As mentioned before, according to the RB model, the phase transition is computed as follows: $p = 1 - e^{-\frac{\alpha}{r}} = 1 - e^{-\frac{0.8}{0.6}} = 0.73$. In practice it is around 0.7 as we can see from the test results shown in Fig. 10. In the case of over constrained problems (tightness greater than 0.8), all the four methods have comparable running time. This is because the inconsistency is detected in the first stage of the resolution method we provided in Sect. 3 (corresponding to the application of the AC3-CCTCSP (see Fig. 4) to the initial problem). Indeed, the first stage is the same for all the four methods. All the methods have also similar running times in the case of under constrained problems. Indeed, in this particular case the extra effort done by MAC and MAC+ does not remove much of the inconsistent values and thus does not improve the overall running time to find a solution. However, when we move toward the phase transition the extra work performed by MAC and especially MAC+ starts to pay off. At the phase transition MAC+ is almost 10,000 times faster than FC and FC+; and 100 times faster than MAC. Note that the superiority of MAC over FC for hard CSPs has already been noticed and reported in [36]. Also, considering inactive variables during the propagation through MAC+ does help a lot on the complexity peaks. Indeed, at the phase transition more search space is explored through the backtrack and many of the inactive variables are activated. Reducing the domains of these latter is thus very relevant in this particular situation.

5.3.2 More versus less dynamic problems: varying a, I, N and D

Figure 11 reports the results of comparative tests conducted when the parameter a varies from 0.02 to 0.3. The other parameters are fixed as follows: $n = 50$, $N = 10$, $D = 5$, $p = 0.5$, $\alpha = 0.8$, $I = 0.8$, $r = 0.6$.

Figure 12 reports the results of comparative tests conducted when the parameter I varies from 0.2 to 0.9. The other parameters are fixed as follows: $n = 50$, $N = 10$, $D = 5$, $p = 0.5$, $\alpha = 0.8$, $a = 0.2$, $r = 0.6$. The lower the value of I is, the more "dynamic" (and diffult to solve) is the problem. In both Figs. 11 and 12 the winners are again MAC and MAC+. While MAC+ does more efforts than MAC when I decreases (since the difference between the two strategies is that MAC+ does the propagation to non active variables as well as active variables) it is the only

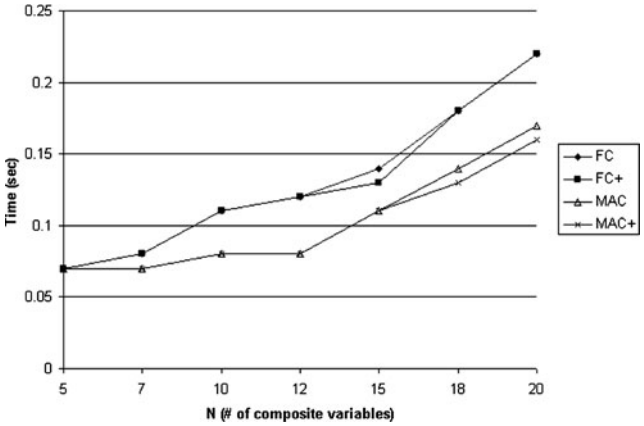


Fig. 13 Comparative tests when varying the number of composite variables

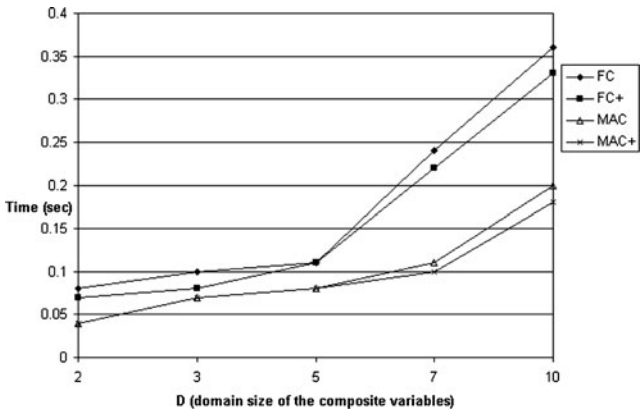


Fig. 14 Comparative tests when varying the domain size of the composite variables

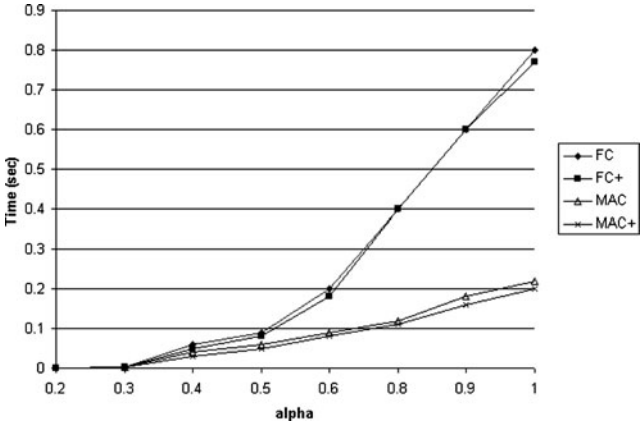


Fig. 15 Comparative tests when varying α

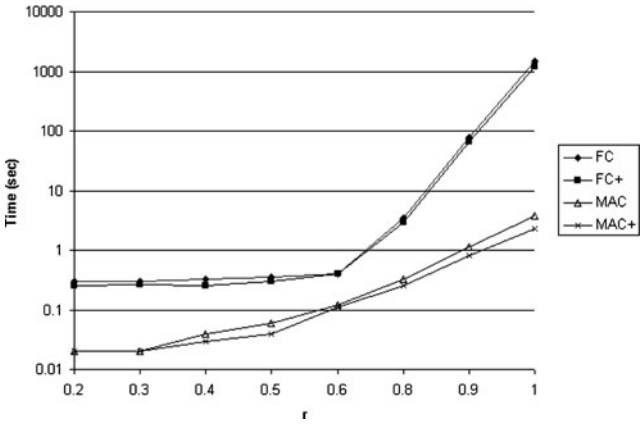


Fig. 16 Comparative tests on random CCTCSPs when varying r

method solving all the problem instances when the tightness is equal to 0.6. Indeed we have conducted other tests (that we did not report here) where $p = 0.6$. The results indicate that FC and FC+ fail to solve all the problems when $I < 0.9$.

Figure 13 reports the results of the tests when varying N. Here the number of composite variables N varies from 5 to 20. The other parameters are fixed as follows: $n = 50$, $D = 5$, $p = 0.5$, $\alpha = 0.8$, $a = 0.2$, $r = 0.6$ and $I = 0.8$. Figure 14 reports the results of the tests when varying D from 2 to 10. The other parameters are fixed as follows: $n = 50$, $N = 10$, $p = 0.5$, $\alpha = 0.8$, $a = 0.2$, $r = 0.6$ and $I = 0.8$.

When N or D is increasing, the generated problems become more dynamic. Here again, MAC and MAC+ outperform the other two methods.

5.3.3 Small versus large size problems: varying α and r

Figure 15 reports the results of the tests when varying α from 0.2 to 1. The other parameters are fixed as follows: $n = 50$, $N = 10$, $D = 5$, $p = 0.5$, $a = 0.2$, $r = 0.6$ and $I = 0.8$. Figure 16 reports the results of the tests when varying r from 0.2 to 1. The other parameters are fixed as follows: $n = 50$, $N = 10$, $\alpha = 0.8$, $D = 5$, $p = 0.5$, $a = 0.2$ and $I = 0.8$. According to the RB model, the domain size d of the events is equal to n^{α}, and the number of generated constraints is equal to $rn \ln n$. Thus, when α increases the domain size of the events becomes very large, and when r increases, the number of constraints increases and the related generated problem becomes more constrained. In both Figs. 15 and 16 MAC and MAC+ outperform FC and FC+.

5.4 Comparing complete and incomplete methods

Since MAC+ is the best method according to the previous tests, we will compare it to the local search methods we described in Sect. 4. The comparative tests are conducted on problem instances generated with the parameters fixed as shown in Sect. 5.3.1. Since MCRW-CCTCSP (respectively SDRW-CCTCSP and Tabu-CCTCSP) is an incomplete method, in case it does not find a complete solution for a given instance we report the quality (percentage of solved constraints) of the best solution obtained and the time it took

Table 2 Comparative tests on random CCTCSPs

Tightness	MCRW		SDRW		TABU		MAC+
	Time	success (%)	Time	success (%)	Time	success (%)	
0.1	0	100	0	100	0	100	0
0.2	0	100	0	100	0.02	100	0.01
0.3	0	100	0.01	100	0.02	100	0.01
0.4	0.01	100	0.02	100	0.03	100	0.05
0.5	0.08	100	0.12	100	0.13	100	0.14
0.6	0.3	80	0.44	80	0.58	80	1.3
0.7	0.4	70	1.6	68	0.9	65	5.2

to get this quality. Note also that we only consider consistent instances. More specifically we only consider those instances with tightness less than or equal to 0.7. For each of these instances we run MAC+ to check that the problem is consistent. The test results are reported in Table 2. For each test, each method is executed on 100 instances and the average running time in seconds is taken. In the case of the three SLS methods, *maxMoves* (the maximum number of iterations), *pWalk* (the random walk parameter) and the size of the tabu list are respectively equal to 10,000, 0.1 and 20. Indeed, preliminary experiments conducted on randomly generated CCTCSPs showed that these values provide the best results for the SLS methods.

As we can see in the table, MCRW outperforms SDRW and Tabu for all tightness values. This is due to the fact that these two latter methods perform more efforts at each iteration than MCRW. Indeed, as we have seen in the previous section, instead of randomly selecting one variable at each iteration (case of MCRW) SDRW and Tabu consider all the active variables. In addition, in the case of Tabu we need to reset the Tabu list anytime the constraint network changes (due to activation|deactivation of variables). We also note that MCRW is as good as MAC+ for under and middle constrained problems. Indeed, in the case of under constrained problems, for example, the solution is obtained in the case of MCRW after a couple of random assignments. However, when we approach the phase transition, the random search is affected by the change of the constraint network at each iteration. Indeed, each time an assignment is reconsidered it usually results in deactivating several variables and activating others. MCRW has then to restart with this new configuration. Note that while MCRW does not solve completely highly constrained problems, it is still a method of choice in case we want to trade search time for the quality of the solution returned. As we can see in the table, in the case where the tightness is equal to 0.7 for example, we can decide to get the incomplete solution (solving 70% of the constraints) within 0.4 seconds with MCRW instead of waiting 5.2 seconds to get a complete one. Trading search time for the quality of the solution can be very relevant for reactive and real time applications where an answer is needed within a given deadline.

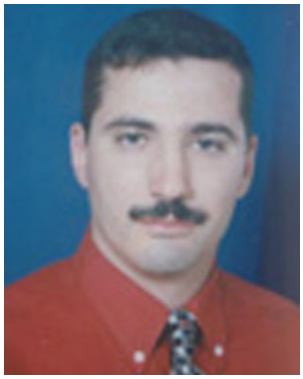
6 Conclusion

We have presented in this paper a CSP based framework for representing and managing numeric and symbolic temporal constraints, activity constraints and composite variables with a unique constraint network that we call Conditional Composite Temporal Constraint Satisfaction Problem (CCTCSP). Solving a CCTCSP consists of finding a solution for one of its possible TCSPs. When considering only composite TCSPs (CTCSPs) the solving algorithm requires $O(D^{N+M}d^M)$ time cost where N, D, M and d are respectively the number of events and their domain size, and the number of composite variables and their domain size. Adding activity constraints will make the problem even harder. In order to overcome this difficulty in practice, we have proposed 2 methods respectively based on constraint propagation and stochastic local search. Constraint propagation prevents later failure earlier which improves, in practice, the performance in time of the backtrack search especially when the propagation is performed through the MAC principle. On the other hand, due to its polynomial time cost and incompleteness, the stochastic local search method MCRW is the method of choice, for highly constrained problems, in case we decide to trade search time for the quality of the solution returned (number of solved constraints).

References

1. Alfonso MI, Barber F (2004) A mixed closure-CSP method for solving scheduling problems. Appl Intell 21(2):173–193
2. Allen JF (1983) Maintaining knowledge about temporal intervals. CACM 26(11):832–843
3. Badaloni S, Giacomin M (1999) A fuzzy extension of Allen's interval algebra. In: E. Lamma, P. Mello, (eds) Proc. of the 6th congress of the Italian assoc. for artificial intelligence, pp 228–237

4. Baptiste P, Le Pape C (1995) Disjunctive constraints for manufacturing scheduling: principles and extensions. In: Third international conference on computer integrated manufacturing. Singapore
5. Bettini C, Wang X, Jajodia S (1998) A general framework for time granularity and its application to temporal reasoning. Ann Math Artif Intell 22:29–58
6. Bettini C, Wang X, Jajodia S (2002) Solving multi-granularity temporal constraint networks. Artif Intell 140(1–2):107–152
7. Bodirsky M, Kára J (2010) The complexity of temporal constraint satisfaction problems. J Assoc Comput Mach 57(2):1–41
8. Boerkoel JC Jr., Durfee EH (2009) Evaluating hybrid constraint tightening for scheduling agents. In: The proceedings of the 8th international conference on autonomous agents and multiagent systems, pp 673–680
9. Dean T (1989) Using temporal hierarchies to efficiently maintain large temporal databases. J Assoc Comput Mach, pp 686–709
10. Dechter R, Dechter A (1988) Belief maintenance in dynamic constraint networks. In: 7th national conference on artificial intelligence, pp 37–42, St Paul
11. Dechter R (2003) Constraint processing. Morgan Kaufmann, San Mateo
12. Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artif Intell 49:61–95
13. Fargier H, Lang J, Schiex T (1996) Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In: The 13th national conference on artificial intelligence (AAAI-96), pp 175–180
14. Fisher M, Gabbay D, Vila L (2005) Handbook of temporal reasoning in artificial intelligence (foundations of artificial intelligence), Elsevier, Amsterdam. ISBN: 0444514937
15. Gelle E (1998) On the generation of locally consistent solution spaces in mixed dynamic constraint problems. PhD thesis, 1826, pp 101–140
16. Gelle E Faltings B (2003) Solving mixed and conditional constraint satisfaction problems. Constraints 8:107–141
17. Gent IP, MacIntyre E, Prosser P, Smith BM, Walsh T (1998) Random constraint satisfaction: flaws and structure
18. Ghallab M, Laruelle H (1994) Representation and control in IxTeT, a temporal planner. In: AIPS 1994:61–67
19. Golumbic C, Shamir R (1993) Complexity and algorithms for reasoning about time: a graphic-theoretic approach. J Assoc Comput Mach 40(5):1108–1133
20. Guesgen H, Hertzberg J (1993) A constraint-based approach to spatiotemporal reasoning. Appl Intell 3(1):71–90
21. Haralick RM, Elliott GL (1980) Increasing tree search efficiency for constraint satisfaction problems. Artif Intell 14:263–313
22. Hwang C, Shubert L (1994) Interpreting tense, aspect, and time adverbials: a compositional, unified approach. In: Proceedings of the first international conference on temporal logic, LNAI, vol 827, Berlin, pp 237–264
23. Jónsson AK, Frank J (2000) A framework for dynamic constraint reasoning using procedural constraints. In: *ECAI 2000*, pp 93–97
24. Jónsson AK, Frank J (2003) Constraint-based attribute and interval planning. Constraints 8(4):339–364
25. Kautz HA, Ladkin PB (1991) Integrating metric and qualitative temporal reasoning. In: AAAI'91, Anaheim, CA, pp 241–246
26. Laborie P (2003) Resource temporal networks: definition and complexity. In: Eighteenth international joint conference on artificial intelligence (IJCAI'03), pp 948–953
27. Mackworth AK (1977) Consistency in networks of relations. Artif Intell 8:99–118
28. Marin R, Cardenas M, Balsa M, Sanchez J (1997) Obtaining solutions in fuzzy constraint networks. Int J Approx Reason 16:261–288
29. Meiri I (1996) Combining qualitative and quantitative constraints in temporal reasoning. Artif Intell 87:343–385
30. Mitra D (2002) A path-consistent singleton modeling (CSM) algorithm for arc-constrained networks. Appl Intell 17(3):313–318
31. Mittal S, Falkenhainer B (1990) Dynamic constraint satisfaction problems. In: Proceedings of the 8th national conference on artificial intelligence. AAAI Press, Boston, pp 25–32
32. Morris P, Muscettola N (2000) Execution of temporal plans with uncertainty. In: AAAI 2000, pp 491–496
33. Mouhoub M, Charpillet F, Haton JP (1998) Experimental analysis of numeric and symbolic constraint satisfaction techniques for temporal reasoning. Constraints Int J 2:151–164
34. Mouhoub M (2004) Reasoning with numeric and symbolic time information. Artif Intell Rev 21:25–56
35. Ryabov V Trudel A (2004) Probabilistic temporal interval networks. In: TIME 2004, pp 64–67
36. Sabin D, Freuder EC (1994) Contradicting conventional wisdom in constraint satisfaction. In: Proceedings of the eleventh European conference on artificial intelligence. Wiley, Amsterdam, pp 125–129
37. Sabin D, Freuder EC (1996) Configuration as composite constraint satisfaction. In: Luger GF (ed) Proceedings of the (1st) artificial intelligence and manufacturing research planning workshop. AAAI Press, Menlo Park, pp 153–161
38. Sabin D, Freuder EC, Wallace RJ (2003) Greater efficiency for conditional constraint satisfaction. In: Proc., ninth international conference on principles and practice of, constraint programming—CP 2003, 2833, pp 649–663
39. Selman B, Kautz H (1993) Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: Proceedings of the 13th international joint conference on artificial intelligence, Chambéry, France. Morgan Kaufmann, San Mateo, pp 290–295
40. Schockaert S, De Cock M (2008) Temporal reasoning about fuzzy intervals. Artif. Intell. 172(8–9):1158–1193
41. Smith B, Dyer M (1996) Locating the phase transition in binary constraint satisfaction problems. Artif Intell 81:155–181
42. Song F, Cohen R (1991) Tense interpretation in the context of narrative. In: AAAI'91, pp 131–136
43. Thornton J, Beaumont M, Sattar A, Maher M (2004) A local search approach to modelling and solving interval algebra problems. J Logic Computat 14(1):93–112
44. Theodoulidis C, Loucopoulos P, Wangler B (1991) A conceptual modelling formalism for temporal database applications. Inf Syst 16(4):401–416
45. Tsamardinos I, Vidal T, Pollack ME (2003) CTP: a new constraint-based formalism for conditional temporal planning. Constraints 8(4):365–388
46. van Beek P (1992) Reasoning about qualitative temporal information. Artif Intell 58:297–326
47. Vidal T, Ghallab M (1996) Dealing with uncertain durations in temporal constraint networks dedicated to planning. In: ECAI-1996, pp 48–52
48. Vidal T, Fargier H (1999) Handling consistency in temporal constraint networks: from consistency to controllabilities. J Exp Theor 11:23–45
49. Vilain M, Kautz H (1986) Constraint propagation algorithms for temporal reasoning. In: Proceedings of the fifth national conference on arifical intelligence (AAAI'86), Philadelphia, PA. MIT Press, Cambridge, pp 377–382
50. Walsh T (2002) Stochastic constraint programming. In: The 15th European conference on artificial intelligence (ECAI-02)
51. Xu K, Li W (2000) Exact phase transitions in random constraint satisfaction problems. J Artif Intell Res 12:93–103



Malek Mouhoub obtained his MSc and PhD in Computer Science from the University of Nancy in France. He is currently Professor of Computer Science at the University of Regina in Canada. His research interests are in Artificial Intelligence and include Temporal Reasoning, Constraint Solving and Programming, Scheduling and Planning. Dr. Mouhoub's research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) federal grant in addition to several provincial and University funds and awards.

Amrudee Sukpan obtained her MSc and PhD degrees in Computer Science respectively from Prince of Songkla University in Thailand and the University of Regina in Canada. Her research interests are in the area of Constraint Programming and Temporal Reasoning. Amrudee's PhD thesis has been nominated by the department of Computer Science, at the University of Regina, for the Natural Sciences and Engineering Research Council of Canada (NSERC) doctoral award.