Scheduling with Structured Preferences

Roberto Micalizio and Gianluca Torta

Abstract The problem of finding a feasible schedule for a partially ordered set of tasks can be formulated as a Disjunctive Temporal Problem (DTP). While there exist extensions to DTPs that augment them by associating numeric costs to the violation of individual temporal constraints, they still make the restrictive assumption that the costs associated with constraints are independent of one another. In this paper we propose a further extension, which enables the designer to specify (directional) *dependencies* between the preferences associated with the constraints. Such preferences are represented by exploiting Utility Difference Networks (UDNs) that allow for the definition of structured objective functions based on the notion of *conditional difference independence* (CDI). Thanks to such conditional independencies, the specification of the utilities and the computation of the utility of (partial) solutions explored during the search for an optimal solution, turn out to be very similar to how probabilities are handled within a Bayesian Network. The paper presents a branch-and-bound algorithm for solving this new class of problems, analyzes its computational complexity and reports some encouraging experimental results.

1 Introduction

Since the seminal work by Dechter et al. [4], Temporal Constraint Satisfaction Problems (TCSPs) have drawn the attention of several AI researchers, and many problem formulations have been proposed along the time. Notably, the notion of Disjunctive Temporal Problems (DTPs) has been introduced in [17], in order to overcome the limits of Simple Temporal Problems (STPs) [4] by enabling the specification of temporal constraints consisting of disjuncts, each of which represents a temporal interval within which legal solutions can be found. This class of problems is expres-

R. Micalizio · G. Torta (✉)
Dipartimento di Informatica, Università di Torino,
C.so Svizzera 185, 10149 Torino, Italy
e-mail: gianluca.torta@unito.it

R. Micalizio
e-mail: roberto.micalizio@unito.it

M. Bramer and M. Petridis (eds.), *Research and Development in Intelligent Systems XXXII*, DOI 10.1007/978-3-319-25032-8_11

sive enough to model scheduling problems [12], as well as other problems of interest in AI (e.g., diagnosis [7, 14]). More recently, the research has been focused on how to address temporal *preferences* (i.e., *soft constraints*). Intuitively, a soft constraint allows one to express preferences on the distance between any two time points. For instance, in a calendar management scenario [10], relevant time points are reasonably the start and end times of the activities to be scheduled. Soft constraints can therefore be used to express the preference that some activities should last as long as possible, or that the distance between the ending of an activity and the starting of the subsequent one should be minimized.

While solving an STP or a DTP usually comes down to verifying the satisfiability of the (hard) constraints specified in the problem, solving a temporal problem with preferences requires to find an assignment of values to the time points that not only satisfies all the hard constraints, but also maximizes a given objective function defined over the soft constraints.

Two main problem formulations taking into account preferences have been proposed in the literature. In the first one, named Disjunctive Temporal Problem with Preferences (DTPP) [5], each constraint is augmented with a function that expresses how well an assignment satisfies the constraint itself. Solving a DTPP requires to find an assignment that maximizes the sum of the preference functions for each involved constraint. In the second formulation, named Valued Disjunctive Temporal Problem [9], each constraint is associated with a value representing the cost "paid" by a solution when that constraint is violated. Thus, in such a case a constraint can actually be violated, but its violation comes at a cost. A solution to a VDTP is therefore a solution whose cost is minimal.[1]

Both formulations, however, assume that the preferences (or costs) associated with the constraints are independent of one another. As a consequence, given a possible solution, its preference value can be computed by a linear function, that aggregates the preference value of each single constraint (i.e., how well the solution satisfies each constraint). Of course, such a function becomes the objective function to be maximized/minimized.

Such an assumption may prove to be too stringent in many applicative domains. Surprisingly, the problem of assessing the preference value of an assignment by taking into account dependencies among constraints has received little attention so far. To the best of our knowledge, only in [10] the authors propose the Multi-Criteria extension to DTPPs (MC-DTPP). Intuitively, the problem formulation includes, besides the disjunctive constraints as usual in DTPPs, also a set of *criteria*. Each criterion is a subset of constraints, which are bundled together as they refer to the same specific feature of the problem at hand. For each criterion (and pair of criteria), the user has to specify a weight denoting how "important" a user considers the satisfaction of that set of constraints.

[1] Note that soft constraints can equivalently be defined in terms of preferences or costs. In this paper we will deal with preferences.

In this paper we propose a different extension, that is suitable to capture a different kind of dependencies. We start by observing that in many practical problems not only there exist dependencies among the preferences, but also that such dependencies are conditional: The best choice for satisfying a constraint might be independent on the choices for the other constraints *given* the choices for a limited set of constraints.

We consider the VDTP formulation as our starting point. To represent causal, directional dependencies, we complement the basic VDTP with a Utility Difference Network (UDN) [1] that allows for the definition of structured objective functions based on the notion of *conditional difference independence* (CDI), after which we name our extended problem formulation *CDI-VDTP*. Thanks to such conditional independencies, the computation of the utility of (partial) solutions explored during the search for an optimal solution turns out to be very similar to how probabilities are computed from a Bayesian network.

The paper is organized as follows. After recalling background information in Sect. 2, we motivate our approach with an example in Sect. 3. In Sect. 4 we formally define CDI-VDTPs, and in Sect. 5 we propose a way to solve them. Section 6 presents experimental results, while Sect. 7 critically discusses related work, before concluding in Sect. 8.

2 Background

2.1 DTPs and VDTPs

A DTP is a pair $\langle \mathbf{X}, C \rangle$ where each element $X_i \in \mathbf{X}$ designates a time point, and each element $C_i \in C$ is a constraint of the form $c_{i,1} \vee \cdots \vee c_{i,n_i}$, and each disjunct $c_{i,j}$ is of the form $a_{i,j} \leq X_{i,j} - X'_{i,j} \leq b_{i,j}$, with $X_{i,j}, X'_{i,j} \in \mathbf{X}$ and $a_{i,j}, b_{i,j} \in \mathfrak{R}$.

A VDTP is a tuple $\langle \mathbf{X}, C, S, \varphi \rangle$ where $\mathbf{X}$, C are as in DTPs, while S and φ are defined as follows. The *valuation structure* S is a tuple $S = \langle E, \circledast, \succ \rangle$ where E is a totally ordered (w.r.t. $\succ$) set of *valuations* that can be combined with $\circledast$, a closed, associative, and commutative binary operator on E. Mapping $\varphi : C \rightarrow E$ assigns a cost $e \in E$ with (the violation of) each constraint $C \in C$. In the *weighted* VDTP, structure S is $\langle \mathfrak{R}^+ \cup \{\infty\}, +, > \rangle$ and the function to optimize is:

$$\sum_i \{\varphi(C_i) | violates(S, C_i)\}.$$

2.2 Utility Difference Networks

Given a set of finite-domain variables $\mathbf{A} = \{A_1, \ldots, A_n\}$ (*attributes*), a *multiattribute utility function* $u(A_1, \ldots, A_n)$ associates a numeric value with each assignment

$\mathbf{a} = a_1 \ldots a_n$ to the attributes. Utility Difference Networks (UDN) [1, 2][2] are a graphical representation of multiattribute utility functions that exhibit strong analogies and properties with the way Bayesian Networks (BN) represent joint probability distributions.

UDNs introduce the notion of a *reference value* $\mathbf{a_i^r}$ for each attribute A_i. The notion of *reference utility function* of a subset of attributes $\mathbf{H} \subseteq \mathbf{A}$ is defined as $u_r(\mathbf{H}) = u(\mathbf{H}\overline{\mathbf{h}}^{\mathbf{r}})$, where $\overline{\mathbf{h}}^{\mathbf{r}}$ is the reference assignment for variables $\overline{\mathbf{H}} = \mathbf{A} \backslash \mathbf{H}$. Based on u_r, the Conditional Independence relation CDI_r and the UDNs are defined then as follows.

Definition 1 [2] Let $\mathbf{H_1}, \mathbf{H_2}, \mathbf{H_3}$ be subsets of attributes. Set $\mathbf{H_1}$ is said to be Conditionally Independent of $\mathbf{H_2}$ given $\mathbf{H_3}$ (denoted $CDI_r(\mathbf{H_1}, \mathbf{H_2}|\mathbf{H_3})$) if for any assignment $\mathbf{h_3} \in dom(\mathbf{H_3})$, $u_r(\mathbf{H_1}|\mathbf{H_2}\mathbf{h_3}) = u_r(\mathbf{H_1}|\mathbf{h_3})$.

Let $\mathbf{A}$ be a set of attributes. A Utility Difference Network (UDN) is a DAG $\mathcal{G} = (\mathbf{A}, \mathbf{E})$ such that $\forall A \in \mathbf{A} : CDI_r(A, Co(A)|Pa(A))$, where $Pa(A)$ are the parents of A, $Dn(A)$ are the descendants of A, and $Co(A) = \mathbf{A} \backslash (\{A\} \cup Pa(A) \cup Dn(A))$.

UDNs decompose a multiattribute utility function into a sum as BNs decompose a joint probability distribution into a product, namely:

$$u(\mathbf{A}) = \sum_{i=1}^{n} u_r(A_i|Pa(A_i))$$

namely, in order to compute the utility of an assignment $\mathbf{a}$ to the attributes, it is sufficient to sum the values of the reference utility functions of each family of the UDN. A table specifying the values of $u_r(A_i|Pa(A_i))$ is named Conditional Utility Table (CUT).

3 Motivating Example

Let us consider a simplified planetary rover scenario as the one discussed in [3], and let us assume that a mission designer is finalizing the mission that a rover has to carry out. The mission plan has already been outlined, and Fig. 1 shows a portion of interest; edges between actions represent precedence links. The basic idea is that, once the rover has collected a soil sample by means of the `DRILL` action, it analyzes the sample and moves (`DRIVE`) to a position suitable for uploading (`COMM`) the collected data. The analysis and the movement could in principle be carried on simultaneously. The designer has to decide the mode with which the activities in the plan segment have to be completed. Such a decision has to be made balancing the quality and accuracy with which some activities are performed, against the time

[2] Utility Difference Networks are called Marginal Utility Networks (MUT) in [2]. In this paper we shall stick to the former name.

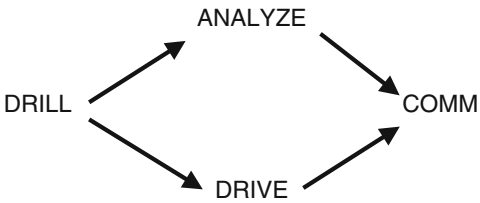

Fig. 1 A segment of a rover plan

DRIVE	
slow	[15, 25]
fast	[8, 13]

DRILL	
deep	[10, 13]
shallow	[5, 7]

COMM	
chn-1	[10, 15]
chn-2	[7, 10]

ANALYZE	
test-1	[7, 9]
test-2	[4, 5]
test-3	[3, 4]

Fig. 2 Modes with which activities can be completed and their expected interval durations

these activities take to be successfully completed. Figure 2 reports, for each action in the plan, the set of action modes and the associated duration intervals. Further inputs for the designer's decision making process are, however, global hard constraints and preferences. The designer has in fact to take into account that the action COMM, must be performed within a communication window, which opens over a precise period. The communication window is a hard constraint since it depends on the position of a satellite functioning as relay, and hence it is outside the control of the mission designer. Moreover, some activity modes are usually more preferred than others. For instance, it is usually preferred, and wiser, to perform a drive action in a slow mode; however, the fast mode can be used, if necessary, to avoid missing the communication window. In the tables of Fig. 2, the modes of each action, considered individually, are ordered from the most preferred down to the least preferred.

The challenge for the designer who has to select a mode for each action arises when we consider actions as being part of a mission. In such a case, the preferred mode for an action might depend on the mode already selected for a previous action. For instance, a scientist would prefer to always drill with modality *deep*, because such a mode usually enables the collection of more interesting samples. On the other hand, when such samples are collected, it is preferable to analyze them with modality *test-1* which is the most accurate one. Both modes, however are very time consuming, moreover the amount of data produced by means of *test-1* mode is usually huge; this impacts the communication, since in that case the 2-channel mode *chn-2* would be preferable, even though the general preference is to use *chn-1* mode.

The problem above could actually be encoded as a VDTP, but the only preferences one could specify would be those informally expressed by the order of the action modes within the tables in Fig. 2. Solving such a problem, thus, would lead to a solution that does not take into account the choices already made. In the following section we first introduce the CDI-VDTP formulation, and then we show how this rover example can be modeled as a CDI-VDTP.

4 Generalizing VDTPs to CDI-VDTPs

A CDI-VDTP is an extension to VDTPs in which the evaluation structure S and mapping φ are substituted by a Utility Difference Network $\mathcal{G}$, and a utility function u over $\mathcal{G}$.

More formally, a CDI-VDTP is a tuple $\langle X, C, \mathcal{G}, u\rangle$, where X and C are as in a standard VDTP; whereas, $\mathcal{G} = \langle \mathbf{A}, \mathbf{E}\rangle$ is a directed acyclic graph representing a Utility Difference Network such that:

- $\mathbf{A}$ is the set of network nodes (attributes). For each constraint $C_i \in C$, there is an attribute $A_i \in \mathbf{A}$ s.t. $dom(A_i)$ consists of the set $\{c_{i,1}, \ldots, c_{i,n_i}\}$ of disjuncts in C_i;
- $\mathbf{E}$ is a set of oriented edges $\langle A, A'\rangle$ such that $A, A' \in \mathbf{A}$. The edges in $\mathbf{E}$ describe the dependencies among the attributes over which one is interested in finding an assignment that maximizes the utility u. For instance, the edge $\langle A_i, A_j\rangle$, means that the selection of a value for A_i (disjunct for constraint C_i) (possibly) affects the utility of the value selection for A_j (i.e., disjunct for C_j) for maximizing the global utility.

Thanks to the properties of UDNs, the utility function u is compactly represented as a set of reference utility functions $u_r(A|Pa(A))$ for each $A \in \mathbf{A}$. In the following, we shall need to compute the maximum utility achievable given an instance $\mathbf{h}$ of a subset $\mathbf{H} \subseteq \mathbf{A}$ of variables. In analogy with the Most Probable Explanation (MPE) for Bayesian Networks, we define the Most Preferred Completion (MPC) of an instance $\mathbf{h}$ as:

$$MPC(\mathbf{h}) = \arg\max_{\overline{\mathbf{h}}}(u(\mathbf{h}, \overline{\mathbf{h}})).$$

Namely, $MPC(\mathbf{h})$ is the instance $\overline{\mathbf{h}}$ that completes $\mathbf{h}$ and yields a maximal utility.

Example 1 Let us consider again the planetary rover scenario, and see how it can be encoded in terms of a CDI-VDTP. The set of temporal variables X consists of a pair of variables for each action in the plan denoting the start and end time of the action itself. For instance, given action `DRILL`, two variables drl_s and drl_e are included in X. Also the communication window is encoded by means of two variables, cw_s and cw_e. In addition, a variable z is used to encode the time point used as a reference. As for the set C of constraints, we have a soft constraint for each action in the plan, for instance the `DRILL` action is associated with the following constraint:

$$C_{drl} = \{[10 \leq drl_e - drl_s \leq 13] \vee [5 \leq drl_e - drl_s \leq 7]\}$$

To model the preference value associated with such a constraint, however, we have to consider the dependencies of the constraint. In particular, we can assume that `DRILL` does not depend on any previous action, but it does influence `ANALYZE`, which in turn influences `COMM`. On the other hand, `DRIVE` can be considered as independent of the other actions. Relying on these observations, in Fig. 3 we sketch the UDN for this problem: Each node corresponds to a constraint in X (including the hard

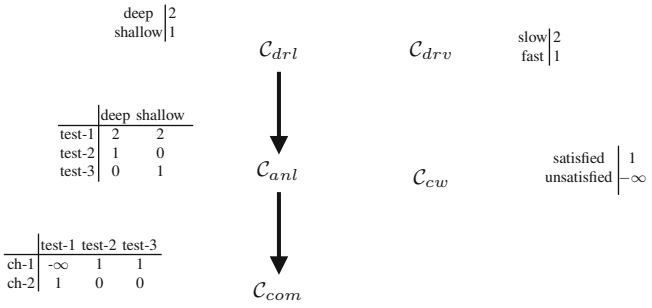

Fig. 3 The utility difference network for the rover example

constraint on the communication window); edges between nodes denote preference dependencies; in addition, in analogy to a Bayesian network, each node is associated with a CUT that defines the preferences for a constraint given its parent nodes.

In this particular case, the utility network has three roots. Two roots are C_{drv} and C_{cw}, representing the constraints associated with the drive action and the communication window, respectively. Being roots, a utility value is directly assigned to each of their disjuncts. For instance, the utility table associated with C_{drv} states that *slow* is generally preferred to *fast*. In addition, since the constraint about the communication window is hard, it is associated with two "fake modes", *satisfied* and *unsatisfied*, this last mode has utility $-\infty$, meaning that any solution that violates the communication window constraint is not acceptable. Note also that these two nodes have no relationships with the other nodes in the network. The third root is C_{drl}, which influences the constraint C_{anl} associated with the analysis action. In this case, the utility associated with each disjunct in C_{anl} depends on the disjuncts that have been selected for its parents (only C_{drl} in this example). The result, thus, is a CUT which looks like a Conditional Probability Table in Bayesian network. The particular table in the figure is to be interpreted as follows; independently of how deep the drill operation is, there is a strong preference in performing *test-1*; however, if the *test-1* is not possible, *test-2* should be preferred when the drill action was *deep*, whereas *test-3* should be preferred when the drill was *shallow*. Similarly, C_{anl} affects C_{com} (i.e., the constraint associated with the communication). Note, in this case, that when the analysis was carried out with mode *test-1*, the usage of mode *ch-1* is practically forbidden. On the other hand, the usage of *ch-1* should be preferred when the analysis was conducted either with *test-2* or *test-3* mode.

It is worth noting that at this stage of development, we assume that the utility values indicated in these tables result from information provided by the problem designer, who takes into account features of the rover that are not explicitly addressed by the temporal problem. For example, the preference on a slow drive could be motivated by safety reasons; whereas the preference of the usage of *ch-1* to *ch-2* could depend on the fact that the second mode is more resource consuming.

5 Solving CDI-VDTPs

Search Process. To solve a CDI-VDTP problem we adopt a strategy similar to the one proposed in [9]. The strategy recursively proceeds in a depth-first manner, and branches are pruned whenever their utility is guaranteed to fall below the cost of the best (i.e., maximal) solution found so far.

Our search strategy is outlined in the algorithm in Fig. 4. The algorithm takes as inputs:

- **h**: a (partial) assignment of modes to a subset of attributes **H**, i.e., a (partial) hypothesis;
- **mpc**: the Most Preferred Completion of **h**;
- $\overline{\mathbf{H}} = \mathbf{A} \backslash \mathbf{H}$ is the set of attributes whose mode has not been assigned yet;
- lwb: the utility of the best solution found so far;
- Δ: the set of all the best solutions found so far.

It is worth noticing that, while the first three arguments are passed by value, the last two arguments are passed by reference. Thereby, any change made during an invocation of **solve-CDI-VDTP** impacts all instances of the algorithm possibly

solve-CDI-VDTP$(\mathbf{h}, \mathbf{mpc}, \overline{\mathbf{H}}, lwb, \Delta)$
1. $util \leftarrow u(\mathbf{h}, \mathbf{mpc})$
2. **if** $util < lwb$ **then**
3. **return**
4. **end if**
5. **if** $\overline{\mathbf{H}} = \emptyset$ **then**
6. **if** $util > lwb$ **then**
7. $\Delta \leftarrow \emptyset$
8. $lwb \leftarrow util$
9. **end if**
10. $\Delta \leftarrow \Delta \cup \{\mathbf{h}\}$
11. **return**
12. **end if**
13. $A_i \leftarrow$ **select-attribute**$(\overline{\mathbf{H}})$;
14. $\overline{\mathbf{H}}' \leftarrow \overline{\mathbf{H}} - \{A_i\}$
15. $modes \leftarrow dom(A_i)$
16. **while** $modes \neq \emptyset$ **do**
17. $m \leftarrow$ **select-mode**$(modes)$; $modes \leftarrow modes \backslash \{m\}$
18. $\mathbf{h}' \leftarrow \mathbf{h} \cup \{A_i \leftarrow m\}$
19. **if consistent**$(\mathbf{h}')$ **then**
20. **solve-CDI-VDTP**$(\mathbf{h}', MPC(\mathbf{h}'), \overline{\mathbf{H}}', lwb, \Delta)$
21. **end if**
22. **end while**

Fig. 4 The solve-CDI-VDTP algorithm

active on the stack. In particular, when the search terminates Δ contains the set of best solutions and *lwb* their utility.

At each invocation, the algorithm determines the upper bound of the utility achievable by completing the current (partial) solution **h** (line 1), and checks whether it is lower than the best one so far (line 2); if yes, such a branch is not useful so it is pruned with the return statement. Otherwise, the algorithm checks whether there are still variables to be assigned (line 5): if $\overline{\mathbf{H}}$ is empty, then all attributes have been assigned and **h** is a complete solution. At this stage, the algorithm checks whether the new complete solution is better than any other solution found so far (lines 6–9); in the positive case, *lwb* is updated to be the utility of **h**, and Δ is emptied as all the solutions found so far were not optimal. In any case, **h** is added to Δ (line 10).

In case **h** is still a partial solution, the algorithm tries to get closer to a solution by selecting an attribute A_i from $\overline{\mathbf{H}}$ (line 14). Then the algorithm considers each mode m in $dom(A_i)$ (lines 16–22), in the order determined by function *select-mode* (line 17), and generates new hypotheses from them. In particular, for each $m \in dom(A_i)$, a new hypothesis $\mathbf{h}'$ is obtained by adding the assignment $A_i \leftarrow m$ to **h**. The temporal consistency of the new hypothesis $\mathbf{h}'$ is then verified by means of function *consistent* (line 19), that performs an STP consistency check. Finally, function *solve-CDI-VDTP* is recursively invoked over the new hypothesis $\mathbf{h}'$ and the new set of unassigned variables $\overline{\mathbf{H}}'$ (line 20).

The choice of the next attribute/mode to assign (calls to *select-attribute* and *select-mode*) can benefit from the heuristics established for DTP solving [19], such as conflict-directed backjumping, removal of subsumed variables, semantic branching, and no-good recording. However, in addition to such standard techniques, the choice of the next mode m to try for an attribute A_i can be determined by exploiting **mpc**. In particular, if $\mathbf{mpc} = MPC(\mathbf{h})$ assigns mode m_{mpc} to attribute A_i which is chosen next, that should be the first mode to try for A_i, since it maximizes the utility according to the UDN. Note that, in general, given a hypothesis **h** there may be several completions that maximize the utility, that may assign different modes to A_i. If the MPC computation is able to return all of them, the calls to *select-mode* should return them before the other modes of A_i.

MPC Computation. As pointed out in [2], one of the most desirable characteristics of UDNs is that most inference algorithms developed for BNs can be adapted with small changes to perform useful inferences on UDNs.

In particular, the computation of the MPC of a hypothesis **h** can be performed by adapting algorithms for computing the MPE of some evidence in a BN. We have chosen to use the jointree algorithm (see, e.g., [16]), which is particularly well-suited to the reuse of partial results for the incremental computation of the MPC of a new hypothesis $\mathbf{h}'$.

A jointree $\mathcal{T}$ derived from a UDN $\mathcal{G} = \langle \mathbf{A}, \mathbf{E} \rangle$ is an undirected tree whose nodes (clusters) are subsets $Cl_i \subseteq \mathbf{A}$ s.t. each family $Fam(A_j) = \{A_j\} \cup Pa(A_j)$ ($A_j \in \mathbf{A}$) is associated with a cluster Cl_i that contains $Fam(A_j)$. The computation of MPC follows the same steps of the classic jointree algorithm for BNs, except that the products of probabilities are replaced by sums of reference utilities, and sums of probabilities

are replaced by the computation of the max of reference utilities. For example, the *potential* of a cluster Cl_j is:

$$\phi_i = \sum_{Fam(A_j) \subseteq Cl_i} u_r(A_j | Pa(A_j))$$

instead of being the product of the CPTs contained in Cl_i.

After arbitrarily choosing a root, the jointree algorithm consists of an inward and an outward message passing phase, where messages flow respectively from the leaves to the root and vice-versa. In particular, during the inward phase, node Cl_i sends to its parent node Cl_j a message $\mathcal{M}_{i,j}$:

$$\mathcal{M}_{i,j} = \max_{Cl_i \setminus S_{i,j}} \left(\phi_i + \sum_{k \neq j} \mathcal{M}_{k,i} \right) \tag{1}$$

where $S_{i,j} = Cl_i \cap Cl_j$. Assume that message $\mathcal{M}_{i,j}$ has been cached during the computation of $MPC(\mathbf{h})$, and it turns out that it does not change during the computation of $MPC(\mathbf{h}')$, where $\mathbf{h}'$ is derived from $\mathbf{h}$ by adding an attribute assignment. Then, node Cl_i can avoid sending a message to node Cl_j. In turn, if node Cl_j does not receive messages from its children and has an unchanged potential, it can avoid the computation of the message for its parent.

It is easy to see that the replacement of $\sum$ with max in the UDN computations greatly increases the chance that messages can be reused. Indeed, the max operator can "absorb" changes in one or more items leaving $\mathcal{M}_{i,j}$ unchanged.

Computational Complexity. Due to space reasons, we just give some insights about the complexity of the proposed **solve-CDI-VDTP** algorithm; a more detailed analysis of a similar algorithmic approach applied to multi-agent diagnosis can be found in [8]. First of all, we note that the algorithm adopts a recursive strategy for exploring the search space, whose size is bounded by the size of the largest attribute domain, let say D_{max}, and by the number of attributes $|A|$, namely by the upper bound ${D_{max}}^{|A|}$ (note, however, that the exploration of the whole search space is very unlikely to occur, since this would require that function **consistent** never prunes the domains of the attributes). The two main sub-functions of **solve-CDI-VDTP**, namely **consistent** and *MPC*, can in principle hide further significant computational cost. It is possible to show that the former is polynomial in $|A|$ as the consistency check can be reduced to a number of invocations of checks over a Simple Temporal Network (STN) proportional to $|A|$ (see [8]). On the other hand, *MPC* is more complex, as we have seen, since its computation mirrors the computation of the MPE in Bayesian networks, which, as pointed out by Park and Darwiche [13], can be computed in space and time complexity exponential in the width of a given order of the BN nodes, and such a width is itself $O(|A|)$. In our implementation, we used a jointree algorithm, which in the worst case has complexity $O({D_{max}}^{|A|})$.

The computational complexity of the **solve-CDI-VDTP** algorithm is given by multiplying the size of the search space by the complexity of *MPC*, and is therefore exponential in the number of attributes $|A|$: $O({D_{max}}^{|A|} \cdot {D_{max}}^{|A|}) = O({D_{max}}^{2|A|})$.

6 Experimental Results

We have implemented the approach described in this paper as a Perl 5.16 program, exploiting the Boost::Graph module for representing STNs and checking their consistency with the Johnson algorithm, and the Graph module for representing the UDNs. Since the paper presents a new problem (namely, the CDI-VDTP), it is not possible to compare our prototype implementation with previous approaches. Therefore we shall focus on the feasibility of the approach and on the effectiveness of the caching technique discussed above.

The tests have been run on a virtual machine running Linux Ubuntu 12.04, equipped with an i7 M640 CPU at 2.80 GHz, and 4 GB RAM. We have considered three test sets *TS1*, *TS2* and *TS3* of increasing scale, each containing 25 cases. Table 1 reports the following characteristics:

- number *#vars* of variables and number *#constrs* of temporal constraints; *#constrs* is given as the sum of the number of *domain* constraints, shared by all test cases, and the number of constraints that change for each case;
- *#edges* of the UDN describing the dependencies among constraints.

Note that, for all test sets, the UDN networks are non-trivial, since they contain several dependencies among constraint preferences (represented by edges).

In order to appreciate the effectiveness of caching in the jointree algorithm (Sect. 5), we have run the test cases both with and without caching. Table 2 shows the average of the following statistics for the three test sets:

Table 1 Number of constraints, and size of UDNs

	TS1	TS2	TS3
#vars	84	164	244
#constrs	142 + 7	282 + 14	422 + 21
UDN #edges	29	59	89

Table 2 Avg time per sol (sec), and number of sols

Cache	TS1		TS2		TS3	
	yes	no	yes	no	yes	no
time/sol	2.6	6.1	8.8	23.3	15.4	40.9
#sols	3		3.8		3.7	

- *time/sol*: time to compute a solution;
- *#sols*: number of preferred solutions found.

Note that caching reduces the time needed for finding a solution by about 66 % for *TS1*, and about 62 % for *TS2* and *TS3*.

7 Related Work

Since the first formulation of the DTP with preferences (DTPPs) presented in [17], many alternative algorithms and techniques have been discussed in order to efficiently solve the problem. A first class of solutions are based on a semi-ring structure [5], which is used for combining local preference values into a global preference, and for ordering such global preferences so as to compare alternative solutions. Other approaches, such as MAXILITIS [11] are based on SAT algorithms, and ARIO [15] in particular is based on SAT algorithms designed for solving a given DTPP encoded as a Mixed Logical Linear Program (MLLP).

A different formulation of the disjunctive temporal problem with preferences is proposed in [9]. The novel formulation, dubbed Valued Disjunctive Temporal Problems (VDTPs), differs from DTTPs as it associates a single weight to each constraint as a whole, rather than a preference function at the object level as in a DTPP. Such a weight has to be interpreted as a cost a solution gathers when that specific constraint is violated; namely, when the solution does not satisfy any of the disjuncts mentioned in the constraint. In [9], VDTPs are solved by means of a branch-and-bound algorithm exploiting a meta-CSP representation of the temporal problem. In particular, each disjunctive constraint of the temporal problem is associated with a variable of the meta-CSP whose domain corresponds to the set of disjuncts in the constraint itself. The formulation of the CDI-VDTP presented in this paper takes a similar approach in formulating a meta-CSP. Also in CDI-VDTP, in fact, each constraint in the original temporal problem is mapped into a corresponding variable in the meta-CSP; the domain of such a meta-variable coincides with the set of disjuncts mentioned by the constraint itself. A significant difference, however, is that we do not associate a cost to the violation of a constraint as a whole, rather we associate a preference value to each of the disjunct of the constraint (i.e., to each value in the domains of meta-variables).

The approaches and formulations mentioned so far, however, all assume that the preference evaluation of a constraint is independent of the assignments made for the satisfaction of the other constraints. To the best of our knowledge, only the Multi-Criteria approach to DTPPs (MC-DTPPs) [10] takes up the challenge of finding optimal solutions in which the preference value of a constraint does depend on how other constraints are actually satisfied by a given solution. More precisely, in a MC-DTPP, one can define a criterion as a set of constraints; the rationale is that all the constraints related to some particular feature of the problem at hand should be collected within a single criterion. Each criterion is therefore associated with a weight, denoting the

importance that criterion has for the user. In addition, a triangular matrix of coefficients is used to represent the magnitude of correlations between any two criteria. The preference value of a solution is therefore computed as a weighted summation of the utilities associated with each criterion. The main difference between MC-DTPP and our CDI-VDTP formulation is that in a CDI-VDTP the dependencies among the constraint are not undirected as in a MC-DTPP. In fact, MC-DTPP criteria define subsets of constraints that are somehow related with each other, but there is no way to express a causal directionality of such relationships. In many practical cases, however, such a directionality exists. (Consider for example business process workflows [6], supply chains and production systems [18], and so on.) The CDI-VDTP formulation takes advantage of the causal directionality, and enables the user to express conditional independences among constraints by relying on the graph-based representation of the UDNs.

8 Conclusions

In this paper we raised the issue of how dealing with preferences that are not completely independent of one another in a disjunctive temporal problem. As far as we know, such a problem has received little attention, and only in [10] a Multi-Criteria DTPP has been proposed.

In this paper we extended the VDTP formulation [9] of temporal problems with the notion of Conditional Difference Independence. The resulting framework, named CDI-VDTP, enables a user to take advantage of the causal dependencies between the preferences associated with the constraints, and to define an objective function shaped over a Utility Difference Network (UDN), in which each node corresponds to a constraint and (oriented) edges between nodes represent causal dependencies. Solving a CDI-VDTP, thus, consists in computing solutions whose utility is optimal; this can be achieved by exploiting algorithms which are similar to those used for computing probabilities in a Bayesian network, but applied to the UDN. In the paper we also presented a branch-and-bound algorithm for solving CDI-VDTPs by exploring the space of possible solutions. Results collected by a preliminary implementation have been discussed, and show that the proposed solution is actually feasible.

As a future work, we intend to further extend the CDI-VDTP formulation with the addition of a set of variables that, although included within the UDN, are not associated with temporal constraints. The rationale would be to explicitly model via these variables aspects of the domain under consideration that might affect the preference values of a subset of constraints. For instance, in the planetary rover scenario, the level of battery power could be represented explicitly within the UDN by means of a specific variable; such a variable could then affect the duration of actions such as drive or communicate depending on the assumed level of power. Problems like planning and diagnosis could therefore exploit such a richer CDI-VDTP to create expectations or verify hypotheses.

References

1. Brafman, R.I., Engel, Y.: Directional decomposition of multiattribute utility functions. Algorithmic Decision Theory, pp. 192–202. Springer, Berlin (2009)
2. Brafman, R.I., Engel, Y.: Decomposed utility functions and graphical models for reasoning about preferences. In: AAAI (2010)
3. Bresina, J.L., Jónsson, A.K., Morris, P.H., Rajan, K.: Activity planning for the mars exploration rovers. In: 15th International Conference on Automated Planning and Scheduling (ICAPS 2005), pp. 40–49 (2005)
4. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**, 61–95 (1991)
5. Khatib, L., Morris, P.H., Morris, R.A., Rossi, F.: Temporal constraint reasoning with preferences. In: IJCAI, pp. 322–327 (2001)
6. Mans, R.S., Russell, N.C., van der Aalst, W.M., Moleman, A.J., Bakker, P.J.: Schedule-aware workflow management systems. Transactions on Petri Nets and Other Models of Concurrency IV, pp. 121–143. Springer, Berlin (2010)
7. Micalizio, R., Torta, G.: Diagnosing delays in multi-agent plans execution. In: Proceedings European Conference on Artificial Intelligence (ECAI'12), pp. 594–599 (2012)
8. Micalizio, R., Torta, G.: Explaining interdependent action delays in multiagent plans execution. J. Auton. Agents Multi-Agent Syst. (2015)
9. Moffitt, M.D.: On the modelling and optimization of preferences in constraint-based temporal reasoning. Artif. Intell. **175**(7–8), 1390–1409 (2011)
10. Moffitt, M.D., Peintner, B., Yorke-Smith, N.: Multi-criteria optimization of temporal preferences. In: Proceedings of CP06 Workshop on Preferences and Soft Constraints (Soft06), pp. 79–93. Citeseer (2006)
11. Moffitt, M.D., Pollack, M.E.: Applying local search to disjunctive temporal problems. In: IJCAI, pp. 242–247 (2005)
12. Oddi, A., Cesta, A.: Incremental forward checking for the disjunctive temporal problem. In: Proceedings European Conference on Artificial Intelligence (ECAI'00), pp. 108–112 (2000)
13. Park, J., Darwiche, A.: Complexity results and approximation strategies for map explanations. J. Artif. Intell. Res. **21**, 101–133 (2004)
14. Roos, N., Witteveen, C.: Diagnosis of simple temporal networks. In: Proceedings European Conference on Artificial Intelligence (ECAI'08), pp. 593–597 (2008)
15. Sheini, H.M., Peintner, B., Sakallah, K.A., Pollack, M.E.: On solving soft temporal constraints using sat techniques. In: Principles and Practice of Constraint Programming (CP 2005), pp. 607–621. Springer (2005)
16. Shenoy, P.P., Shafer, G.: Propagating belief functions with local computations. IEEE Expert **1**(3), 43–52 (1986)
17. Stergiou, K., Koubarakis, M.: Backtracking algorithms for disjunctions of temporal constraints. Artif. Intell. **120**(1), 81–117 (2000)
18. Tan, K.C.: A framework of supply chain management literature. Eur. J. Purch. Supply Manag. **7**(1), 39–48 (2001)
19. Tsamardinos, I., Pollack, M.E.: Efficient solution techniques for disjunctive temporal reasoning problems. Artif. Intell. **151**(1), 43–89 (2003)