Temporal Constraint Satisfaction Problems and Difference Decision Diagrams: A Compilation Map

Hélène Fargier, Frédéric Maris, Vincent Roger

Temporal Constraint Satisfaction Problems and Difference Decision Diagrams: a Compilation Map

Hélène Fargier, Frédéric Maris, Vincent Roger

IRIT (Institut de Recherche en Informatique de Toulouse)
University of Toulouse
France

***Abstract*—The frameworks dedicated to the representation of quantitative temporal constraint satisfaction problems, as rich as they are in terms of expressiveness, define difficult requests - typically NP-complete decision problems. It is therefore adventurous to use them for an online resolution. Hence the idea to compile the original problem into a form that could be easily solved. Difference Decision Diagrams (DDDs) have been proposed by [1] as a possible way to cope with this difficulty, following a compilation-based approach. In this article, we draw a compilation map that evaluates the relative capabilities of these languages (TCSP, STP, DTP and DDD) in terms of algorithmic efficiency, succinctness and expressiveness.**

***Keywords*-Temporal Constraint Satisfaction Problems; Difference Decision Diagrams; Knowledge Compilation; Computational Complexity.**

I. Introduction

In constraint-based reasoning, a model is built by means of a set of constraints restricting the combinations of values of the variables. Soon enough, [2] proposed a formalism to represent temporal problems which derives from the definitions developed in the classical CSP framework, namely the TCSP (Temporal Constraint Satisfaction Problem) formalism. In a TCSP, each variable represents an instant at which an event (start or end of an action, expiration of a milestone) occurs, and the domains are continuous (equal to $\mathbb{R}^+$). Binary constraints represent the possible time intervals between two events; for instance, this allows to express that a task must take place before or after another (by a constraint of the form $start_2 - end_1 \in\ ]-\infty, -duration_1 - duration_2] \cup [0, +\infty[$), or that the beginning of two tasks must be synchronized ($start_1 - start_2 \in [0, 0]$).

A TCSP whose constraints are not disjunctive (i.e. problems with only one interval per constraint) is called a STP (Simple Temporal Problem). Unlike the resolution of TCSP instance, which defines a NP-hard problem, that of STP instances can be performed in polytime.

The STP framework has been extended by [3] yielding the framework of generalized disjunctive temporal problems (DTP): while in a TCSP each constraint is a disjunction of elementary constraints on the same pair of variables (it is a disjunction of literal of the form $x - y \in I$ bearing on the same x and y), this restriction is dropped in the DTP formalism: the "literals" in a single constraint may relate different variables. The difficulty is that TCSP and DTP problems can not be solved online with the guarantee that the response will be given in polynomial time: checking the consistency of this type of network defines a NP-complete problem [2], [3]. Hence, the idea of a preprocessing, a "compilation" of the original problem by its translation into a form that allows an efficient treatment of the queries. It is an emerging idea in different areas of IA, like constraint-based reasoning [4], [5], product configuration [6], planning under uncertainty [7] or automated reasoning [8]. Technically, this compilation is performed offline, before the online query phase: this relaxes the constraints on its temporal complexity[1].

This idea has already been proposed for temporal problems with the introduction of Difference Decision Diagrams (DDDs) [1]. The purpose of the present paper is to draw a complexity study of DDDs, TCSPs, STPs and DTPs that enables the comparison of the different languages in terms of expressiveness, succinctness and ability to address in polynomial time some queries and transformations, e.g. the consistency checking query or the conditioning transformation (that corresponds to the assignment of one (interval) of value(s) to some difference $x - y$); in other term, to draw a compilation map in the sense of [9].

After introducing in the next Section the definitions and notations that we need, we present in Section 3 the formalisms studied in this work: TCSP, STP, DTP and DDD. Then, Section 4 and 5 present our results, namely a first compilation map for these languages: the former is devoted to the relative succinctness and expressivity of these languages, while the latter studies their ability to address the requests of interest in polytime.

II. Preliminaries

Let $\chi = \{x_1, ..., x_n\}$ be a finite set of variables. Let D_{x_i} be the domain of variable x_i, with $i = 1..n$. For any subset $X \subseteq \chi$, $\overrightarrow{x}$ is an assignment of the variables in X. $val(x_i, \overrightarrow{x})$ is the value assigned to $x_i \in X$ by $\overrightarrow{x}$. D_X is the cartesian product of the domains of the variables of X. The concatenation of

[1]But obviously the compiled form may have a spatial complexity exponential in the worst case.

two assignments $\overrightarrow{x}$ and $\overrightarrow{y}$ of two disjoint sets of variables X and Y is an assignment of $X \bigcup Y$; it is denoted $\overrightarrow{x}.\overrightarrow{y}$. For any function f on χ and any assignment $\overrightarrow{x}$ of a subset of χ, we denote by $f_{|\overrightarrow{x}}$ the restriction of f to $\overrightarrow{x}$, i.e. $f_{|\overrightarrow{x}}(\overrightarrow{y}) = f(\overrightarrow{x}.\overrightarrow{y})$.

Following [10], [11], a representation language L on a set of variables χ is a tuple $\langle C_L, Var_L, f^L, s_L \rangle$ where:

- C_L is a set of data structures (also called "formulas" or "L-representations");
- $Var_L : C_L \rightarrow 2^\chi$ is a scope function associating each L-representation the subset of X it depends on
- f^L is an interpretation function associating each L-representation α a mapping f^L_α from the set of all assignments over $Var_L(\alpha)$ to ν (typically $\nu = \{\top, \bot\}$, i.e. $f^L_\alpha(\overrightarrow{x})$ is a boolean value);
- s_L is a size function from C_L to $\mathbb{N}$ that provides the size of any L-representation.

In other words, a data structure α represents a function f^L_α; data structures in the same language obey the same syntax, and are interpreted using the same interpretation function f^L. For example, let χ be a set of boolean variables, C_L defines all the CNF on χ; each CNF α is interpreted according to the principles of propositional logic and for any assignment $\overrightarrow{x}$ of χ, $f^L_\alpha(\overrightarrow{x})$ is the truth value of α according to $\overrightarrow{x}$.

A L-representation α and a L'-representation β are said to be equivalent if for every $\overrightarrow{x}$, $f^L_\alpha(\overrightarrow{x}) = f^{L'}_\beta(\overrightarrow{x})$, i.e. if they represent the same function. A language L is "canonical" if the same function can not be represented by two different L-representations. For example, $\text{OBDD}_>$ is a canonical language, while CNF is not. The property of canonicity is important in a graph-based language like the one of decision diagrams, for practical reasons: by a caching process, it allows to merge equivalent sub-formulas (i.e. isomorphic sub-graphs) and therefore to compact the data structure - to save space.

When the interpretation functions take their values in $\{\bot, \top\}$ - typically, when considering constraint satisfaction problems, clauses bases, etc - $\overrightarrow{x}$ is a solution (or model) of α iff $f^L_\alpha(\overrightarrow{x}) = \top$; we denote by $sol(\alpha)$ the set of its solutions.

III. Representing temporal problems

In the sequel, we focus on representation languages dedicated to quantitative temporal problems. The set χ of variables correspond to instants, events, milestones, etc, and therefore domains are equal to $\mathbb{R}^+$.

A. Constraint-based languages

1) The TCSP language: As a data structure, a temporal constraint satisfaction problem is a directed acyclic graph $\alpha = (\mathcal{N}_\alpha, \mathcal{A}_\alpha)$ whose nodes are bijectively labelled by variables of χ ($var(N)$ denotes the variable labeling node N) and arcs by intervals in $\mathbb{R}$. $Interv((N, N'))$ denotes the set of (disjoint) intervals labelling (N, N'): (N, N') represents the temporal constraint $var(N') - var(N) \in \bigcup_{I \in Interv((N,N'))} I$.

C_{TCSP} is the set of graphs that can be built over χ; $var_{TCSP}(\alpha)$ is the set of labels of nodes in the graph; $s_{TCSP}(\alpha) = Card(\mathcal{N}_\alpha) + \Sigma_{(N,N') \in \mathcal{A}_\alpha} Card(Interv((N, N')))$ measures the size of the data structure by the number of intervals and variables carried by the graph; finally, an assignment $\overrightarrow{x}$ is a solution of TCSP α iff it satisfies all its constraints. Formally, if for arc (N, N') in $\mathcal{A}_\alpha$, $val(var(N), \overrightarrow{x}) - val(var(N'), \overrightarrow{x}) \in \bigcup_{I \in Interv((N,N'))} I$ then $f^{TCSP}_\alpha(\overrightarrow{x}) = \top$, and $f^{TCSP}_\alpha(\overrightarrow{x}) = \bot$ otherwise.

In the original model, the intervals carried by a TCSP are closed intervals; in this paper open intervals are allowed (which does not make the problems more difficult: determining the consistency of such a TCSP remains an NP-complete problem).

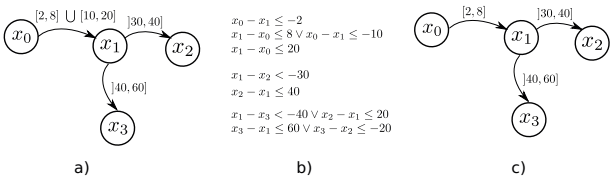

Fig. 1. (a) a TCSP, (b) a DTP and (c) a STP. The STP (c) represents a sequence of events compatible with (a) and (b).

2) The STP language: Simple temporal constraints satisfaction problems (STP) are TCSP in which each edge carries one interval only. The functions var_{STP}, s_{STP} and the interpretation function f^{STP} are the same as those defined for the TCSP language; the STP language is therefore a sublanguage of the TCSP one, obtained by a restriction on the "syntax".

The STP language is incomplete with respect to the TCSP language: the set of solutions of TCSP with two nodes labeled x and y and carrying constraint "$x - y \in I \cup J$", I and J being two disjoint intervals, can not be represented by a single STP on these two variables.

3) The DTP language: The framework of Disjunctive Temporal Problems is a generalization of the STP one proposed by [3]. As a data structure, a DTP is a set of clauses $\alpha = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$. Each clause is a set of literals $\mathcal{C}_i = \{l_1, \ldots l_{m_j}\}$ and a literal l_j is a tuple $(x, y, \lesssim, c)$ which represents the constraint $x - y \lesssim c$ where:

- $x \in \chi$, $y \in \chi$, $x \neq y$ ($var(l_j)$ denotes the ordered pair (x, y) of variables associated to literal l_j);
- c is a constant (which will be denoted $const(l_j)$);
- $\lesssim$ (which will be denoted $op(l_j)$) belongs to $\{<, \leq\}$;

C_{DTP} is the set of sets of clauses that can be built in this way over χ; $var_{DTP}(\alpha)$ is the set of variables associated with the literals of α; $s_{DTP}(\alpha) = \Sigma_{\mathcal{C}_i \in \alpha} Card(\mathcal{C}_i)$ measures the size of the data structure; finally, an assignment of $\overrightarrow{x}$ is a solution of a DTP iff it satisfies all its clauses, which results in the following interpretation function: $f^{DTP}_\alpha(\overrightarrow{x}) = \top$ if $\overrightarrow{x}$ satisfies at least one literal (on difference constraint) in each clause of α and $\bot$ otherwise. The set $sol(\alpha)$ of assignments $\overrightarrow{x}$ such that $f^{DTP}_\alpha(\overrightarrow{x}) = \top$ is the set of solutions of DTP in the usual sense of the term.

B. Difference Decision Diagrams

Difference Decision Diagrams [1] constitute an attempt of extending of Ordered Decision Diagram to temporal problems.

The idea is basically to label the nodes by difference constraints, rather than by propositional variables.

1) The DDD language: A DDD over χ is a directed acyclic graph $\alpha = (\mathcal{N}_\alpha, \mathcal{A}_\alpha)$ with a single root, denoted $root(\alpha)$ and two terminal nodes, labeled by $\bot$ and $\top$ respectively- see Figure 2 for an example.

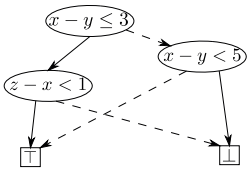

Fig. 2. DDD ordered according to the order $z \succ y \succ x$; it represents the solutions of the system of inequalities $(x-y \geqslant 5) \vee (x-y \leq 3 \wedge z-x < 1)$

Each non-leaf node N has exactly two children, $low(N)$ and $high(N)$, and is labeled by a constraint of the form $x_i - x_j \lesssim c$, where:

- $x_i \in \chi$ (denoted $pos(N)$, the positive variable) and $x_j \in \chi$ (denoted $neg(N)$, the negative variable) are such that $i \neq j$
- c is a constant (denoted $const(N)$)
- $\lesssim$ (which will be denoted $op(N)$) belongs to $\{<, \leq\}$

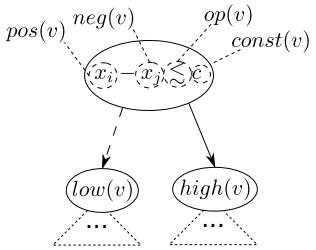

Fig. 3. Illustration of basic notations for a non-leaf node v

[1] propose the following notations:

- $high(v)$ (respectively $low(v)$) is the first child (respectively the second child) of N; this function selects the constraints to be satisfied by an assignment $\overrightarrow{x}$ when the label of N evaluates to true (respectively false).
- $var(v) = (pos(v), neg(v))$
- $bound(v) = (const(v), op(v))$
- $cstr(v) = (var(v), bound(v))$
- $attr(v) = (cstr(v), high(v), low(v))$

The size of a DDD is its number of arcs and nodes: $s_{DDD}(\alpha) = Card(\mathcal{N}_\alpha) + Card(\mathcal{A}_\alpha)$; $var_{DDD}(\alpha)$ is obviously the set of variables involved in at least one node of α. The interpretation function of α is defined as follows:

- For any node N and any assignment $\overrightarrow{x}$ (covering at least the two variables that label the node), $f_N^{node}(\overrightarrow{x})$ returns the value $\top$ if $\overrightarrow{x}$ satisfies the constraint labelling N and $\bot$ otherwise. Formally, iff: $(val(pos(N), \overrightarrow{x}) - val(neg(N), \overrightarrow{x}))\ op(N)\ const(N)$.
- if α is a leaf, then $f_\alpha^{DDD}(\overrightarrow{x})$ is equal to the value carried by this node. So we either have $f_\alpha^{DDD}(\overrightarrow{x}) = \top$ or $f_\alpha^{DDD}(\overrightarrow{x}) = \bot$.
- otherwise the root N of α is a non-leaf node and $f_\alpha^{DDD}(\overrightarrow{x}) = f_{high(N)}^{DDD}(\overrightarrow{x})$ if $f_N^{DDD}(\overrightarrow{x} = \top$ and $f_\alpha^{DDD}(\overrightarrow{x}) = f_{low(N)}^{DDD}(\overrightarrow{x})$ otherwise.

A DDD is in locally reduced form if it does not have isomorphic nodes, nonselective nodes nor stammering nodes - see Figure 4. Formally, if any pair of non-leaf nodes u and v satisfies the following properties:

1) $(attr(u) = attr(v)) \Rightarrow u = v$
2) $low(u) \neq high(u)$
3) $(var(v) = var(low(v)))$ and $bound(v) \leq bound(low(v)) \Rightarrow (high(v) \neq high(low(v)))$

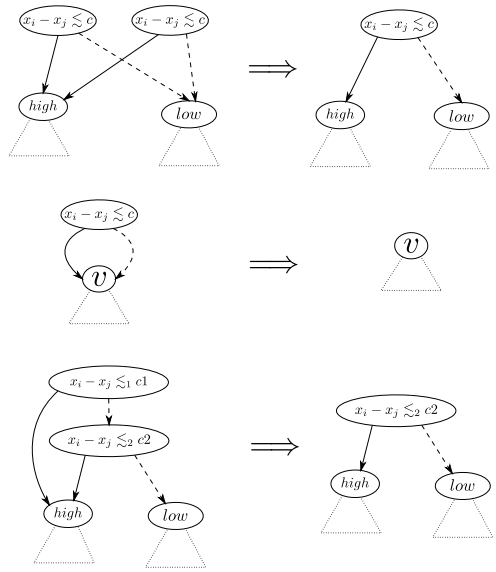

Fig. 4. An example of applications of the three rules for obtaining a locally reduced DDD . Top to down: isomorphic nodes are merged; then stammering nodes are merged; finally a non-selective node is removed.

It is always possible to transform a DDD representation into an equivalent locally reduced representation in polynomial time (the fusion of isomorphic nodes can in particular be performed implicitly using a "unique table"). This property remaining satisfied for any kind of DDD, it is assumed in the following that the DDD considered are locally reduced.

2) Ordered Difference Decision Diagrams: In the DDD framework like in the BDD framework, it can be required that the diagrams respect an order that determines how the variables are met along the paths of the graph; Let $\succ$ be a total order on χ; $\succ$ extends to pairs of variables (and thus to nodes) by means of a lexicographic extension: $(x, y) \succ (x', y')$ if $y \succ y'$ or if $y = y'$ and $x \succ x'$; to rank-order two nodes bearing the same variables, it is required that the one who carrying the more restrictive constraint is met first on the paths (see Figure 2). Formally, α obeys $\succ$ if and only if for every node N we have:

1) $pos(N) \succ neg(N)$
2) $var(high(N)) \succ var(N)$
3) $var(low(N))) \succ (var(N) \vee (var(N) = var(low(N)) \wedge bound(low(N)) \succ bound(N))$

Let us denote $\texttt{DDD}_{\succ}$ the language defined by the DDD representations on χ that are ordered by $\succ$.

3) Path-reduced and Tight Difference Decision Diagrams: [1] then proposes several restrictions on DDD thus several sub languages eg, P−DDD, the language of path-feasible DDDs and PT−DDD, that of path-feasible and minimal DDDs.

A path p in a DDD represents a set $[p]$ of constraints: for any pair of consecutive nodes N and N' of p, $[p]$ contains a constraint $cons(N, N')$ of the form $x - y \leq c$ or $x - y < c$ if $N' = high(N)$, or a constraint of the form $x - y > c$ or $x - y \geq c$ if $N' = low(N)$. A path is feasible (consistent) if and only if there is an assignment $\overrightarrow{x}$ over χ which satisfies all the constraints of $[p]$. A DDD α is said to be path-feasible if each of its paths is feasible. [1] shows that any DDD representation can be transformed into an equivalent and path-feasible one, but the procedure proposed in this seminal paper is exponential in the worst case (assuming that $P \neq NP$), as shown in the following Section. We denote P−DDD the language of path feasible DDDs, and $\texttt{P−DDD}_{\succ}$ its ordered variant.

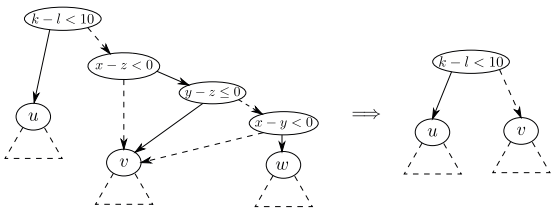

Fig. 5. A non path-feasible DDD (left) and a path-feasible version (right) of this latter

The P−DDD language allows only one possible representation for tautologies (the node $\top$) and only one representation for unsatisfiability (the node $\bot$) - [1] calls this property "semi canonicity". However, the usual property of canonicity does not hold, neither for path-feasible DDD, nor for path-feasible ordered DDDs ($\texttt{P−DDD}_{\succ}$): consider the order $z > y > x$, the set of solutions of the system of equalities $\{x - y = 0,\ y - z = 0,\ x - z = 0\}$ can be represented by four different (but equivalent) path-feasible DDD which carry respectively the constraints sets $\{x - y = 0,\ y - z = 0,\ x - z = 0\}$, $\{x - y = 0,\ y - z = 0\}$, $\{x - y = 0,\ x - z = 0\}$, $\{y - z = 0,\ x - z = 0\}$.

[1] then propose to make the constraints of the diagram as restrictive as possible, in a sense close to the notion of minimality proposed in the TCSP framework. To this extend they introduce the concept of *dominant* constraint: $x_i - x_j \lesssim c$ is a dominant constraint in a path p (and more broadly in a set of constraints, such as $[p]$) if and only if any other constraints $(x_i - x_j \lesssim' c')$ bearing on the same variables is less restrictive (i.e. $(c, \lesssim) < (c', \lesssim')$). The idea is to require that such dominant constraints are as restrictive as possible, without changing the set of solutions: a constraint $\beta = x_i - x_j \lesssim c$ is tight in a path $p = p_1 \wedge \beta \wedge p_2$ if there is no constraint $\beta' = x_i - x_j \lesssim' c'$ such that $(c', \lesssim') < (c, \lesssim)$ and that $[p_1] \cup \{\beta'\} \cup [p_2] \equiv [p]$. Hence, a path is tight iff all its dominant constraints are tight and a P−DDD is tight if and only if all the paths that compose it are tight. We call PT−DDD the language of tight and path-feasible DDDs, and $\texttt{PT−DDD}_{\succ}$ its ordered variant.

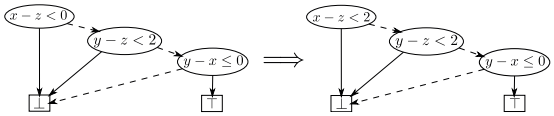

Fig. 6. A non tight DDD (left) and a tight version (right) of this latter

IV. A Compilation Map

We have described in the previous Section a number of languages devoted to the representation temporal problems with numeric variables. The objective of the preliminary compilation map presented in this paper is to define their respective merits in terms of efficiency with respect to a number of tasks, be they queries (e.g. "does the problem have a solution ?") or transformations (e.g. "make the conjunction of two representations representing the constraints of the planner for the former, the objectives of the customer for the latter"). The question is also to evaluate the respective representational capacity of the languages in terms of expressiveness and of succinctness.

A. Expressiveness, polynomial translation and succinctness

The languages TCSP, STP, DTP and DDD (and sub languages) defined on a given χ do not have the same power of representation; e.g. a TCSP representing the solutions of the constraint $(x - y \leq 0) \vee (x - y \geq 1)$ can not be transformed into an equivalent STP. In a compilation map, this capability is captured by the notion of expressiveness.

Formally, considering two languages L and L' defined on the same variables (on χ) targeting the same output set (in our context, $\{\bot, \top\}$), we say that L can be compiled into L' (that L' is at least as expressive as L), and denote $L' \preceq_e L$, iff for any L representation α there is a L'-representation β such that $\alpha \equiv \beta$; we write $L' \prec_e L$ when the compilation is possible from L to L' only and $L \sim_e L'$ when possible in both directions. L' is said to be as least as succinct as L iff for any L representation α there is an equivalent L'-representation β the size of which is polynomial with respect to the one of α; this is denoted $L' \preceq_s L$. If this compilation can be performed in polytime, i.e. if there exists a polynomial algorithm which, for any L-representation α computes a L'-representation β such that $\alpha \equiv \beta$, we write $L' \preceq_p L$. $L \prec\!\succ_e L'$ (resp. $L \prec\!\succ_s L'$) means that L and L' are incomparable in terms of expressivity (resp. succinctness).

Any STP representation can trivially be transformed into an equivalent DTP representation the clauses of which are singletons; this STP is also a TCSP with unary constraints only; moreover, it is clear that any STP representation can be transformed in linear time into an equivalent DDD representation (or ordered DDD, or P−DDD or PT−DDD representation); however, we have seen that STP is less expressive than the other three languages - i.e. $\texttt{TCSP} \prec_e \texttt{STP}$, $\texttt{PT−DDD}_{\succ} \prec_e \texttt{STP}$

and DTP $\prec_e$ STP etc; and TCSP $\prec_p$ STP, PT−DDD$_\succ$ $\prec_p$ STP and DTP $\prec_p$ STP. Similarly, the TCSP language is incomplete with respect to the DTP language: the set of solutions of the DTP with three variables x, y and z and one constraint $(x - y \leq 0) \vee (z - x \leq 0)$ can not be represented by a TCSP on these three variables. It is possible to encode a DTP as a TCSP, but this requires the addition of a (linear) number of variables [12].

Our results in terms of succinctness are based on the following Propositions:

Proposition 1: Let $\succ$ be a total order over χ. Any TCSP representation on χ can be translated, in polynomial time and space, into an equivalent DTP representation on χ.

Indeed, consider an arc of a TCSP between a variable y and a variable x carrying the set of intervals: $\{I_1, ..., I_m\} = \{[a_1, b_1], ..., [a_m, b_m]\}$; it represents the binary constraint $T_{xy} = (a_1 \leq x - y \leq b_1) \vee ... \vee (a_m \leq x - y \leq b_m)$. We can assume without loss of generality that intervals are pairwise disjoints: $\forall i \in \{1..m-1\}, b_i < a_{i+1}$. In order to encode T_{xy} in the DTP formalism, we first add two constraints $y - x \leq -a_1$ and $x - y \leq b_m$ in order to prevent the assignment of $x - y$ to be lower than I_1 or greater I_m. Then for each $i \in \{1..m-1\}$, we add the disjunctive constraint $(x - y \leq b_i) \vee (y - x \leq -a_{i+1})$; this prevents the assignments of $x - y$ to be between the end of I_i and the beginning of I_{i+1}. Hence, in the DTP, $x - y$ is required to belong exactly to the set of intervals defining constraint T_{xy} in the TCSP (see Figure 7 for an example).

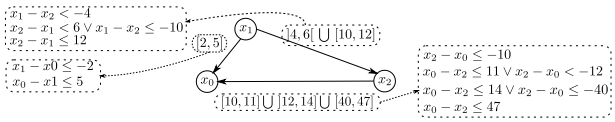

Fig. 7. Compilation of TCSP as a DTP representation

Proposition 2: Let $\succ$ be a total order over χ. It holds that:

(i) Any TCSP representation on χ can be translated, in polynomial space and time, into an equivalent DDD$_\succ$ representation.

(ii) Any DTP representation on χ can be translated, in polynomial space and time, into an equivalent DDD representation on χ.

(iii) Any DDD representation on χ can be translated, in polynomial space and time, into an equivalent DTP representation on χ.

The principle of the proofs are the following:

(i) Consider a TCSP and any of its constraints $T_{xy} = (a_1 \leq x - y \leq b_1) \vee ... \vee (a_m \leq x - y \leq b_m)$ (for instance the one of Figure 8). Each elementary pair of inequations $a_i \leq x - y \leq b_i$ can be directly compiled into a small DDD$_\succ$. Let γ be the one corresponding to the first disjunct; we iteratively add the next one, say, β as follows: all the arcs of γ pointing to the leaf $\bot$ point now to $root(\beta)$; the $\top$ sinks of γ and β are merged.We thus get a DDD$_\succ$ for each constraint. The conjunction of these diagrams is performed as follows: let γ be the one corresponding to the first of them; we iteratively add the next one, say, β by redirecting all the arcs of γ pointing to the leaf $\top$ to $root(\beta)$; the $\bot$ sinks of γ and β are merged. We thus get in polynomial time a DDD$_\succ$ equivalent to the original TCSP. If the constraints are considered in accordance to the order $\succ$ on the variables / constraints, the resulting structure is an instance of DDD$_\succ$. Then, the diagram will be transformed into a path-feasible one, or a tight one, but these operations are not necessarily polynomial neither in time nor in space.

(ii) Similarly, a DTP is a conjunction of disjunctions of literals of the form $x - y \simeq a$; each literal can be directly transcribed into a small DDD. Binary operations $\wedge$ and $\vee$ between these DDD are applied as for the TCSP compilation in the previous proof. This DDD is not necessarily ordered.

(iii) To prove this point, simply enumerate the paths of the DDD from the $\top$ node to the root; we obtain a disjunction of conjunctions, which may be (by distributivity) transformed into a conjunction of disjunctions.

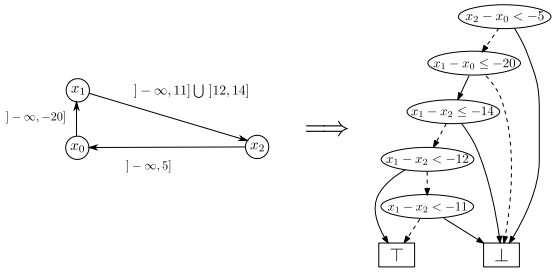

Fig. 8. Compilation of an inconsistent TCSP as a DDD$_\succ$ representation

Proposition 3: Let $\succ$ be a total order over χ. There are PT−DDD$_\succ$ representations which have no equivalent DTP representation of polynomial size and there are DTP representations that have no equivalent P−DDD representation of polynomial size, nor any equivalent DDD$_\succ$ representation of polynomial size.

The incomparability in terms of succinctness of DTP and DDD$_\succ$ come from the fact that one can always convert a CNF into a DTP and an OBDD$_\succ$ into a PT−DDD$_\succ$: just consider for any propositional variable p, the pair of variables x_p, y_p; the positive literal p leads to the constraint $x_p - y_p \geq 1$ and the negative literal $\neg p$ to the constraint $x_p - y_p \leq 0$. The incomparability of DTP and PT−DDD$_\succ$ then derives from the one of CNF and OBDD$_\succ$. The proof of the incomparability of DTP and P−DDD is similar: we can always transform a CNF into a DTP and a FBDD into a PT−DDD (using the same encoding of literal by difference constraints) while CNF and FBDD are incomparable in terms of succinctness.

Proposition 4: Let $\succ$ be a total order over χ.

(i) Any P−DDD representation on χ can be translated into an equivalent DDD$_\succ$, but there are P−DDD representations

that have no equivalent $\texttt{DDD}_{\succ}$ representation of polynomial size.

(ii) Any DDD representation on χ can be translated into an equivalent P−DDD representation on χ, but under the assumption $P \neq NP$, this compilation can not be done in polynomial time.

The proof first item relies on the fact that any FBDD can be transformed into a P−DDD: to each variable p of the FBDD correspond two variables x_p and y_p; nodes labeled by p in FBDD are labeled by $x_p - y_p \geq 0$: we obtain a P−DDD; if it were possible to transform this P−DDD into a $\texttt{DDD}_{\succ}$ of polysize, we could recover an OBDD; but there are FBDD representations which have no equivalent OBDD representations of polynomial size.

For proving the second item, recall that [1] propose algorithms to transform a DDD (resp. P−DDD) into an equivalent P−DDD (resp a PT−DDD). The non polynomiality of these algorithms (assuming that $P \neq NP$) is trivial since any TCSP can be transformed into an equivalent DDD in polynomial time, while (i) the test of consistency is NP-complete for the TCSP language and (ii) it is an easy problem for P−DDD and sublanguages.

These propositions yield the following results about the relative expressiveness and succinctness of the temporal languages (summarized in Tables I, II and III):

Theorem 1 (Expressiveness):

- DDD $\sim_e$ DTP $\prec_e$ TCSP $\prec_e$ STP
- DDD $\sim_e$ $\texttt{DDD}_{\succ}$ $\sim_e$ $\texttt{P-DDD}_{\succ}$ $\sim_e$ $\texttt{PT-DDD}_{\succ}$ $\sim_e$ $\texttt{DDD}_{\succ}$ $\sim_e$ P−DDD $\sim_e$ PT−DDD

We can now compare the succinctness of equally expressive languages:

Theorem 2 (Succinctness):

- For all $L \in \{\texttt{DDD}_{\succ}, \texttt{P-DDD}_{\succ}, \texttt{PT-DDD}_{\succ}, \texttt{P-DDD}, \texttt{PT-DDD}\}$, $L \prec\succ_s$ DTP
- DDD $\prec_s$ P−DDD $\prec_s$ $\texttt{P-DDD}_{\succ}$,
- DDD $\prec_s$ PT−DDD $\prec_s$ $\texttt{PT-DDD}_{\succ}$

↗	STP	TCSP	DTP
STP	✓	✓	✓
TCSP	!	✓	✓
DTP	!	!	✓
DDD	!	!	•
$\texttt{DDD}_{\succ}$	!	!	•
P−DDD	!	!	•
$\texttt{P-DDD}_{\succ}$	!	!	•
PT−DDD	!	!	•
$\texttt{PT-DDD}_{\succ}$	!	!	•

TABLE I
EXPRESSIVITY AND SUCCINCTNESS OF STP, TCSP AND DTP. ✓MEANS "COMPILATION IN POLYNOMIAL TIME", • MEANS "NO POLYNOMIAL COMPILATION ALGORITHM" (NOT ALL REPRESENTATIONS IN THE SOURCE LANGUAGE ARE REPRESENTABLE IN POLYSIZE IN THE TARGET LANGUAGE); ! MEANS "THE COMPILATION IS NOT ALWAYS FEASIBLE" (INCOMPLETE TARGET LANGUAGE)

.

↗	DDD	P−DDD	PT−DDD
STP	✓	✓	✓
TCSP	✓	○	○
DTP	✓	○	○
DDD	✓	○	○
$\texttt{DDD}_{\succ}$	✓	○	○
P−DDD	✓	✓	?
$\texttt{P-DDD}_{\succ}$	✓	✓	?
PT−DDD	✓	✓	✓
$\texttt{PT-DDD}_{\succ}$	✓	✓	✓

TABLE II
EXPRESSIVITY AND SUCCINCTNESS OF NON ORDERED DIFFERENCE DECISION DIAGRAMS. ✓MEANS "COMPILATION IN POLYNOMIAL TIME", ○ MEANS "NO POLYNOMIAL COMPILATION ALGORITHM UNLESS P = NP"; ? MEANS "DON'T KNOW"

.

↗	$\texttt{DDD}_{\succ}$	$\texttt{P-DDD}_{\succ}$	$\texttt{PT-DDD}_{\succ}$
STP	✓	✓	✓
TCSP	✓	○	○
DTP	•	•	•
DDD	•	•	•
$\texttt{DDD}_{\succ}$	✓	○	○
P−DDD	•	•	•
$\texttt{P-DDD}_{\succ}$	✓	✓	?
PT−DDD	•	•	•
$\texttt{PT-DDD}_{\succ}$	✓	✓	✓

TABLE III
EXPRESSIVITY AND SUCCINCTNESS OF ORDERED DIFFERENCE DECISION DIAGRAMS. ✓MEANS "COMPILATION IN POLYNOMIAL TIME", ○ MEANS "NO POLYNOMIAL COMPILATION ALGORITHM UNLESS P = NP", • MEANS "NO POLYNOMIAL COMPILATION ALGORITHM";? MEANS "DON'T KNOW"

.

B. Compilation map: queries and transformations

A direct consequence of the previous results is that consistency cannot be checked in polytime for DDD nor for $\texttt{DDD}_{\succ}$, assuming $P \neq NP$. Indeed, if it were the case, it could be possible to test the consistency of a TCSP representation α by compiling it as a DDD α' (which is polytime) then to test the consistency of α' in polynomial time.

The DDD and $\texttt{DDD}_{\succ}$ languages are actually comparable to BDD (which do not guarantee consistency checking in polynomial time) and not to FBDD nor to OBDD. The class that could be close to the FBDD one is the class path-feasible DDD (where all paths are feasible) and the one that could correspond to ordered boolean decision diagram is $\texttt{P-DDD}_{\succ}$.

Beyond the query of consistency, we describe in this Section a number of requests, and we analyze the ability of the studied languages to address them efficiently.

1) Definitions: Let us first recall some basic queries of transformation that are worthwhile studying for temporal application (typically, for planning problems):

- **CO**:, L satisfies consistency if and only if there exists a polytime algorithm that maps any L-representation α to 1 if it has a solution (if $\exists \overrightarrow{x}, f^L_\alpha(\overrightarrow{x}) = \top$) and to 0 otherwise.
- **EQ**: L satisfies equivalence if and only if there exists a polytime algorithm that maps every pair of L-representations $\langle \alpha, \beta \rangle$ to 1 if $f^L_\alpha = f^L_\beta$ and to 0 otherwise.
- **MX**: L satisfies model extraction if and only if there exists a polytime algorithm that maps L-representation α to an assignment $\overrightarrow{x}$ such that $f^L_\alpha(\overrightarrow{x}) = \top$ if any and stops without returning anything otherwise.
- **CE**: L satisfies clausal entailment if there exists a polytime algorithm that maps any L-representation α and any temporal clause β to 1 if $sol(\alpha) \subseteq sol(\beta)$, and to 0 otherwise.

And for transformations:

- **CD**: L satisfies conditioning if and only if there is a polytime algorithm that maps any L-representation α, any pair of variables $(x, y) \subseteq \chi$ and any $w \in \mathbb{R}$ to a L-representation of $\bigvee_{u,v \text{ s.t } u-v=w} f^L_{\alpha|x \leftarrow u.y \leftarrow v}$.
- $\wedge$**BC**: L satisfies bounded conjunction if and only if there exists a polytime algorithm that maps any pair of L-representations α and β to a L-representation of $f^L_\alpha \wedge f^L_\beta$
- $\wedge$**C**: L satisfies conjunction if and only if there exists a polytime algorithm that maps any set $\{\alpha_1, ..., \alpha_n\}$ of L-representations to a L-representation of $\wedge^n_{i=1} f^L_{\alpha_i}$
- $\vee$**BC**: L satisfies bounded disjunction if and only if there exists a polytime algorithm that maps any pair of L-representations α and β to a L-representation of $f^L_\alpha \vee f^L_\beta$
- $\vee$**C**: L satisfies disjunction if and only if there exists a polytime algorithm that maps any set $\{\alpha_1, ..., \alpha_n\}$ of L-representations to a L-representation of $\vee^n_{i=1} f^L_{\alpha_i}$

Many planners use DTP or TCSP solvers to schedule events in time. The task of the planner is then to build the plan by selecting actions, and the handling of time is left to the temporal solver. Consistency checking (CO) is used to know if selected actions can lead to an admissible plan, and model extraction (MX) to order actions and/or fix their start date. A set of admissible plans (all composed by the same actions, but possibly different in their scheduling) is memorized; At execution, when a temporal contingent event occurs that breaks the plan chosen (for example, because the user learns that an action has begun), a replanning phase is entered, that conditions the set of plans according to the new information, i.e. performs a CD transformation and then extracts a new (temporally consistent) plan - again, a MX request.

It is worthwhile noticing that the definition of conditioning we propose does not correspond to the usual one, which assigns a given value to one single variable, because the classical definition of "conditioning" would be of no use in the current context: it would lead to formula that are beyond the language. E.g. the conditioning of the STP $x_1 - x_2 \in [3, 8]$ by $x_1 \leftarrow 4$ can not be represented by a STP over $\chi = \{x_1, x_2\}$ (or by any of the languages described in this article). In general, in temporal reasoning, there is a distinguished variable, say x_0, which represents the beginning of time - in our example, we would have $\chi = \{x_0, x_1, x_2\}$; assigning a value to an instant x_i yields assigning a value to the difference $x_i - x_0$ (in our example, we would choose $x_1 - x_0 \leftarrow 4$).

Other requests (typically, $\wedge$- and $\vee$- based transformations, and the EQ query on which the caching mechanism rely) are crucial when building a compiled form in a bottom up approach.

2) Results: As to queries, we have obtained the following results:

Theorem 3: The results of Table IV hold.

	CO	EQ	MX	CE
STP	✓	✓	✓	✓
TCSP	○	○	○	○
DTP	○	○	○	○
DDD	○	○	○	○
$\text{DDD}_\succ$	○	○	○	○
P−DDD	✓	○	✓	?
$\text{P−DDD}_\succ$	✓	○	✓	?
PT−DDD	✓	○	✓	?
$\text{PT−DDD}_\succ$	✓	(? ○)	?	?

TABLE IV
MAP OF QUERIES. ✓MEANS "SATISFIES IN POLYNOMIAL TIME", ○: "DO NOT SATISFY IN POLYNOMIAL TIME, UNLESS P = NP", ●: "DO NOT SATISFY", ?: "DON'T KNOW"; (?) DENOTES A CONJECTURE

Let us briefly sketch the proofs of these results:

- CO: easy on STP and hard on TCSP [2]; hard on DTP [3]; hard on $\text{DDD}_\succ$ and thus on DDD because any TCSP can be transformed in polytime into an equivalent $\text{DDD}_\succ$; easy on P−DDD and subclasses by definition.
- EQ: easy on STP (compute the distance matrices and compare them: they must be equal); hard on TCSP and DTP, DDD and $\text{DDD}_\succ$: if EQ were satisfied, it would provide a way to satisfy CO (by testing the equivalence between the formula and the unsatisfiable one, represented by the sink labeled by $\bot$) - and we have seen that the languages do not satisfy CO; hard on P−DDD since every FBDD can be encoded as a P−DDD and EQ is hard on FBDD. We conjecture hardness on $\text{PT−DDD}_\succ$ because it is not a canonical language ($x - y = 3, y - z = 4, x - z = 7$ may be represented by several $\text{PT−DDD}_\succ$).
- MX: easy on STP (by computing the distance matrix); hard on the TCSP and DTP, DDD and $\text{DDD}_\succ$ languages since they do not satisfy CO; easy on P−DDD and sublanguages (P−DDD, $\text{P−DDD}_\succ$, PT−DDD, $\text{PT−DDD}_\succ$): it is enough to choose the path from the $\top$ sink to the root - because the paths are feasible, we get a *consistent* STP the solutions of which are solutions on the DDD.
- CE: easy on STP, by computing the corresponding distance matrix and applying a Floyd-Warshall pass on it before testing the literals of the clause iteratively until getting a one that is a consequence of the STP; hard on TCSP and DTP, DDD and $\text{DDD}_\succ$ languages since they do not satisfy CO. We conjecture that it is also hard on

PT−DDD$_\succ$ and on its superclasses.

As to transformations, we obtained:

Theorem 4: The results of Table V hold.

	CD	∧BC	∧C	∨BC	∨C
STP	✓	✓	✓	!	!
TCSP	✓	✓	✓	!	!
DTP	✓	✓	✓	✓	•
DDD	✓	✓	✓	?	?
DDD$_\succ$	✓	?	•	?	•
P−DDD	•	•	•	○	•
P−DDD$_\succ$	•	•	•	?	•
PT−DDD	•	•	•	○	•
PT−DDD$_\succ$	•	•	•	?	•

TABLE V
MAP OF TRANSFORMATIONS. ✓: "SATISFIES IN POLYNOMIAL TIME", ○: "DOES NOT SATISFY IN POLYNOMIAL TIME, UNLESS P = NP", •: "DOES NOT SATISFY", !: "THE TRANSFORMATION IS NOT ALWAYS FEASIBLE"; ?: "DON'T KNOW"

The surprising result is that conditioning is hard on the DDD languages; to show that, consider a P−DDD on $\chi = \{x, z_1, \ldots, z_n, y\}$ representing the constraint set $\{x - z_1 \in [1,1] \cup [0,0], z_1 - z_2 \in [1,1] \cup [0,0], \ldots, z_n - z_{n-1} \in [1,1] \cup [0,0], y - z_n [1,1] \cup [0,0]\}$ and whose nodes are labeled $x - z_1$, $z_1 - z_2$, $z_{n-1} - z_n$, $y - z_n$. This diagram is path-feasible and tight. It implies that $x - y \in [0, n]$. If it is now conditioned by $x - y = [\frac{n}{2}, \frac{n}{2}]$, the size of the representation of the corresponding function by a P−DDD explodes (it is not enough to add $x - y \in [\frac{n}{2}, \frac{n}{2}]$ at the end). The operation of conditioning is exponential for P−DDD and all its sub languages.

The other proofs are easy:

- The satisfaction of CD, ∧BC and ∧C is immediate on STP, TCSP, DTP.
- DTP satisfies ∨BC because disjunction is distributive on conjunction. Finally, we get a • regarding ∨C on DTP because any CNF of propositional logic can be encoded by a DTP whose "literals" are of the form $x_p - y_p \in \ ]-\infty, -1] \cup [1, +\infty[$ (each p corresponding to a propositional variable of the original CNF) and ∨C is not satisfied on CNF language.
- the hardness of ∧BC (and therefore of ∧C) on P−DDD and sublanguages results from hardness of conditioning on these languages.
- the hardness of ∨BC (and therefore of ∨C) on P−DDD and PT−DDD results from the correspondence between these languages and FBDD, and on the hardness of ∨BC on FBDD.
- the hardness of ∧C on P−DDD and sub-languages results from the correspondence between these languages and OBDD / FBDD for which unbounded conjunction is exponential in space in the worst case.

The situation is thus not so good for the DDD languages: as soon as CO is required, almost all transformations are hard; even worst: conditioning, which is the basic transformation for planification as for many other applications, may lead to an exponential explosion.

V. CONCLUSION

In this article we presented preliminary results for the building of a compilation map of temporal problems. We looked at the efficiency of several representation languages, namely TCSP, DTP and different types of decision diagrams - ordered DDDs, path-feasible DDDs and path-feasible and tight DDDs.

It appears that, from a theoretical point of view at least, basic ordered difference decision diagram are disappointing: they do not satisfy any of the considered requests. And this despite the fact than an order on the variables is required: in the DDD framework, this assumption does not provide any good property, contrarily to what happens in the classical decision diagram one. The language of path-feasible DDD (and its sub languages) is more interesting from the perspective of queries. Unsurprisingly, compiling a temporal constraints satisfaction problem (a TCSP or a DTP instance) as a path-feasible decision diagram is hard, and the compiled form can be exponentially more space-consuming than the original one. The bad news is that even the most basic transformation, namely conditioning, is not satisfied by any kind of path-feasible DDD.

These conclusions are rather pessimistic regarding the formalism proposed in [1] when compared to the classical, non compiled, ones. This does not mean that efficient target languages do not exist for temporal problems; sets of "convex" Horn-type constraints seem for instance worthwhile studying. The compilation of temporal problems simply remains an open question.

REFERENCES

[1] J. B. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard, "Difference decision diagrams," in *Proceedings of the 8th Annual Conference of the EACSL*, 1999, pp. 111–125.

[2] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

[3] K. Stergiou and M. Koubarakis, "Backtracking algorithms for disjunctions of temporal constraints," *Artificial Intelligence*, vol. 120, no. 1, pp. 81–117, 2000.

[4] N. R. Vempaty, "Solving constraint satisfaction problems using finite state automata," in *AAAI'1992*, 1992, pp. 453–458.

[5] R. Weigel and B. Faltings, "Compiling constraint satisfaction problems," *Artificial Intelligence*, vol. 115, no. 2, pp. 257–287, 1999.

[6] J. Amilhastre, H. Fargier, and P. Marquis, "Consistency restoration and explanations in dynamic csps application to configuration," *Artificial Intelligence*, vol. 135, no. 1-2, pp. 199–234, 2002.

[7] J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier, "SPUDD: stochastic planning using decision diagrams," in *UAI '99*, 1999, pp. 279–288.

[8] M. Cadoli and F. M. Donini, "A survey on knowledge compilation," *AI Communications*, vol. 10, no. 3-4, pp. 137–150, 1997.

[9] A. Darwiche and P. Marquis, "A knowledge compilation map," *JAIR*, vol. 17, pp. 229–264, 2002.

[10] G. Gogic, H. Kautz, and C. P. F, "The comparative linguistics of knowledge representation," in *IJCAI'95*, 1995, pp. 862–869.

[11] H. Fargier, P. Marquis, A. Niveau, and N. Schmidt, "A knowledge compilation map for ordered real-valued decision diagrams," in *AAAI'14*, 2014, pp. 1049–1055.

[12] L. R. Planken, "Temporal reasoning problems and algorithms for solving them (literature survey)," Master's thesis, Delft University of Technology, October 2007.