On the Tractability of Restricted Disjunctive Temporal Problems

T. K. Satish Kumar
Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

Abstract

In this paper, we provide a polynomial-time *deterministic* algorithm, and an even simpler *randomized* algorithm, for solving a restricted (but very expressive) class of disjunctive temporal problems (DTPs). The general form of a DTP is as follows. We are given a set of events $\mathcal{X} = \{X_0, X_1 \ldots X_N\}$ (X_0 is the "beginning of the world" node and is set to 0 by convention), and a set of constraints $\mathcal{C}$. A constraint $c_i \in \mathcal{C}$ is a disjunction of the form $s_{(i,1)} \vee s_{(i,2)} \ldots s_{(i,T_i)}$. Here, $s_{(i,j)}$ $(1 \leq j \leq T_i)$ is a simple temporal constraint of the form $L_{(i,j)} \leq X_{b_{(i,j)}} - X_{a_{(i,j)}} \leq U_{(i,j)}$ for $0 \leq a_{(i,j)}, b_{(i,j)} \leq N$. We will first provide a pseudo-polynomial-time *randomized* algorithm for solving the following restricted class of DTPs (which we will refer to as RDTPs (restricted DTPs)): Any $c_i \in \mathcal{C}$ is of one of the following types: (Type 1) $(L \leq X_b - X_a \leq U)$, (Type 2) $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_a \leq U_2) \ldots (L_{T_i} \leq X_a \leq U_{T_i})$, (Type 3) $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_b \leq U_2)$. We will then provide a strongly polynomial-time *deterministic* algorithm for solving the same problem, and extend the ideas further to provide an even simpler *randomized* algorithm—the expected running time of which is much less than that of the *deterministic* algorithm. Our polynomial-time algorithms for solving RDTPs bear important implications on not only being able to handle limited (but very useful) forms of disjunctions in metric temporal reasoning (that would otherwise require an exponential search space), but also in pruning large parts of the search spaces associated with general DTPs.

Introduction

Expressive and efficient temporal reasoning is central to many areas of Artificial Intelligence (AI). Several tasks in planning and scheduling, for example, involve reasoning about temporal constraints between actions and propositions in partial plans (see (Nguyen and Kambhampati 2001) and (Smith *et al.* 2000)). These tasks may include threat resolution between actions in partial order planning, analyzing resource consumption envelopes to guide the search for a good plan (see (Kumar 2003)), etc. Among the important formalisms used for reasoning with metric time are simple temporal problems (STPs) and disjunctive temporal problems (DTPs) (see (Oddi and Cesta 2000) and (Stergiou and Koubarakis 1998)). Unlike DTPs, STPs can be solved in polynomial time, but are not as expressive as DTPs.

An STP is characterized by a graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$, where $\mathcal{X} = \{X_0, X_1 \ldots X_N\}$ is a set of events (X_0 is the "beginning of the world" node and is set to 0 by convention), and $e = \langle X_i, X_j \rangle \in \mathcal{E}$, annotated with the bounds $[LB(e), UB(e)]$, is a simple temporal constraint between X_i and X_j indicating that X_j must be scheduled between $LB(e)$ and $UB(e)$ seconds after X_i is scheduled ($LB(e) \leq UB(e)$). Figure 1(A) shows an example of an STP which (like all other instances of the class) can be solved in polynomial time using shortest paths (see (Dechter *et al.* 1991)).

DTPs are significantly more expressive than STPs, and allow for disjunctive constraints. The general form of a DTP is as follows. We are given a set of events $\mathcal{X} = \{X_0, X_1 \ldots X_N\}$ (X_0 is the "beginning of the world" node and is set to 0 by convention), and a set of constraints $\mathcal{C}$. A constraint $c_i \in \mathcal{C}$ is a disjunction of the form $s_{(i,1)} \vee s_{(i,2)} \ldots s_{(i,T_i)}$. Here, $s_{(i,j)}$ $(1 \leq j \leq T_i)$ is a simple temporal constraint of the form $L_{(i,j)} \leq X_{b_{(i,j)}} - X_{a_{(i,j)}} \leq U_{(i,j)}$ for $0 \leq a_{(i,j)}, b_{(i,j)} \leq N$. Figure 1(B) shows an example of a DTP which expresses disjunctive constraints.

Although DTPs are expressive enough to capture many tasks in planning and scheduling (like threat resolution and plan merging), they require an exponential search space. The principal approach taken to solve DTPs has been to convert the original problem to one of selecting a disjunct from each constraint, and then checking that the set of selected disjuncts forms a consistent STP. Checking the consistency of, and finding a solution to an STP can be performed in polynomial time using shortest path computations (see (Dechter *et al.* 1991)). The computational complexity of solving a DTP comes from the fact that there are an exponentially large number of disjunct combinations possible. The "disjunct selection problem" can also be cast as a constraint satisfaction problem (CSP) (see (Oddi and Cesta 2000) and (Stergiou and Koubarakis 1998)), or a satisfiability problem (SAT) (see (Armando *et al.* 1999)) and solved using standard search techniques applicable for them. Epilitis is a systems that efficiently solves DTPs using CSP search techniques like *conflict-directed backjumping* and *nogood recording* (see (Tsamardinos and Pollack 2003)).

In this paper, we will first provide a pseudo-polynomial-time *randomized* algorithm for solving the following re-

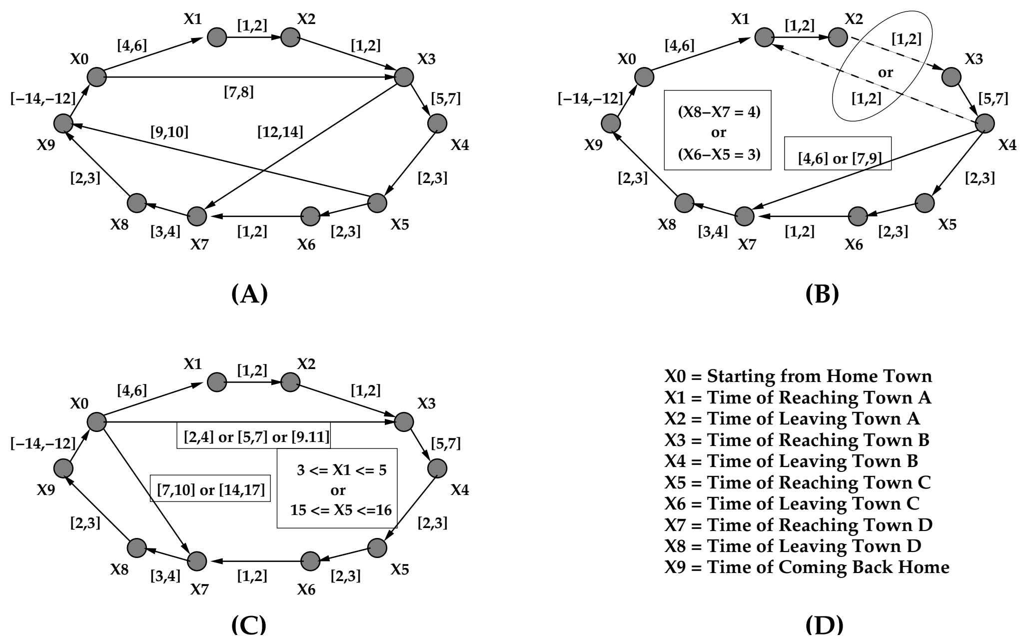

Figure 1: Shows an example to illustrate the kinds of reasoning possible in (A) STPs, (B) DTPs, and (C) RDTPs. The example is about an agent who should plan her visit to 5 towns (starting and ending at her home town) respecting various temporal constraints. In (A), only simple temporal constraints are specified. An edge from X_3 to X_7 annotated with $[12, 14]$, for example, means that she should reach town D within 12 and 14 days of reaching town B. In (B), 3 disjunctive constraints are specified (enclosed by boxes). One of these, for example, says that either $X_3 - X_2 \in [1, 2]$ or $X_1 - X_4 \in [1, 2]$. Such a constraint may arise when the agent has no preference visiting town A before or after town B, but only knows that she can drive between the 2 towns within 1 and 2 days. Similarly, the constraint $(X_8 - X_7 = 4) \vee (X_6 - X_5 = 3)$ may arise out of her preference to stay in at least one of the 2 towns C and D for as long as possible. In (C), all the disjunctions are of Type 2 or Type 3 (enclosed by boxes). The constraint $X_7 - X_0 \in [7, 10] \cup [14, 17]$, for example, is a Type 2 disjunction, and may arise out of the agent's requirement to attend a social gathering in town D, which takes place only on certain days of a week. Similarly, the constraint $(X_1 \in [3, 5]) \vee (X_5 \in [15, 16])$ is a Type 3 disjunction, and may arise out of the agent's need to meet at least one of two friends who are respectively available in towns A and C on specific days. (D) gives an annotation of the time points $X_0, X_1 \dots X_9$ used in (A), (B) and (C).

stricted class of DTPs (which we will refer to as RDTPs (restricted DTPs)): Any $c_i \in \mathcal{C}$ is of one of the following types: (Type 1) $(L \leq X_b - X_a \leq U)$, (Type 2) $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_a \leq U_2) \dots (L_{T_i} \leq X_a \leq U_{T_i})$, (Type 3) $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_b \leq U_2)$. We will then provide a strongly polynomial-time *deterministic* algorithm for solving the same problem, and extend the ideas further to provide an even simpler *randomized* algorithm—the expected running time of which is much less than that of the *deterministic* algorithm. Our polynomial-time algorithms for solving RDTPs bear important implications on not only being able to handle limited (but very useful) forms of disjunctions in metric temporal reasoning (that would otherwise require an exponential search space), but also in pruning large parts of the search spaces associated with general DTPs. Figure 1(C) shows an example of an RDTP.

Random Walks and Expected Arrival Times

In this section, we will provide a quick overview of random walks, and the theoretical properties attached with them. Figure 2(A) shows an undirected graph with weights on edges. A *random walk* on such a graph involves starting at a particular node, and at any stage, randomly moving to one of the neighboring positions of the current position. The probability with which we move to a specific neighbor of the current node is proportional to the *weight* on the edge that leads to that neighbor. One of the properties associated with such random walks on undirected graphs is that if we denote the expected time of arrival at some node (say L) starting at a particular node (say R) by $T(R, L)$, then $T(R, L) + T(L, R)$ is $O(m\mathcal{H}(L, R))$. Here, m is the number of edges, and $\mathcal{H}(L, R)$ is the "resistance" between L and R, when the weights on edges are interpreted as electrical resistance values (see (Doyle and Snell 1984)).

Figure 2(B) shows a particular case of the one in Figure 2(A), in which the nodes in the graph are connected in a linear fashion, and the edges are unweighted—i.e. the probabilities of moving to the left or to the right from a particular node are equal (except at the end-points). In this scenario, it is easy to note that by symmetry, $T(L, R) = T(R, L)$. Further, using the property of random walks stated above, if there are n nodes in the graph, then both $T(L, R)$ and

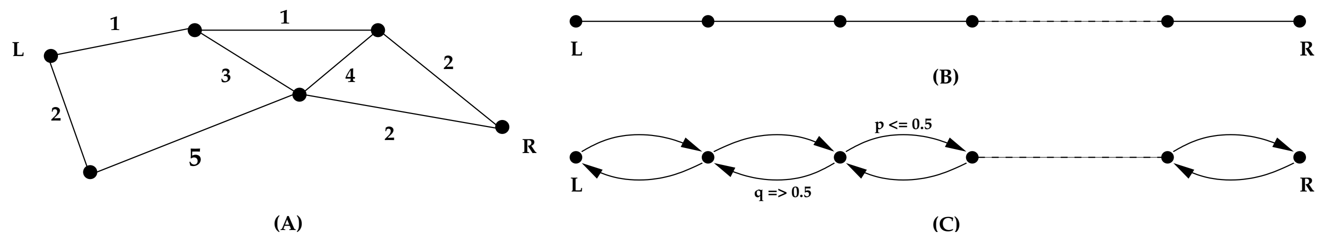

Figure 2: Shows three scenarios in which random walks are performed. In an undirected graph (weighted as in (A), or unweighted as in (B)), for any two nodes L and R, $T(R, L) + T(L, R)$ is related to the "resistance" between them. In case (C) (when $p \leq q$ at every node), $T(R, L)$ is less than that in (B) because of an increased "attraction" towards L at every node.

$T(R, L)$ are $O(n^2)$.

Figure 2(C) shows a slightly modified version of that in Figure 2(B), where the graph is directed, although it is still linear. Moreover, there are weights associated with edges which are interpreted as probabilities in the random walk; and the weight on $\langle s, s_{left} \rangle$ is, in general, not equal to that on $\langle s, s_{right} \rangle$. Here, s is some node in the graph, and s_{left} and s_{right} are respectively the nodes occurring immediately to the left and right of it. However, we are guaranteed that the probability of moving to the left at any node is greater than that of moving to the right (i.e. $p \leq q$). Given this scenario, it is easy to see that the expected time of arrival at the left end point (L), starting at the right end point (R), is also $O(n^2)$ (if there are n nodes in all). Informally, this is because at every node, there is an increased "attraction" to the left compared to that in Figure 2(B); and the expected arrival time can only be less than that in the latter.

Simple Temporal Problems Revisited

In this section, we will provide two different kinds of algorithms for solving STPs. The first algorithm (which we will only briefly review) is based on the computation of shortest paths (as shown in (Dechter *et al.* 1991)). The second algorithm is based on the properties of *random walks* on *directed* graphs. We will then compare the strengths and weaknesses of these two algorithms, and eventually (in the next section), combine the intuitions behind the working of these two different algorithms to develop strongly polynomial-time algorithms for solving RDTPs.

Figure 3 provides a simple *deterministic* procedure for solving STPs based on the computation of shortest paths. Central to this algorithm is the notion of a *distance graph* $D(\mathcal{G})$ associated with an STP $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ (see step (1) of Figure 3). An edge $\langle X_i, X_j \rangle$ in the *distance graph* is annotated with a real number w (instead of temporal bounds), and encodes the constraint $X_j - X_i \leq w$. Therefore, every edge in the STP is compiled to 2 edges in the *distance graph*. The following Lemma then characterizes the consistency of an STP.

Lemma 1: A consistent schedule exists for $X_0, X_1 \ldots X_N$ in $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ if and only if the *distance graph* $D(\mathcal{G})$ does not contain any negative cycles.

Proof: (see (Dechter *et al.* 1991)).

The running time of the algorithm in Figure 3 is similar to that of the Bellman-Ford algorithm for computing single-source shortest paths (in the presence of negative weights on edges), and is equal to $O(N|\mathcal{E}|)$.[1]

Figure 4 presents a pseudo-polynomial-time *randomized* algorithm for solving STPs. Central to this algorithm is the relationship between simple temporal constraints and *random walks* on *directed* graphs. Temporarily, we will assume that all the specified bounds in the STP are integers with absolute value $\leq B$.

The idea is to start with any integer assignment to all the events, and use the violated constraints in every iteration to guide the search for the true assignment A^* (if it exists). In particular, in every iteration, a violated constraint is chosen, and the assignment of one of the two participating variables is either increased or decreased by 1 unit. Since we know that the true assignment A^* satisfies all constraints, and therefore the chosen one too, *randomly* moving along one of the axes (in the direction of the feasible region), will reduce the L_1-distance between the current assignment A and A^* with a probability ≥ 0.5.[2] The geometry of a violated constraint is shown in Figure 3. Much like the *random walk* in Figure 2(C), therefore, we can bound the convergence time to A^* by a quantity that is only quadratic in the maximum L_1-distance between any two complete assignments.

Lemma 2: If all the numbers are integers with absolute values $\leq B$, then there exists a solution A^* having integer time schedules for all the events.

Proof: From the previous Lemma, we know that if there exists a solution, one of them is given by assigning to each X_i ($1 \leq i \leq N$), the length of the shortest path from X_0 to X_i. Since all the numbers are integers, so are the lengths of the shortest paths from X_0 to all X_i, hence establishing the truth of the Lemma.

Lemma 3: The L_1-distance between any two integer assignments $A = \langle X_1 = x_1, X_2 = x_2 \ldots X_N = x_n \rangle$ and $A' = \langle X_1 = x'_1, X_2 = x'_2 \ldots X_N = x'_n \rangle$ (in the above context) is at most $2N^2B$, and 0 if and only if $A = A'$.

Proof: Consider the L_1-distance $|x_1 - x'_1| + |x_2 - x'_2| \ldots |x_N - x'_N|$. Because the absolute value of all the bounds $\leq B$, and at most N numbers can contribute to the

[1] Any negative cycle (inconsistency in the simple temporal constraints) is detected by the Bellman-Ford algorithm.

[2] The L_1-distance between two assignments $A = \langle X_1 = x_1, X_2 = x_2 \ldots X_N = x_n \rangle$ and $A' = \langle X_1 = x'_1, X_2 = x'_2 \ldots X_N = x'_n \rangle$ is equal to $|x_1 - x'_1| + |x_2 - x'_2| \ldots |x_N - x'_N|$.

ALGORITHM: SOLVE-STP-DETR
INPUT: An STP $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$.
OUTPUT: A solution s (if it exists).
(1) Construct the *distance graph* $D(\mathcal{G})$ on the nodes of $\mathcal{G}$ as follows:
(a) For every edge $e = \langle X_i, X_j \rangle \in \mathcal{E}$:
(A) Add the edge $\langle X_i, X_j \rangle$ annotated with $UB(e)$.
(B) Add the edge $\langle X_j, X_i \rangle$ annotated with $-LB(e)$.
(2) For every X_i:
(a) Compute $dist(X_0, X_i)$ (shortest path length) from X_0 to X_i in $D(\mathcal{G})$.
(3) RETURN: $s = \{X_i \leftarrow dist(X_0, X_i)\}$.
END ALGORITHM

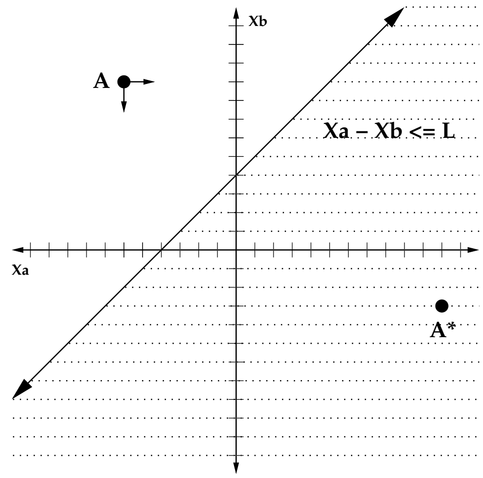

Figure 3: The left side of the figure shows a *deterministic* algorithm for solving STPs based on shortest path computations. The right side of the figure illustrates the geometry of a violated simple temporal constraint ($X_a - X_b \leq L$). A is the current assignment, and A^* is the required (integral) solution.

ALGORITHM: SOLVE-STP-RAND
INPUT: An STP $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ with all the specified bounds being integers of absolute value $\leq B$.
OUTPUT: A solution s (if it exists).
(1) For $i = 1$ to N:
(a) Set X_i to a random integer in $[-B, B]$.
(2) While there exists a violated constraint of the form $(L \leq X_b - X_a \leq U)$:
(a) If $X_b - X_a < L$:
(A) Do one of the following with equal probabilities:
(ONE) $X_b = X_b + 1$.
(TWO) $X_a = X_a - 1$.
(b) If $X_b - X_a > U$:
(A) Do one of the following with equal probabilities:
(ONE) $X_b = X_b - 1$.
(TWO) $X_a = X_a + 1$.
(3) RETURN: $s =$ the current assignment to all variables.
END ALGORITHM

Figure 4: Shows a pseudo-polynomial-time *randomized* algorithm for solving STPs.

length of any shortest path in the *distance graph*, all the terms are $\leq 2NB$. This means that the L_1-distance is always $\leq N(2NB) \leq 2N^2B$. Further, since all the terms are ≥ 0, the L_1-distance can be 0 only when all the individual terms are 0—which in turn, happens only when A and A' are identical.

Lemma 4: For any violated constraint, step (2) in Figure 4 reduces the L_1-distance between A (current assignment) and A^* (integral solution) with a probability ≥ 0.5.

Proof: When there exists a violated constraint, some inequality of the form $(X_a - X_b \leq L)$ is not satisfied. We know that A^* is placed within the feasible region of this constraint, and the current assignment A is in the other half-plane (see Figure 3). In step (2), we *randomly* move towards the feasible region of the constraint (by 1 unit) along one of the two axes (X_a or X_b). For any point in the feasible region, at least one of these moves reduces the L_1-distance to it. Further, since the initial assignment I is integral, and the step size is 1 unit, the current assignment A, in any iteration, is guaranteed to be integral. Finally, since A^* is integral, and the step size is equal to the smallest possible integer increment (decrement), the L_1-distance between A and A^* is decreased by at least 1 with a probability ≥ 0.5, and increased by at most 1 with a probability ≤ 0.5.

Lemma 5: The *expected* number of iterations of the algorithm 'SOLVE-STP-RAND' is $O(N^4B^2)$.

Proof: From Lemma 3, we know that the maximum L_1-distance between the initial random assignment I, and the true satisfying assignment A^*, is $O(N^2B)$. Further, in every iteration, we perform a *random walk* exactly analogous to that in Figure 2(C)—with the left end-point being A^*, I being only as far as the other end-point, and a maximum of $O(N^2B)$ nodes in between. The truth of the Lemma then follows from the properties of *random walks* on *directed* graphs.

From Lemma 5, we have that the *expected* running time of 'SOLVE-STP-RAND' is $O(N^4|\mathcal{E}|B^2)$ (since checking for a violated constraint in every iteration takes $O(|\mathcal{E}|)$ time).[3]

One clear advantage of the first algorithm is that its running time is strongly polynomial. However, the second algorithm has the advantage that it can be extended to handle other types of constraints too (which the first one cannot). In particular, it can handle the kinds of disjunctive temporal constraints as shown in Figure 5. (A) and (B) are respectively the Type 2 and Type 3 disjunctions allowed in

[3]Even when *randomized* algorithms are analyzed only in terms of their *expected* running time, Markov's inequality yields that the probability that we do not terminate even after k (say 100) times the *expected* number of time steps is $\leq 1/k$ ($\leq 1/100$).

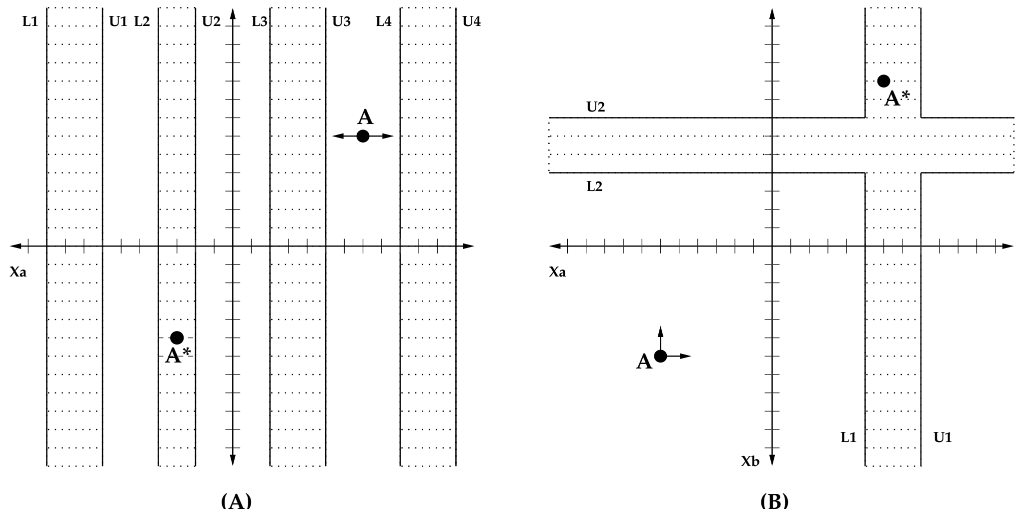

Figure 5: Shows two other kinds of constraints that can be handled by *random walk* strategies. (A) and (B) respectively correspond to Type 2 and Type 3 disjunctions allowed by RDTPs. In both cases, there exist two directions at all infeasible points such that moving along at least one of them (by 1 unit) decreases the L_1-distance to the solution (A^*), no matter where it is placed in the feasible region of the constraints.

ALGORITHM: SOLVE-RDTP
INPUT: An RDTP with all the specified constants and bounds being integers of absolute value $\leq B$.
OUTPUT: A solution s (if it exists).
(1) For $i = 1$ to N:
  (a) Set X_i to a random integer in $[-B, B]$.
(2) While there exists a violated constraint:
  (a) If it is of the form $(L \leq X_b - X_a)$:
    (A) Do one of the following with equal probabilities:
      (ONE) $X_b = X_b + 1$.
      (TWO) $X_a = X_a - 1$.
  (b) If it is of the form $(L_1 \leq X_a \leq U_1) \vee \ldots (L_k \leq X_a \leq U_k)$:
    (A) Do one of the following with equal probabilities:
      (ONE) $X_a = X_a + 1$.
      (TWO) $X_a = X_a - 1$.
  (c) If it is of the form $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_b \leq U_2)$:
    (A) Do one of the following with equal probabilities:
      (ONE) $X_a = X_a + 1$ if ($X_a < L_1$), and $X_a = X_a - 1$ otherwise.
      (TWO) $X_b = X_b + 1$ if ($X_b < L_2$), and $X_b = X_b - 1$ otherwise.
(3) RETURN: $s =$ the current assignment to all the variables.
END ALGORITHM

Figure 6: Shows a pseudo-polynomial-time *randomized* algorithm for solving RDTPs.

RDTPs. It is easy to see that in both these cases, no matter where A^* lies within the feasible region of the constraint, there exist two directions at every infeasible point such that moving along at least one of them reduces the L_1-distance between A (the current infeasible assignment), and A^*. Figure 5 shows these required pairs of directions with respect to (violated) Type 2 and Type 3 constraints. We note again that if all the numbers are integers with absolute value $\leq B$, and the step size is 1, A and A^* (one of the solutions) are guaranteed to be integral. Figure 6 provides a pseudo-polynomial-time *randomized* algorithm for solving RDTPs, and is a simple extension of that in Figure 4.

Strongly Polynomial-time Algorithms for RDTPs

In this section, we will design strongly polynomial-time algorithms for solving RDTPs by pulling together ideas drawn from both the above presented algorithms for solving STPs. We will first present a strongly polynomial-time *deterministic* algorithm, and then provide an extremely simple *randomized* algorithm—the time and space complexity of which is much less than that of the *deterministic* algorithm.

In both these algorithms, we will cast an RDTP as a "disjunct selection problem" (see (Oddi and Cesta 2000) and (Stergiou and Koubarakis 1998)), and therefore model it as a meta-CSP. In particular, we will associate the meta-variables $\mathcal{Y} = \{Y_1, Y_2 \ldots Y_Q\}$ with the Type 2 constraints, and the meta-variables $\mathcal{Z} = \{Z_1, Z_2 \ldots Z_R\}$ with the Type 3 constraints. That is, if $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_a \leq U_2) \ldots (L_T \leq X_a \leq U_T)$ is a Type 2 constraint with the variable Y_j associated with it, then the domain of Y_j is $D_{Y_j} = \{(L_1 \leq X_a \leq U_1), (L_2 \leq X_a \leq U_2) \ldots (L_T \leq X_a \leq U_T)\}$. Similarly, if $(L_1 \leq X_a \leq U_1) \vee (L_2 \leq X_b \leq U_2)$ is a Type 3 constraint with the variable Z_j associated with it, then the domain of Z_j is $D_{Z_j} = \{(L_1 \leq X_a \leq U_1), (L_2 \leq X_b \leq U_2)\}$. The goal is now to find an instantiation of the variables in $\mathcal{Y} \cup \mathcal{Z}$ such that, together with the Type 1 constraints, the induced set of simple temporal constraints is consistent.

For notational convenience, we will refer to the disjunct $(L_1 \leq X_a \leq U_1)$ as $X_a \in [L_1, U_1]$. For any interval $I = [L, U]$, we will denote its left end-point (viz. L) by $\mathcal{L}(I)$, and its right end-point (viz. U) by $\mathcal{R}(I)$. We will also assume that for Type 2 constraints, the disjuncts are arranged in ascending order of the end points of their corresponding intervals.[4] We will refer to these natural orderings on the domains of variables in $\mathcal{Y}$ as their *nominal* orderings, and show that it plays a crucial role in the working of both the strongly polynomial-time algorithms that follow. For a Type 2 constraint $(X_a \in [L_1, U_1]) \vee (X_a \in [L_2, U_2]) \ldots (X_a \in [L_T, U_T])$ with the attached meta-variable $Y_i \in \mathcal{Y}$, we will use V_{Y_i} to denote the variable occurring in the disjunction—namely, X_a. Also, we will use a constraint interchangeably with its $(0, 1)$-matrix representation. A *binary* constraint between variables W_1 and W_2 using particular orderings on their domains, is represented as a 2D $(0, 1)$-matrix with the '1's and '0's respectively indicating the *allowed* and the *disallowed* tuples. Finally, we will use the notation $dist(X_i, X_j)$ to indicate the distance from X_i to X_j in the *distance graph* resulting from compiling only the Type 1 constraints.

A Strongly Polynomial-time Deterministic Algorithm

In this subsection, we will provide a strongly polynomial-time *deterministic* algorithm for solving RDTPs (see Figure 7). Central to the algorithm is the notion of *bounded minimal conflicts*, and the relationship between the resulting *binary* constraints and *CRC (connected row-convex)* constraints (see (Deville *et al.* 1999)). The following Lemmas rigorously establish this relationship, and prove the correctness of the algorithm in Figure 7.

Lemma 6: An instantiation of the variable $W \in \mathcal{Y} \cup \mathcal{Z}$ to the disjunct $X_a \in I$ requires us to successfully add the edges $\langle X_0, X_a \rangle$ annotated with $\mathcal{R}(I)$, and $\langle X_a, X_0 \rangle$ annotated with $-\mathcal{L}(I)$ to the *distance graph* (resulting from the Type 1 constraints) without creating a negative cycle.

Proof: If we have to ensure that the variable X_a is in the interval I, we have to make sure that $X_a - X_0 \leq \mathcal{R}(I)$, and $X_a - X_0 \geq \mathcal{L}(I)$. Retaining the semantics of the *distance graph*—where the constraint $X_j - X_i \leq w$ is specified by the edge $\langle X_i, X_j \rangle$ annotated with w—this corresponds to the addition of the edges $\langle X_0, X_a \rangle$ annotated with $\mathcal{R}(I)$, and $\langle X_a, X_0 \rangle$ annotated with $-\mathcal{L}(I)$, to the *distance graph* without creating an inconsistency (which, by Lemma 1, is characterized by the presence of a negative cycle).

Definition 1 (*conflicts and minimal conflicts*): A *conflict* is an instantiation of a set of variables in $\mathcal{Y} \cup \mathcal{Z}$ that results in an inconsistency with the Type 1 constraints. A *minimal conflict* is a *conflict* no proper subset of which is also a *conflict*.

Lemma 7: An instantiation of a set of variables in $\mathcal{Y} \cup \mathcal{Z}$ is consistent if and only if there is no subset of them that constitutes a *minimal conflict*.

Proof: By definition of a *conflict*, an instantiation of a set of variables in $\mathcal{Y} \cup \mathcal{Z}$ is consistent if and only if there is no subset of them that constitutes a *conflict*. Further, the truth of the Lemma follows from the fact that a set of events constitutes a *conflict* if and only if some subset of them constitutes a *minimal conflict*.

Lemma 8: The size of every *minimal conflict* is ≤ 2.

Proof: Suppose we try to instantiate a set of variables $W_1, W_2 \ldots W_h$ in $\mathcal{Y} \cup \mathcal{Z}$. Since instantiating any meta-variable $W_i \in \mathcal{Y} \cup \mathcal{Z}$ requires committing to some variable X_{W_i} to be within some interval I_{W_i}, we would have to add the following edges to the *distance graph*: $\langle X_0, X_{W_p} \rangle$ annotated with $\mathcal{R}(I_{W_p})$, and $\langle X_{W_p}, X_0 \rangle$ annotated with $-\mathcal{L}(I_{W_p})$ (for all $1 \leq p \leq h$). We will refer to these edges as "special" edges. Knowing that the *distance graph* initially does not contain any negative cycles (because any inconsistency in the Type 1 constraints can be caught right away), if a negative cycle is newly created, it must involve one of the "special" edges. Since all "special" edges have X_0 as an end point, the negative cycle must contain X_0. Further, since a fundamental cycle can have any node repeated at most once, at most 2 "special" edges can be present in a newly created negative cycle. Finally, since "special" edges correspond to the instantiation of variables in $\mathcal{Y} \cup \mathcal{Z}$, the size of a *minimal conflict* is ≤ 2.

Lemma 9: RDTPs constitute a *binary* CSP over the meta-variables $\mathcal{Y} \cup \mathcal{Z}$.

Proof: From the previous Lemma, we know that the size of a *minimal conflict* is ≤ 2. This means that either the *conflicts* are of size 1 or of size 2. The enumeration of all size-2 *conflicts* results in a *binary* CSP. Further, the size-1 *conflicts* need not be enumerated explicitly because they are just reflected as domain values not consistent with any instantiation of any other variable. Hence, step (2) in Figure 7 is justified—establishing the truth of the Lemma.

Lemma 10: Consider the *binary* constraint between $Y_i \in \mathcal{Y}$ and $Y_j \in \mathcal{Y}$. Under the *nominal* domain orderings for Y_i and Y_j, the '1's in any row or column appear consecutively (see Figure 8(A)).

Proof: We will only prove this Lemma for rows (assuming that the domain values of Y_i constitute the rows, and those of Y_j constitute the columns). Proving the Lemma for columns is exactly symmetric. Suppose there is some row where a '0' appears in between two '1's. That is, suppose $V_{Y_i} \in I_{(i,h)}$ conflicts with $V_{Y_j} \in I_{(j,k_2)}$, but does not conflict with $V_{Y_j} \in I_{(j,k_1)}$ and $V_{Y_j} \in I_{(j,k_3)}$ (for some h and $k_1 < k_2 < k_3$). The fact that $V_{Y_i} \in I_{(i,h)}$ does not conflict with $V_{Y_j} \in I_{(j,k_1)}$ implies that $\mathcal{R}(I_{(j,k_1)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h)}) \geq 0$ and $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_1)}) \geq 0$. Similarly, $\mathcal{R}(I_{(j,k_3)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h)}) \geq 0$ and $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_3)}) \geq 0$. A *conflict* between $V_{Y_i} \in I_{(i,h)}$ and $V_{Y_j} \in I_{(j,k_2)}$ implies that $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_2)}) < 0$ or $\mathcal{R}(I_{(j,k_2)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h)}) < 0$. The former cannot be true because $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_3)}) \geq 0$ and $\mathcal{L}(I_{(j,k_3)}) > \mathcal{L}(I_{(j,k_2)})$. Similarly, the latter cannot be true

[4] For example, the Type 2 constraint $(X_1 \in [7, 9]) \vee (X_1 \in [4, 6]) \vee (X_1 \in [1, 2]) \vee (X_1 \in [3, 5])$ would first be reduced to $(X_1 \in [7, 9]) \vee (X_1 \in [1, 2]) \vee (X_1 \in [3, 6])$, and then be rewritten as $(X_1 \in [1, 2]) \vee (X_1 \in [3, 6]) \vee (X_1 \in [7, 9])$.

ALGORITHM: SOLVE-RDTP-DETR
INPUT: An RDTP over the events $\{X_1, X_2 \ldots X_N\}$.
OUTPUT: A solution s (if it exists).
(1) Cast RDTP as a disjunct selection problem using meta-variables $\mathcal{Y} = \{Y_1, Y_2 \ldots Y_Q\}$ and $\mathcal{Z} = \{Z_1, Z_2 \ldots Z_R\}$ for Type 2 and Type 3 constraints respectively.
(2) For every W_1 and W_2 in $\mathcal{Y} \cup \mathcal{Z}$:
 (a) Build a *binary* constraint as follows:
 (A) An instantiation of disjuncts to W_1 and W_2 is disallowed, if and only if, together with Type 1 constraints, they introduce a negative cycle in the underlying *distance graph*.
(3) Solve these *binary* constraints using the procedure for solving CRC (connected row-convex) constraints.
(4) RETURN: $s =$ SOLVE-STP-DETR (induced STP).
END ALGORITHM

Figure 7: Shows a strongly polynomial-time *deterministic* algorithm for solving RDTPs.

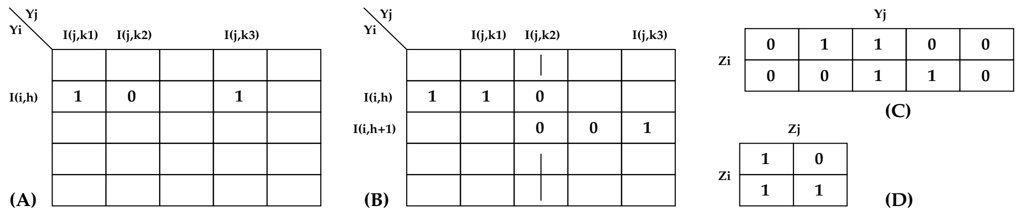

Figure 8: Shows a few diagrams to support and illustrate some of the arguments made in the proofs of Lemmas 10 to 13.

because $\mathcal{R}(I_{(j,k_1)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h)}) \geq 0$ and $\mathcal{R}(I_{(j,k_2)}) > \mathcal{R}(I_{(j,k_1)})$. By contradiction, therefore, the truth of the Lemma is established.

Lemma 11: Consider a *binary* constraint between $W_1 \in \mathcal{Y} \cup \mathcal{Z}$ and $W_2 \in \mathcal{Y} \cup \mathcal{Z}$. Under the *nominal* domain orderings for variables in $\mathcal{Y}$, and any domain orderings for variables in $\mathcal{Z}$, the '1's in any row or column appear consecutively.

Proof: From the previous Lemma, we know that this is true when $W_1, W_2 \in \mathcal{Y}$. When $W_1 \in \mathcal{Z}$ and $W_2 \in \mathcal{Y}$, a simple rewriting of the proof of the previous Lemma shows that all the '1's appear consecutively in any row (column) if the domain values of W_2 constitute the columns (rows). Further, since the domain size of W_1 is 2, no matter how many '1's appear in every column (row), they always appear consecutively (see Figure 8(C)). Finally, when $W_1, W_2 \in \mathcal{Z}$, the statement is trivially true for any 2×2 matrix (see Figure 8(D)).

Lemma 12: Consider the *binary* constraint between $Y_i \in \mathcal{Y}$ and $Y_j \in \mathcal{Y}$. Under the *nominal* domain orderings for Y_i and Y_j, and for some h and $k_1 < k_2 < k_3$, if (a) $V_{Y_i} \in I_{(i,h)}$ does not conflict with $V_{Y_j} \in I_{(j,k_1)}$, (b) $V_{Y_i} \in I_{(i,h)}$ conflicts with $V_{Y_j} \in I_{(j,k_2)}$, (c) $V_{Y_i} \in I_{(i,h+1)}$ conflicts with $V_{Y_j} \in I_{(j,k_2)}$, and (d) $V_{Y_i} \in I_{(i,h+1)}$ does not conflict with $V_{Y_j} \in I_{(j,k_3)}$, then the column $V_{Y_j} \in I_{(j,k_2)}$ does not contain any '1's (see Figure 8(B)).

Proof: Since $V_{Y_i} \in I_{(i,h)}$ does not conflict with $V_{Y_j} \in I_{(j,k_1)}$, we have (1) $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_1)}) \geq 0$ and (2) $\mathcal{R}(I_{(j,k_1)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h)}) \geq 0$. Since $V_{Y_i} \in I_{(i,h)}$ conflicts with $V_{Y_j} \in I_{(j,k_2)}$, we have (3) $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_2)}) < 0$ or $\mathcal{R}(I_{(j,k_2)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h)}) < 0$. Since $V_{Y_i} \in I_{(i,h+1)}$ conflicts with $V_{Y_j} \in I_{(j,k_2)}$, we have (4) $\mathcal{R}(I_{(i,h+1)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_2)}) < 0$ or $\mathcal{R}(I_{(j,k_2)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h+1)}) < 0$. Since $V_{Y_i} \in I_{(i,h+1)}$ does not conflict with $V_{Y_j} \in I_{(j,k_3)}$, we have (5) $\mathcal{R}(I_{(i,h+1)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_3)}) \geq 0$ and (6) $\mathcal{R}(I_{(j,k_3)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h+1)}) \geq 0$. Consider the disjunction in (3). From (2), it reduces to (7) $\mathcal{R}(I_{(i,h)}) + dist(V_{Y_i}, V_{Y_j}) - \mathcal{L}(I_{(j,k_2)}) < 0$ (because $\mathcal{R}(I_{(j,k_2)}) > \mathcal{R}(I_{(j,k_1)})$). Consider the disjunction in (4). From (5), it reduces to (8) $\mathcal{R}(I_{(j,k_2)}) + dist(V_{Y_j}, V_{Y_i}) - \mathcal{L}(I_{(i,h+1)}) < 0$ (because $\mathcal{L}(I_{(j,k_2)}) < \mathcal{L}(I_{(j,k_3)})$). Now consider any entry in the column of $V_{Y_j} \in I_{(j,k_2)}$. From (7), we have that $V_{Y_j} \in I_{(j,k_2)}$ conflicts with $I_{(i,e)}$ for any $e < h$ (because $\mathcal{R}(I_{(i,e)}) < \mathcal{R}(I_{(i,h)})$). Similarly, from (8), we have that $V_{Y_j} \in I_{(j,k_2)}$ conflicts with $I_{(i,e)}$ for any $e > h + 1$ (because $\mathcal{L}(I_{(i,e)}) > \mathcal{L}(I_{(i,h+1)})$). Putting these together, the truth of the Lemma is established.

Lemma 13: Consider a *binary* constraint between $W_1 \in \mathcal{Y} \cup \mathcal{Z}$ and $W_2 \in \mathcal{Y} \cup \mathcal{Z}$. Under the *nominal* domain orderings for variables in $\mathcal{Y}$, and any domain orderings for variables in $\mathcal{Z}$, the positions of '1's in two consecutive rows (columns) intersect, or touch each other (after removing empty rows and columns).

Proof: From the previous Lemma, we know that this is true when $W_1, W_2 \in \mathcal{Y}$. From Lemma 11, it is also easily seen to be true when $W_1 \in \mathcal{Z}$ and $W_2 \in \mathcal{Y}$ (because the domain size of W_1 is only 2 (see Figure 8(C))). Finally, when $W_1, W_2 \in \mathcal{Z}$, the statement is trivially true for any 2×2 matrix (see Figure 8(D)).

The above Lemmas establish that all the resulting meta-level *binary* constraints are CRC. A *binary* constraint is CRC if, after removing empty rows and columns (rows or columns that do not contain any '1's), the '1's appear consecutively in every row and column (see Lemma 11), and the positions of the '1's in any two consecutive rows or columns intersect, or touch each other (see Lemma 13). Unlike row-convex constraints, CRC constraints are closed under *composition*, *intersection* and *transposition* (the three basic operations of algorithms that enforce *path-consistency* in a *binary* constraint network)—hence establishing that

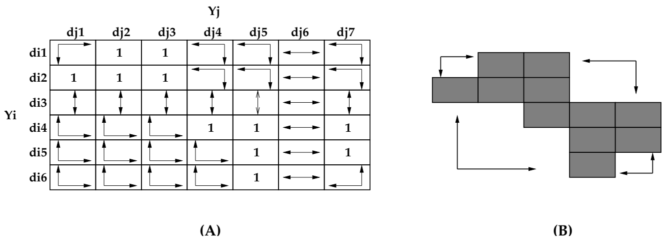

Figure 9: Illustrates the geometry of a CRC constraint. (A) shows the matrix representation of a CRC constraint with the required pair of directions marked against each '0' (not shown explicitly for clarity). (B) illustrates the general pattern of the required pair of directions for '0's in a CRC constraint (shaded areas indicate feasible regions).

path-consistency over CRC constraints is sufficient to ensure global consistency. An instantiation of the generic path-consistency algorithm that further exploits the structure of CRC constraints has a running time complexity of $O((Q + R)^3 d_{max}^2)$, and a space complexity of $O((Q + R)^2 d_{max})$ (see (Deville *et al.* 1999)). Here, $Q = |\mathcal{Y}|$, $R = |\mathcal{Z}|$, and d_{max} is the maximum number of disjuncts in any constraint. The total running time of the algorithm is therefore $O((Q+R)^3 d_{max}^2 + (Q+R)^2(N|\mathcal{E}| + d_{max}^2))$. The first term measures the cost of solving the CRC constraints, and the second term measures the cost of building the *binary* constraints in the first place (by computing the pair-wise shortest paths in the *distance graph*).

A Strongly Polynomial-time Randomized Algorithm

Figure 10 presents an extremely simple *randomized* algorithm for solving CRC constraints. Central to this algorithm is the observation that in the matrix representation of a CRC constraint, the following is true: "At every '0', there exist *two* directions such that with respect to every other '1', moving along *at least one* of these directions *decreases* the *manhattan distance* to it" (see Figure 9). From the theory of *random walks* on *directed* graphs, we know that the *expected* number of iterations of 'SOLVE-CRC-RAND' is only $O(N^2K^2)$ (where K is the size of the largest domain). The *expected* running time of the algorithm, however, is $O(N^2K^2M)$ (the factor M arises due to the inner loop of the procedure, where we are required to repeatedly check for the presence of a violated constraint).

We will now show how we can significantly reduce the above factor M by employing appropriate data structures. We exploit the fact that it is sufficient for us to consider *any* violated constraint in every iteration. This is because every violated constraint is CRC, and gives us a chance to move closer to the solution with a probability ≥ 0.5. Figure 11 presents a diagrammatic illustration of the required data structures. A series of doubly linked lists are maintained. The list '*All*' contains all the constraints, and for every variable X_i, a list L_i is maintained. L_i contains exactly those constraints that variable X_i participates in. Further, a list of satisfied constraints ('*Sat*'), and a list of unsatisfied constraints ('*Unsat*') are also maintained. These lists are updated incrementally in every iteration, and in the beginning, are built in accordance with the initial assignment I. Additions to '*Sat*' or '*Unsat*' are always made at the beginning of the lists, and therefore take constant time. Similarly, deletion from '*Sat*' or '*Unsat*' (given a pointer to the element to be deleted) takes constant time (because the lists are doubly linked, and deletion can be realized by linking together the neighbors of the element to be deleted).

In every iteration, the first constraint in '*Unsat*' is chosen, and the assignment of one of the two variables participating in it is changed. The only constraints that can be affected by this are the ones in which this variable appears. Walking through the corresponding list of all such constraints, we check each one of them for being satisfied or not. If a constraint under consideration was originally satisfied and is now unsatisfied (or vice-versa), we perform the appropriate addition and deletion operations on the '*Sat*' and '*Unsat*' lists. Both these operations can be done in constant time, and the complexity of the update procedure is therefore equal to the number of elements in the list (that contains all the constraints the chosen variable participates in).

If the maximum number of constraints any variable participates in is d (corresponds to the degree of the constraint network), the running time of the *randomized* algorithm for solving CRC constraints can be reduced to $O(N^2K^2d)$. This is less than that of the *deterministic* algorithm for solving CRC constraints (see (Deville *et al.* 1999)). In the worst case too, d is only as large as N, and the running time is $O(N^3K^2)$—equaling that of the *deterministic* algorithm, but with a much lesser space complexity. These arguments, in turn, suggest an extremely simple *randomized* algorithm for solving RDTPs—the time and space complexity of which is less than that of the *deterministic* algorithm. We also note that the *randomized* algorithm circumvents the use of complex data structures otherwise required for optimally implementing path-consistency subroutines, etc.

Figure 12 presents an algorithm for solving RDTPs without explicitly building the CRC constraints. The following Lemma establishes the correctness of the algorithm (implicitly also establishing the truth of the above quoted property of CRC constraints).

Lemma 14: For any violated constraint, step (5) in Figure 12 reduces the *manhattan distance* between the current assignment A, and the true (satisfying) assignment A^*, with a probability ≥ 0.5. (Note that A and A^* are complete assignments to all the variables in $\mathcal{Y} \cup \mathcal{Z}$.)

Proof: From Lemma 9, we know that all the meta-level constraints are *binary*. A violated constraint is therefore one between some 2 variables $W_1, W_2 \in \mathcal{Y} \cup \mathcal{Z}$. If $W_1, W_2 \in \mathcal{Y}$, then one of them (say W_1) contributes the incoming edge to X_0—annotated with $-\mathcal{L}(I_{k_2})$, and the other (W_2) contributes the outgoing edge from X_0—annotated with $\mathcal{R}(I_{k_1})$. Certainly, increasing the rank of the value assigned to W_1, and decreasing that of W_2, will only decrease the weight of the negative cycle. Therefore, at least one of decreasing the rank of the value assigned to W_1, or increasing the rank of the value assigned to W_2, will decrease the *manhattan distance* to A^*. Similarly, if $W_1, W_2 \in \mathcal{Z}$, *ran-*

ALGORITHM: SOLVE-CRC-RAND
INPUT: A CSP over N variables $\{X_1, X_2 \dots X_N\}$, and M CRC constraints $\{C_1, C_2 \dots C_M\}$.
OUTPUT: A solution to the CSP.
(1) Let the ordered domain of the variable X_i be D_i viz. $\langle d_{(i,1)}, d_{(i,2)} \dots d_{(i,|D_i|)} \rangle$.
(2) Start with an initial random assignment I to all the variables.
(3) While the current assignment A violates some CRC constraint $C(X_i, X_j)$: (with the domain values of X_i constituting the rows, and the domain values of X_j constituting the columns)
(a) Let $d_{(i,k_1)}$ and $d_{(j,k_2)}$ be the current assignments to the variables X_i and X_j respectively.
(b) Let $\{e_1, e_2\}$ be the direction pair associated with the entry $\langle X_i, X_j \rangle = \langle d_{(i,k_1)}, d_{(j,k_2)} \rangle$ in $C(X_i, X_j)$.
(c) Choose p uniformly at random from $\{e_1, e_2\}$.
(d) If $p = \mathcal{LF}$: set X_j to $d_{(j,k_2-1)}$.
(e) If $p = \mathcal{RT}$: set X_j to $d_{(j,k_2+1)}$.
(f) If $p = \mathcal{DN}$: set X_i to $d_{(i,k_1+1)}$.
(g) If $p = \mathcal{UP}$: set X_i to $d_{(i,k_1-1)}$.
END ALGORITHM

Figure 10: A simple *randomized* algorithm for solving CRC constraints. The symbols $\mathcal{LF}$, $\mathcal{RT}$, $\mathcal{DN}$ and $\mathcal{UP}$ indicate the directions *left* (decrease the rank of the assignment to X_j), *right* (increase the rank of the assignment to X_j), *down* (increase the rank of the assignment to X_i) and *up* (decrease the rank of the assignment to X_i) respectively.

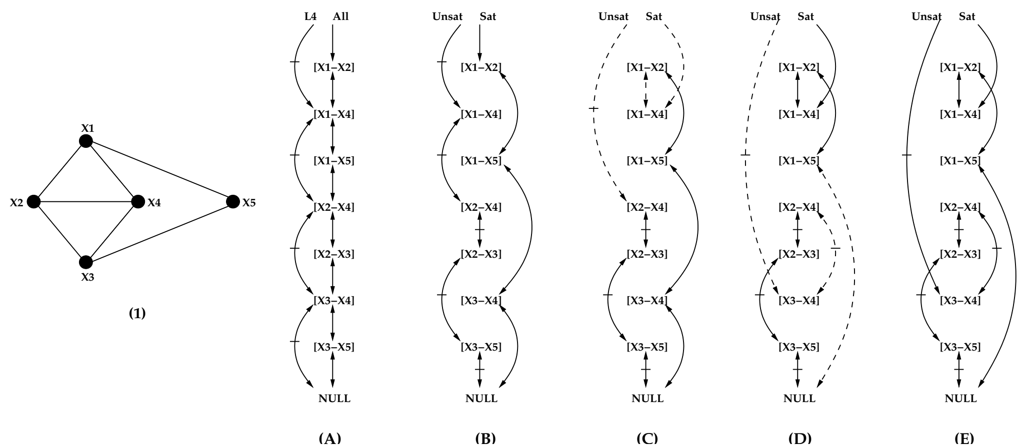

Figure 11: Illustrates the data structures (and the operations performed on them) to reduce the running time of the *randomized* algorithm for solving CRC constraints. (1) shows the constraint network of an example *binary* CSP on 5 variables. (A) shows two doubly linked lists, '*All*' and 'L_4'. The pointers in 'L_4' are distinguished from those of '*All*' by using a small horizontal mark on them. (B) shows the two doubly linked lists, '*Sat*' and '*Unsat*'. The pointers in '*Unsat*' are distinguished from those of '*Sat*' by using a small horizontal mark on them. (C) shows how the lists '*Sat*' and '*Unsat*' are updated when the first unsatisfied constraint in (B) (viz. $C(X_1, X_4)$)) is chosen, and the variable X_4 happens to be reassigned, possibly now satisfying $C(X_1, X_4)$. (D) shows what happens when $C(X_2, X_4)$ remains unsatisfied, but $C(X_3, X_4)$ changes from being satisfied to being unsatisfied. (E) shows the final lists of satisfied and unsatisfied constraints. Note that the ordering of the constraints is inconsequential in all the lists.

domly reassigning W_1 or W_2 to the other disjunct achieves the same effect. If, however, $W_1 \in \mathcal{Y}$ and $W_2 \in \mathcal{Z}$, then either the correct assignment for W_2 is the other disjunct, or the correct assignment for W_1 is the one that can potentially increase the weight of the negative cycle (depending on whether it contributes the incoming edge or the outgoing edge). Therefore, *randomly* doing one of these will decrease the *manhattan distance* to A^* by 1 with a probability ≥ 0.5. It is now easy to see that all these cases are compactly represented and taken care of in step (5) of Figure 12.

Applications to General DTPs

The above algorithms for solving RDTPs can also be useful in the context of solving general DTPs. In particular, large parts of the search space can be pruned easily when partial instantiations to some of the variables induce sub-problems that look like RDTPs. Figure 13 shows an example of a DTP, the search space of which can be pruned significantly when sub-problems resemble RDTPs.

Conclusions and Future Work

We described a class of metric temporal problems, which we referred to as RDTPs, that formed a middle ground between STPs and DTPs. We showed that RDTPs could be solved in polynomial time, and could encode limited, but very useful, forms of temporal disjunctions that would otherwise require an exponential search space. We provided both *deterministic* and *randomized* algorithms for efficiently solving RDTPs—with the latter having much better time and space complexities compared to the former. The expressive power of RDTPs, along with their tractability, makes them a suitable model for many real-life applications that involve metric temporal reasoning. As part of our future work, we are trying to incorporate and reason with preferences attached to

ALGORITHM: SOLVE-RDTP-RAND
INPUT: An RDTP over the events $\{X_1, X_2 \ldots X_N\}$.
OUTPUT: A solution s (if it exists).
(1) Cast RDTP as a disjunct selection problem using meta-variables $\mathcal{Y} = \{Y_1, Y_2 \ldots Y_Q\}$ and $\mathcal{Z} = \{Z_1, Z_2 \ldots Z_R\}$ for Type 2 and Type 3 constraints respectively.
(2) Let $\mathcal{H}$ be the set of all variables that participate in any of the Type 2 or Type 3 constraints.
(3) For all X_a and X_b in $\mathcal{H}$:
  (a) Compute $dist(X_a, X_b)$ in the *distance graph* induced by the Type 1 constraints.
(4) Start with an initial random assignment I to all the variables in $\mathcal{Y} \cup \mathcal{Z}$.
(5) While $\exists$ a negative cycle in the induced *distance graph*:
  (a) If the negative cycle is of the form $\mathcal{R}(I_{k_1}) + dist(X_a, X_b) - \mathcal{L}(I_{k_2}) < 0$:
    (A) Do one of the following with equal probabilities:
      (ONE) [work on the outgoing edge from X_0]
        — If $X_a \in I_{k_1}$ came from assigning some variable $Y_i \in \mathcal{Y}$, increase the *rank* of the assignment to Y_i.
        — If $X_a \in I_{k_1}$ came from assigning some variable $Z_i \in \mathcal{Z}$, assign the other disjunct to Z_i.
      (TWO) [work on the incoming edge to X_0]
        — If $X_b \in I_{k_2}$ came from assigning some variable $Y_j \in \mathcal{Y}$, decrease the *rank* of the assignment to Y_j.
        — If $X_b \in I_{k_2}$ came from assigning some variable $Z_j \in \mathcal{Z}$, assign the other disjunct to Z_j.
(6) RETURN: $s =$ SOLVE-STP-DETR (induced STP under the current assignment of values to variables in $\mathcal{Y} \cup \mathcal{Z}$).
END ALGORITHM

Figure 12: Shows an extremely simple *randomized* algorithm for solving RDTPs.

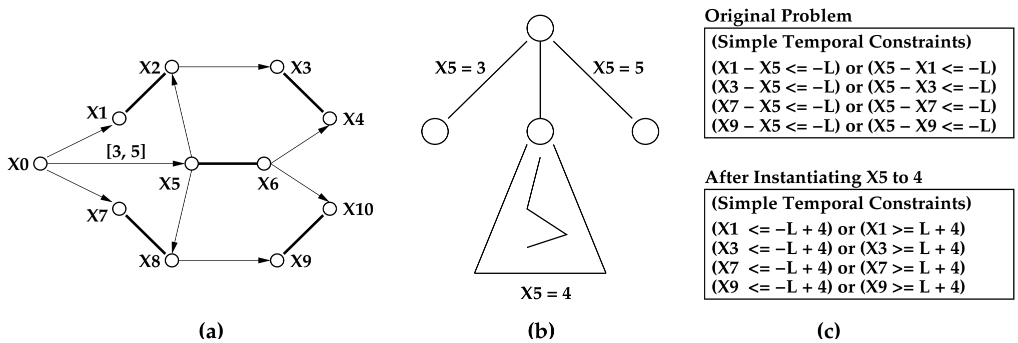

Figure 13: Illustrates how polynomial-time procedures for solving RDTPs can be exploited in pruning the search space for solving general DTPs. (a) shows 5 actions (indicated by dark lines), each of a fixed length L. The simple temporal constraints between them are indicated by lighter lines, and for clarity, the bounds on them are not shown explicitly (although for this example, we assume them to be integers). If the action $\langle X_5, X_6 \rangle$ competes for (different) resources with each of the other actions, the resulting DTP has a search space of size 16. On the other hand, if the search problem is cast as one of finding integer schedules for all the events (because all the temporal bounds are integers), X_5 should be assigned one of 3, 4 or 5. Instantiating it with any of these values induces an RDTP—hence resulting in a search space of size only 3 (see (b) and (c)).

temporal constraints. (Kumar 2004) presents a few relevant results—providing a polynomial-time algorithm for solving a restricted class of such problems.

References

Armando A., Castellini C. and Giunchiglia E. SAT-based Procedures for Temporal Reasoning. *ECP 1999.*

Dechter R., Meiri I. and Pearl J. Temporal Constraint Networks. *Artificial Intelligence, Vol. 49, 1991, pp. 61-95.*

Deville Y., Barette O. and Van Hentenryck P. Constraint Satisfaction over Connected Row-Convex Constraints. *Artificial Intelligence, 109(1-2):243-271, 1999.*

Doyle P. G. and Snell E. J. Random Walks and Electrical Networks. *Carus Math. Monographs 22, Math. Assoc. Amer., Washington, D. C. 1984.*

Kumar T. K. S. Incremental Computation of Resource-Envelopes in Producer-Consumer Models. *CP 2003.*

Kumar T. K. S. A Polynomial-time Algorithm for Simple Temporal Problems with Piecewise Constant Domain Preference Functions. *AAAI 2004.*

Nguyen X. and Kambhampati S. Reviving Partial Order Planning. *IJCAI 2001.*

Oddi A. and Cesta A. Incremental Forward Checking for the Disjunctive Temporal Problem. *ECAI 2000.*

Smith D., Frank J. and Jonsson A. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review 15:1, 2000.*

Stergiou K. and Koubarakis M. Backtracking Algorithms for Disjunctions of Temporal Constraints. *In the 15th National Conference on Artificial Intelligence, 1998.*

Tsamardinos I. and Pollack M. E. Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems. *Artificial Intelligence, 151(1-2):43-90, 2003.*