Multi-Agent Plan Repair With DTPs

Pieter Buzing*, Adriaan ter Mors and Cees Witteveen

Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands,
{p.buzing, a.w.termors, c.witteveen}@ewi.tudelft.nl

Abstract

A planning problem can be transformed into a (temporal) constraint satisfaction problem called a Disjunctive Temporal Problem (DTP). Many planning problems have a distributed nature: multiple parties are involved in the construction of a global plan. We discuss a multi-agent architecture where each agent acts as a DTP constraint solver responsible for the consistency of his own problem partition.

Though there are efficient algorithms for the Distributed Constraint Satisfaction Problem (DCSP), they can not be directly applied to a DTP. This is because multiple agents have to reach consensus over *which* simple constraint in a disjunctive constraint to pursue.

In this paper we propose a new method that combines the planning capabilities that DTPs offer with the distributed power found in DCSP algorithms. Our flexible representation and solutions provide efficient tools for on-line plan repair.

1 Introduction

Due to limited resource availability and high efficiency requirements, planning and scheduling constitute crucial activities in many complex processes like transportation and production. For example, take an airport where a constant flow of arriving and departing flights have to be coordinated; or think of a factory where production costs have to be minimized and (shared) resources like machines and tools have to be used to their full potential.

One complicating requirement in many such planning problems is that often different autonomous parties are involved. These entities – also called *actors* or *agents* – have their own interests, resources and concerns. For example, in multi-modal transportation problems, different companies are involved in the transportation of goods or passengers, each having their own interests, business policies and planning tools. This implies that centralized planning for these actors is often not possible and, therefore, planning control should be distributed over the actors themselves. Note that this does not imply that these actors can make their plan independently from the others: since a joint plan has to be solved, often these problems are tightly coupled.

In this paper we will therefore discuss an approach to *distributed* planning for autonomous actors that have to solve a joint planning problem. In particular, we will concentrate on temporal planning and scheduling problems, where disruptions

*This research is/was supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs.

during the execution phase are common and on-line plan repair is needed. Throughout this article we use a representation where time points (corresponding to events like the beginning or ending of actions) are variables and temporal intervals form the constraints between the variables.

Where there are multiple agents, we assume that time points (the variables) are distributed over $n > 1$ actors. Each of the actors has to determine suitable values for these variables. Such a problem is also known as a (temporal) constraint satisfaction problem (CSP). In this paper, we will concentrate on two special variants of the temporal CSP problem: the Simple Temporal Problem (STP) [3] and the Disjunctive Temporal Problem (DTP) [10]. While it is well known that CSPs can be solved in a distributed manner (for literature on Distributed Constraint Satisfaction Problems (DCSPs) we refer to [14]), DTPs pose some extra challenges as opposed to general DCSPs.

Finding a solution to a DTP comes down to taking one disjunct in each DTP disjunction and then solving this STP projection. This means that a DTP is actually a meta-CSP where the disjuncts are the variables and the consistency of the STP projection is the constraint to be fulfilled. The difference with general DCSPs is that the variables in the meta-CSP (i.e., disjuncts) are not owned by a single agent, but by a group of agents. This means that two subproblems have to be addressed. First, agents have to reach consensus about which disjunct to instantiate. Second, when no disjunct produces a consistent STP, backtracking has to be applied. Each backtracking step involves revising a previously chosen value assignment. In a multi-agent setting, the values are chosen by autonomous agents, and therefore there is no straightforward order in which backtracking steps must be taken.

To tackle the first problem, we must formulate a coordination protocol that results in choosing a disjunct to instantiate, while taking into account the preferences that individual agents may have for different disjuncts. In this paper we show how a Clarke-Tax voting procedure can be used to implement such a coordination protocol. To address the problem of how to backtrack when the current set of disjunct instantiations is infeasible, we make use of Asynchronous Weak-commitment Search [14].

The new method we obtain by using the above techniques combines the planning capabilities of DTPs with the distributed power found in DCSP algorithms. The relevance of our contribution is presented in the context of plan repair. The DTP formalism is extended with preference information [7] about constraints: each agent can (privately) express an amount of preference for values within a time interval. With this subtle extension, we demonstrate (using a detailed example) that during the execution phase, plans can still be adjusted with low time and coordination costs.

This paper is structured as follows: first a detailed description of the STP and DTP formalism is given (Section 2), along with a comprehensive example in the domain of air traffic control. The application of our work to plan repair is presented next in Section 3. Finally, in Section 4, we present how the two main techniques Clarke-tax voting and Asynchronous Weak-commitment Search can be applied to the running example of Section 2.

2 Temporal Networks

A (temporal) planning problem can be expressed as a constraint satisfaction problem $G = (V, E)$, where G is a graph with vertices V (a set of time point variables) and edges E (a set of constraints between the time points). A solution to such a temporal constraint problem is an assignment of (time) values to each of the time points, such that all constraints are satisfied. In this section we first discuss the Sim-

ple Temporal Problem framework, followed by the more expressive formalism called the Disjunctive Temporal Problem (DTP). The concluding subsection provides an example for its application.

2.1 Simple Temporal Problem (STP)

A Simple Temporal Problem (STP)[3] consists of a number of time points (corresponding to the start and end points of actions) and a set of binary constraints $a \leq X - Y \leq b$ expressing that the difference between the time points X and Y should differ minimally a and maximally b for some $a, b \in \mathbb{Z}$. These intervals can be used to specify the order in which actions have to occur, or the minimum amount of delay between events. In this article we adopt an interval notation, instead of the inequalities: $X - Y \in [a, b]$.

An STP can be solved by calculating the all-pairs distance graph (*d-graph*). The d-graph is a complete directed graph where each edge is labeled with the shortest path length between variables. The length of a path from i to j via a number of nodes $(i_0 = i, i_1, i_2, \ldots, i_k = j)$ is the sum of the distances between the nodes, thus $d(X_i, X_j) \leq \sum_{j=1}^{k} d(a_{i_{j-1}, i_j})$. The d-graph is often represented as a table and the earliest and latest possible instantiations of time points can be read directly from this shortest path table. The STP is consistent if there are no negative cycles in the d-graph, i.e. when constraints imply that the latest possible occurrence of an event is *before* the earliest possible occurrence. The table can be created with e.g. the *Floyd-Warshall* all-pairs-shortest-paths algorithm [5] in $O(n^3)$ time where n is the number of time points (variables) involved. An alternative method based on triangulation is Xu's $\triangle STP$ algorithm [12]. It has the same worst-case complexity as Floyd-Warshall, but avoids futile propagation of constraints over irrelevant graph components.

2.2 Disjunctive Temporal Problem (DTP)

Most planning problems, however, pose ordering choices ("action A should start after action B is finished, or action B should start after action A is done") or multiple time intervals ("action C can be applied before lunch, or after lunch"). The STP formalism is not powerful enough to express (and solve) these kind of problems. A Disjunctive Temporal Problem is designed to represent these kind of choices. The general form of a DTP is a disjunction of simple constraints: $C_i : X_1 - Y_1 \in [a_1, b_1] \vee \ldots \vee X_k - Y_k \in [a_k, b_k]$. We will refer to the simple constraint $X_j - Y_j \in [a_j, b_j]$ as disjunct d_j, or in matrix representation $C[i][j]$, i.e. the j^{th} disjunct of the i^{th} constraint.

A common strategy for solving a DTP is to regard it as a meta-CSP in which the constraints C_1 to C_k are variables and disjuncts d_1 to d_k are values. The m-ary constraint in this CSP is that all instantiated disjuncts result in a consistent STP. The problem is then to find an instantiation of disjuncts in each constraint, such that the resulting STP is consistent.

The general setup of DTP algorithms is as follows: one-by-one constraints are instantiated and the consistency of the STP is checked. If an inconsistent STP is encountered the algorithm backtracks (or backjumps) by undoing a constraint choice and instantiating an alternative disjunct. This is repeated until all constraints are instantiated (and the STP is consistent) or no consistent instantiation can be made. See e.g. [10, 11] for further refinements on this approach.

2.3 Example

As a guiding example we introduce a simple airport with 1 runway and 6 aircraft that have to depart. In Table 1(a) the initial data is presented in the form of two attributes: weight and R-CTOT. The Refined Calculated Take-Off Time (R-CTOT) is issued by the Central Flow Management Unit (CFMU) and determines the allocated time slot for each flight. More specifically, the R-CTOT is a time point with a $[-3, 3]$ deviation window in which the aircraft is allowed to depart. As can be seen in Table 1(a) the R-CTOTs are not unique, so the air traffic controller (referred to as ATC) has the task to create a conflict-free departure sequence.

Each aircraft belongs to a weight class, which is either Light, Medium, or Heavy. Since the take-off process of an aircraft creates a considerable amount of turbulence (called wake vortex) a separation time between adjacent departures has to be respected. This separation time is either 1, 2, or 3 minutes and depends mainly on the weight of the two adjacent aircraft: if the first aircraft is heavier than the following one the separation tends to be high, and if the leading aircraft is lighter than the second one a small separation time is needed. Table 1(b) shows separation values (in minutes) that we will use throughout this example.

The ETD column is reserved for the Estimated Time of Departure that ATC assigns to each aircraft. We will refer to the ETDs as the planning. The First Time of Departure (FTD) is communicated by the pilot to the ATC when he is about to clear the gate. Basically it says whether the pilot will actually make his ETD deadline.

#	Weight	R-CTOT	FTD	ETD
1	L	12:00		
2	H	12:00		
3	M	12:00		
4	L	12:05		
5	M	12:05		
6	H	12:05		

(a) Initial data: weight and R-CTOT

		second		
		H	M	L
First	H	2	2	3
	M	1	2	3
	L	1	1	2

(b) Separation table

Table 1: There are six flights to be scheduled. The weight is an important factor for the determination of separation times between aircraft. R-CTOT is the middle of a six minute interval in which the aircraft should depart, issued by CFMU. These numbers should lead to a planning by ATC in the form of ETD times. Note that the FTDs (provided by the pilots) remain unknown until minutes before the planned take-off.

The full list of constraints is given in Table 2. Here we see the 15 constraints induced by the separation regulations and the 6 constraints caused by the R-CTOT times. X_1 up to X_6 stand for the departure time (ETD) of the six aircraft; R_1 and R_2 are the R-CTOT times. Note that the separation constraints have a disjunctive form, so a choice has to be made whether aircraft 1 departs before or after aircraft 2.

In our example the different disjuncts in a constraint involve the same variables which makes it a Temporal Constraint Satisfaction Problem (TCSP)[3], but the general case is referred to as Disjunctive Temporal Problem (DTP).

In Figure 1(a) we see the constraints of Table 2 in a graphical form. Note that this figure shows a particular projection of the DTP, i.e. one disjunct in each constraint has been instantiated[1]. Applying constraint propagation leads to the

[1]We refer to Dechter's Forward and Go-back algorithm [3] as the basic backtracking algorithm

i	d=1	d=2
1	$(X_1, X_2) \in [1, \infty]$	$(X_1, X_2) \in [-\infty, -3]$
2	$(X_1, X_3) \in [1, \infty]$	$(X_1, X_3) \in [-\infty, -3]$
3	$(X_1, X_4) \in [2, \infty]$	$(X_1, X_4) \in [-\infty, -2]$
4	$(X_1, X_5) \in [1, \infty]$	$(X_1, X_5) \in [-\infty, -3]$
5	$(X_1, X_6) \in [1, \infty]$	$(X_1, X_6) \in [-\infty, -3]$
6	$(X_2, X_3) \in [2, \infty]$	$(X_2, X_3) \in [-\infty, -1]$
7	$(X_2, X_4) \in [3, \infty]$	$(X_2, X_4) \in [-\infty, -1]$
8	$(X_2, X_5) \in [2, \infty]$	$(X_2, X_5) \in [-\infty, -1]$
9	$(X_2, X_6) \in [2, \infty]$	$(X_2, X_6) \in [-\infty, -2]$
10	$(X_3, X_4) \in [3, \infty]$	$(X_3, X_4) \in [-\infty, -1]$
11	$(X_3, X_5) \in [2, \infty]$	$(X_3, X_5) \in [-\infty, -2]$
12	$(X_3, X_6) \in [1, \infty]$	$(X_3, X_6) \in [-\infty, -2]$
13	$(X_4, X_5) \in [1, \infty]$	$(X_4, X_5) \in [-\infty, -3]$
14	$(X_4, X_6) \in [1, \infty]$	$(X_4, X_6) \in [-\infty, -3]$
15	$(X_5, X_6) \in [1, \infty]$	$(X_5, X_6) \in [-\infty, -2]$
16	$(R_1, X_1) \in [-3, 3]$	
17	$(R_1, X_2) \in [-3, 3]$	
18	$(R_1, X_3) \in [-3, 3]$	
19	$(R_2, X_1) \in [-3, 3]$	
20	$(R_2, X_2) \in [-3, 3]$	
21	$(R_2, X_3) \in [-3, 3]$	

Table 2: The constraints with their disjunctions. The i^{th} constraint in a DTP is represented as $C[i]$ and the d^{th} disjunct of constraint i as $C[i][d]$.

minimal constraint network shown in Figure 1(b). Not all triangulated constraints (called fill-arcs) are shown; only $(A3, R_2)$ which states that it has been derived that aircraft 3 has to depart between 3 and 7 minutes before 12:05.

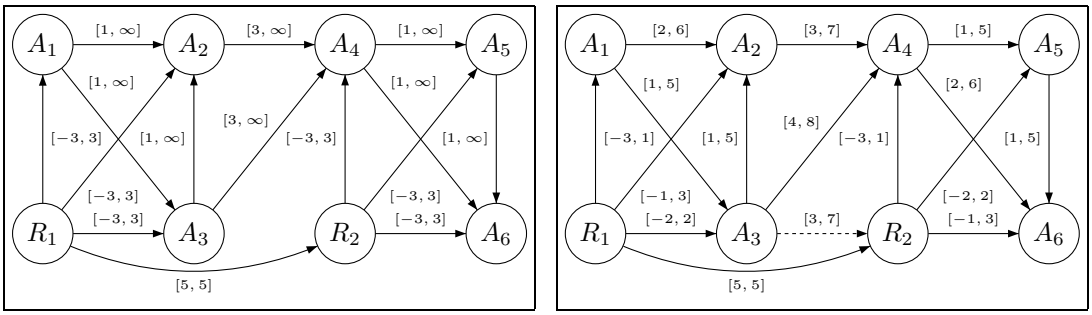
Extracting a schedule from Figure 1(b) is easy now. We take R_1 and R_2 as fixed time points – respectively 12:00 and 12:05 – and add the earliest possible occurrence of $A1$ through $A6$. This produces departure times of ETD_1 = 11:57, ETD_2 = 11:59, ETD_3 = 11:58, ETD_4 = 12:02, ETD_5 = 12:03, ETD_6 = 12:04. Of course also the latest possible times could be used to establish a schedule. Note that an individual aircraft can be scheduled at any time within its constraints (e.g. ETD_1 = 12:00), but this restricts the possibilities of the other planes and another round of constraint propagation is needed in order to guarantee consistency.

3 Preferences and Repair

Many problem domains have a certain amount of uncertainty, i.e. during the actual execution phase unexpected events may occur. This means that plans have to be adapted. In our ATC example the actual time that an aircraft can reach the runway is influenced by all sorts of passenger delays, luggage issues, taxiway congestion or technical complications that threaten the correct execution of a plan. Small delays are very common on an airport and small plan repairs are everyday activities for the controllers.

It seems that more flexibility should be expressed in the (temporal) constraints, but general CSP methods are not designed to handle soft constraints.

and step around the details in this example.



(a) Original constraints

(b) Minimal graph

Figure 1: The (separation) constraints between the planes (a) and the resulting minimal graph (b) after applying Floyd-Warshall.

3.1 Preference Information

To address this lack of refined information Khatib et al.[7] added a preference function to the STP formalism. This function f defines a preference value for each time point within an interval, i.e., $\langle X_j - X_i \in [a, b], f : x \in [a, b] \rightarrow A \rangle$ where A is a (finite) set of preference values in the range $[0, max]$. Infeasible time values (i.e. not included in the interval range) are defined to have preference 0.

An STP with preference functions is referred to as a STPP. One strategy to find the "best" solution for a STPP is to apply binary search to the space of preference levels [9]. Because Floyd-Warshall's algorithm has to be called (at most) $\log |A|$ times the complexity of solving a STPP becomes $O(n^3 \log |A|)$. Note that the *optimal* solution should be interpreted in terms of Weakest Link Optimality: all constraints are fulfilled with at least preference value p_{opt}.

These preference functions can also be applied to the constraints in a Disjunctive Temporal Problem. Peintner [8] presents the following Solve-DTPP algorithm:

- First it finds a feasible instantiation of disjuncts for each constraint at the lowest preference level. This can be done with Dechter's Forward and Go-back algorithm [3].
- Then a higher level projection of the current STPP is checked for consistency, and if so the algorithm tries again a higher preference level.
- This is repeated until inconsistency is reached and the highest consistent STPP solution is returned.

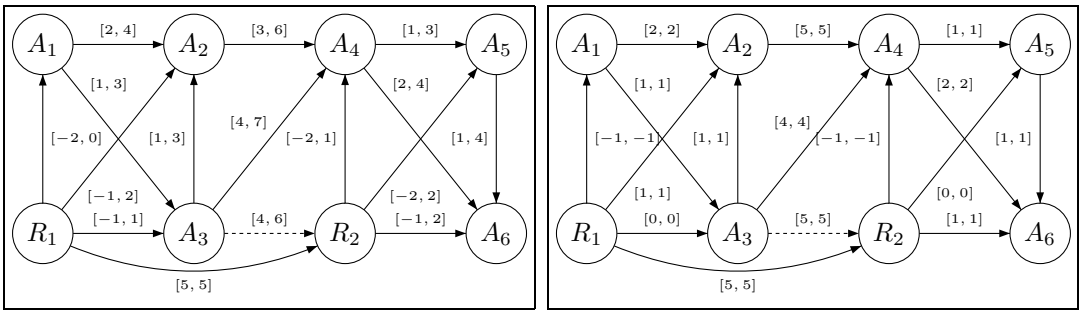
Solve-DTPP can be executed with binary search for efficiency, resulting in a time complexity of $O(d^m n^3)$, where d is the maximum number of disjuncts per constraint, m is the number of constraints, and n is the number of variables.

Example (cont.) The R-CTOT times provided by CFMU are in fact target times for departures. CFMU wants the aircraft to leave at these specific times because it optimizes the European air space, i.e., it balances the sector load. A small deviation, however, is acceptable as long as the departure is within the 6-minute window.[2] For simplicity we can define CFMU's preference function for the ETD as follows: $f(-3) = 1$, $f(-2) = 2$, $f(-1) = 3$, $f(0) = 4$, $f(1) = 3$, $f(2) = 2$, $f(1) = 2$,

[2]Actually, departures outside of the R-CTOT window are accepted, but this is handled by another protocol.

$f(3) = 1$. Function f only accepts integer time values and returns 0 for values smaller than -3 or greater than 3.

When we use this function to calculate the preference value of the extracted schedule ($ETD_1 = 11{:}57$, $ETD_2 = 11{:}59$, $ETD_3 = 11{:}58$, $ETD_4 = 12{:}02$, $ETD_5 = 12{:}03$, $ETD_6 = 12{:}04$) we find that no aircraft is assigned the target time and that for flights 1 and 4 the lowest preference values apply. Even though all constraints are fulfilled, this schedule seems to be a bad one. The question arises whether the planner can find a schedule that fulfills the constraints at a higher preference value.



(a) Preference level 2

(b) Preference level 3

Figure 2: The constraints from Table 2 remain consistent at preference level 2 (a) and preference level 3 (b).

The minimal graph shown in Figure 1(b) was created with constraints at preference level 1. The constraints can be projected at preference level of 2 by intersecting CFMU's constraints with the interval $[-2, 2]$. This also results in a consistent graph (see Figure 2(a)). Even a projection at preference level 3 is consistent, as can be seen in Figure 2(b). Unfortunately, the projection at the highest preference level (i.e., 4) results in an inconsistent graph, which will not surprise the reader, since the constraints at level 3 are already maximally tight.

By applying the constraint propagation mechanism a number of times, we discovered that the six aircraft can be scheduled at an optimal preference level of $p_{opt} = 3$. The schedule that can be extracted from the optimal constraint graph (i.e., Figure 2(b)) is the following: $ETD_1 = 11{:}59$, $ETD_2 = 12{:}01$, $ETD_3 = 12{:}00$, $ETD_4 = 12{:}04$, $ETD_5 = 12{:}05$, $ETD_6 = 12{:}06$.

3.2 Plan Repair

If we look back at the proposed algorithms for STPPs and DTPPs some interesting opportunities for plan repair come to mind. The main issue is that if a plan is consistent at some preference level p_{opt} the plans encountered at level $p < P_{opt}$ are just a relaxation of the optimal plan. This means that if some constraint is changed the agents may still find a solution at a lower preference level. The additional space required for maintaining all visited temporal networks (i.e. the d-graphs) if binary search is used is just $O(\log |A|)$. The time complexity of retrieving such a back-up solution is of course constant.

In recent work [1] we proposed a method for small plan repair actions in a multi-agent setting. It basically comprises a distributed STP algorithm called $\triangle STP$ (described by Xu and Choueiry in [12]) that iteratively builds up a number of constraint graphs at different preference levels. When (in the after-planning phase) a constraint is changed, the agents investigate if they can return to a lower preference

level solution. This distributed STPP algorithm is limited in the sense that it can only handle simple constraints.

The next step is of course a similar repair scheme for a distributed DTPP. Peintner has shown that his Solve-DTPP algorithm can solve centralized DTPPs; Extending his approach to the multi-agent case poses some additional design problems, however. The main problem is that constraints are shared between agents, so coordination is needed to choose the disjuncts. In the next section we will present our solutions to overcome the coordination obstacles. An example of plan repair in a distributed DTPP will be presented at the end of subsection 4.2.

4 Distributed DTP

Solving an STP in a distributed setting is fairly straightforward because each agent knows which constraints have to be fulfilled. This is not the case in DTPs, since agents have to choose which disjunct of a constraint to instantiate. DTPs are (generally) non-binary, meaning that different time points – and therefore different agents – are involved. This implies that the choice of which disjunct to instantiate is not up to one single agent but to a group. This necessitates a new coordination level on top of the constraint propagation mechanism. Here we choose a Clarke-Tax voting scheme to implement this coordination mechanism. In addition, we have to solve the problem of how to backtrack in case no disjunct yields a consistent network. This is done using Asynchronous Weak-Commitment Search [13].

Note that our goal is not to design a unifying framework for temporal planning and multi-agent systems, but simply to enable coordination between the planning agents. This means that other (Distributed AI) techniques are also feasible, depending on the particular application.

4.1 Clarke-Tax Voting

We assume that every agent is aware of all (disjunctive) constraints it is involved in and that he has a preference value for each of the disjuncts in those constraints. There are n agents involved in a constraint with k disjuncts; possibly an agent is responsible for more than one time point. If only one agent is involved in a disjunctive constraint the voting becomes a trivial procedure.

The Clarke-Tax mechanism [2, 4] is a closed-bid voting procedure, where each party places a bid for all of the possible states or options – here disjuncts d_1 to d_k. The bid of an agent i is a vector of scores $\langle s^i_1, \ldots, s^i_k \rangle$ and $\sum_{j=1}^{k} s^i_j = 0$. Additionally, a minimal and a maximal score value have to be defined; here we adopt a score range of $[-1, 1]$. After collecting all bids a winner is appointed: disjunct d_w with a score $s_w = \text{argmax}_w \sum_{i=1}^{n} s^i_w$. [3] Now, for each agent i the scores are recounted *without* the bid of this agent. If this results in a different winner ($d_{w'}$ with a score $s^{-i}_{w'}$ against a score of s^{-i}_w, where s^{-i} is the sum without agent i's score), agent i influenced the voting with a magnitude of $s_w - s^{-i}_{w'}$. This difference is the Clarke-Tax that agent i has to pay. A nice property of this mechanism is that there is only one dominant strategy, namely to be honest.

Note: the only reason that a tax is levied is to ensure that the agents will not try to manipulate the voting process. In the remainder of this article we will not address the issue of *how* these taxes are to be collected. However, we do want to add that these taxes should be distributed over agents outside of the voting group, since it must not play a role in the agent's decision process.

[3] In the case of a tie a random number generator should appoint the winner. Note that predefined tie-breaking rules could influence the bidding strategy of the agents.

Example (Cont.) Let's look at constraint $C[1]$ in Table 2: $(X_1, X_2) \in [1, \infty] \vee (X_1, X_2) \in [-\infty, -3]$. The agents involved in this constraint are aircraft 1 (A1) and aircraft 2 (A2). First we consider A1's bids. Essentially the two options are departing first (letting A2 wait for at least 1 minute) or letting A2 depart first and wait for 3 minutes or more. It is clear that the former is more attractive to A1. Of course it is completely up to A1 to assign relative ratings to the two options, but we assume that A1 has a preference function which basically says that small separation times are preferred (since they lead to an efficient algorithm that most parties will agree on), but leaving before the other plane is also preferred. A function that expresses such preferences over time could be:

$$f(t) = \begin{cases} \frac{max}{t+1} & \text{if } t \geq 0 \\ \frac{max}{-2*t+1} & \text{if } t < 0 \end{cases}$$

where max is the maximal preference value; in our example $max = 4$. The factor 2 is an arbitrary value, which expresses the degree of asymmetry between positive and negative time values. A1's rating for the first constraint ($(X_1 - X_2) \in [1, \infty]$) is $r_1 = 4/2 = 2$ and A1's rating for the second constraint ($(X_1, X_2) \in [-\infty, -3]$) is $r_2 = 4/7 = 0.57$. We can normalize these values to scores in the range $[-1, 1]$ with the formula

$$s_i = \frac{2 * r_i}{\sum_{j=1}^{n} r_j} - 1$$

The bid of A1 would thus be $\langle s_1, s_2 \rangle = \langle 0.56, -0.56 \rangle$.

The calculation for A2 goes along the same line, although the difference is that A2 only has to wait 1 minute if the voting has the "wrong" outcome. This means that – if he is rational – A2 will not bid very high for his preferred disjunct. His ratings[4] are respectively $r_1 = 4/4 = 1$ and $r_2 = 4/3 = 1.33$. The normalized scores for A2 would be $\langle s_1, s_2 \rangle = \langle -0.14, 0.14 \rangle$.

agent i	**bids**		**sum without** i		**tax for** i
	d_1	d_2	d_1	d_2	
A1	.56	-.56	-.14	.14	.28
A2	-.14	.14	.56	-.56	0
sum	.42	-.42			

Table 3: The outcome of the Clarke-Tax voting. Agent A1 has to pay a tax of $.42 - .14 = .28$ because if he had not voted the alternative d_2 would have won; .28 is the difference that A1 made.

The agents place their bids and the results are in Table 3. Option d_1 is the clear winner. Now what if each agent had not voted? Withdrawing A1's bids would of course leave only A2's offer $\langle -0.14, 0.14 \rangle$ so the outcome would change into d_2's favor. Placing the bid resulted in a sum of .42 for the winner (d_1), while *not* voting would have given a winning sum of .14 for option d_2. Apparently, agent A1 made a difference of $.42 - .14 = .28$. Agent A2 is not fined with a tax because if he would withdraw his bid the outcome would still be the same.

The symmetry in Table 3 is caused by the small scale of this example, i.e., only two agents are involved with only two bids. More elaborate votings would result in multiple agents paying different amounts of tax.

4.2 Asynchronous Weak-Commitment Search

Backtracking (or backjumping) is a standard procedure in CSP algorithms: a variable that cannot find a valid value (i.e., consistent with the other constraints) trig-

[4]For simplicity, we use the same function for both agents.

gers a *previously* instantiated variable to withdraw its choice and insert another value. However, in a multi-agent system such an ordering is not available.

Asynchronous Weak-Commitment Search (Asynchronous WS) [14] is a Distributed Constraint Satisfaction (DCSP) technique that dynamically assigns a backtracking ordering to the variables. This ordering is achieved by assigning a priority value to each variable. A variable instantiation then has to be consistent with higher priority variables. In the case of equal priority values, the alphabetical ordering of (the identifiers of) the variables will break the tie. Each variable has a priority value that is initially zero. If a variable cannot be instantiated with a feasible value the following steps will be taken:

- A no-good set containing all violated variable-value pairs of neighboring variables (two variables are neighbors if they share a constraint) is made.
- This no-good set is communicated to the variable with a higher priority. The priority value of the inconsistent variable is incremented.
- The higher-priority variable is then instantiated with another value that doesn't violate the no-good set.

In the case of DTPs, the "neighbor" is actually a number of agents associated to a (disjunctive) constraint. If they are informed by a lower-priority group that they have to withdraw their disjunct, the agents look again at their preference bids and as these scores are now public, the new disjunct to instantiate is clear for all parties.

Example (Cont.) We assume that the departure times extracted from Figure 2(b) have been assigned to the aircraft. At 11:30, however, the pilot of aircraft 1 communicates to the air traffic controller that he will not make his ETD of 11:59. Instead, his first time of departure (FTD) will be 12:02. Returning to a lower preference level will not solve this acute planning problem, since the minimal constraint graph in Figure 1(b) shows that aircraft 1 can leave maximally 1 minute after its R-CTOT time. That is, the constraint $(R_1, X_1) \in [-3, 1]$ is incompatible with the new constraint $(R_1, X_1) \in [2, \infty]$.

The first repair option is simply to replace the violated constraint with a different disjunct – using the ordering that resulted from the voting procedure. However, since the constraint between R_1 and X_1 has no alternative disjuncts to instantiate (see Table 2, constraint C_{16}), repair has to be achieved through backtracking.

A no-good set is constructed and communicated upwards in the priority list. This set does not have to be minimal, but we assume that the agent guarding X_1 constructs a no-good set with the disjuncts that he is involved in: $\{\langle(X_1, X_2), [1, \infty]\rangle, \langle(X_1, X_3), [1, \infty]\rangle\}$. The agent responsible for R_1 may construct a different no-good set: $\{\langle(R_1, X_2), [-3, 3]\rangle, \langle(R_1, X_3), [-3, 3]\rangle\}$. Both agents exchange their (partial) sets and merge them into a mutual no-good set: $\{\langle(X_1, X_2), [1, \infty]\rangle, \langle(X_1, X_3), [1, \infty]\rangle, \langle(R_1, X_2), [-3, 3]\rangle, \langle(R_1, X_3), [-3, 3]\rangle\}$. Note that the latter two constraints (i.e., C_{17} and C_{18}) have a domain of size 1. This means that they play no role in the backtracking process and can be ignored without consequences.

The no-good set is communicated to the agents involved in the disjunctive constraint that has a higher priority. If we assume that all priority values are still 0, the alphabetic ordering identifies $C_2 : (X_1, X_3)$ as the next disjunctive constraint to be re-assigned. Since agent A1 is already aware of the no-good set only the agent responsible for variable X_3 (i.e., agent A3) will be informed. Additionally, the priority value of constraint C_{16} is incremented.

Constraint C_2 is instantiated with its alternative disjunct (i.e., $(X_1, X_3) \in [-\infty, -3]$) and tested for consistency. This new disjunct indeed yields a (locally)

consistent temporal network. The new constraint value of C_2 is communicated to neighboring agents guarding lower priority constraints. It can be easily verified that the lower priority constraints are still consistent.

The disjunct replacement in this example means that the departure order changes: $[A1, A3, A2]$ now becomes $[A3, A1, A2]$. Also the departure times are slightly adapted, but only by a few minutes. This plan repair was achieved with only local interaction.

5 Conclusion

In this paper we demonstrate a method that can solve the Disjunctive Temporal Problem in a distributed setting. Additionally, we show that adding preference information can be easily achieved – this also provides us with powerful tools for Plan Repair.

Figure 3 gives an overview of the relations between STPs, DTPs and their preference and distributed extensions. Rina Dechter's Simple Temporal Problem framework is central in our approach. The generalizations Temporal Constraint Satisfaction Problem and Disjunctive Temporal Problem are expressive enough to encode any temporal plan. Francesca Rossi's group has extended the STP theory with preference information: STPP. We showed in previous work that a distributed variant of STPPs (using Lin Xu's $\triangle STP$ algorithm) can be used for simple plan repair. Enabling more complex repair actions required extra attention for the coordination issues. We followed the approaches of Tsamardinos, Stergiou and Koubarakis, who represented a DTP as a meta-CSP, and Bart Peintner and Martha Pollack, who did the same for its preference extension DTPP. In the literature on Distributed CSPs, we found that Yokoo's Asynchronous Weak-Commitment Search provides a good algorithm for multi-agent DTPP solving. The Clarke-Tax Voting protocol was used to illustrate one possible means for reaching consensus among a group of autonomous planners.

We would like to stress that our main research interest lies in the development of Plan Repair methods. In the near future we will implement the presented algorithms and publish experimental results with this new approach. We will also investigate alternative repair policies besides Weakest Link Optimality, like *local* preference alterations (e.g., [6]).

References

[1] P. Buzing and C. Witteveen. Distributed (re)planning with preference information. In *Proceedings of BNAIC04*, Groningen, The Netherlands, Oct 2004.

[2] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 18:19–33, 1971.

[3] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[4] E. Ephrati and J. S. Rosenschein. The Clarke Tax as a consensus mechanism among automated agents. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 173–178, Anaheim, California, July 1991.

[5] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[6] M. S. Franzin, E. C. Freuder, F. Rossi, and R. Wallace. Multi-agent constraint systems with preferences: Efficiency, solution quality, and privacy loss. *Special issue of Computational Intelligence on "Preferences in AI and CP"*, 20(2), 2004.

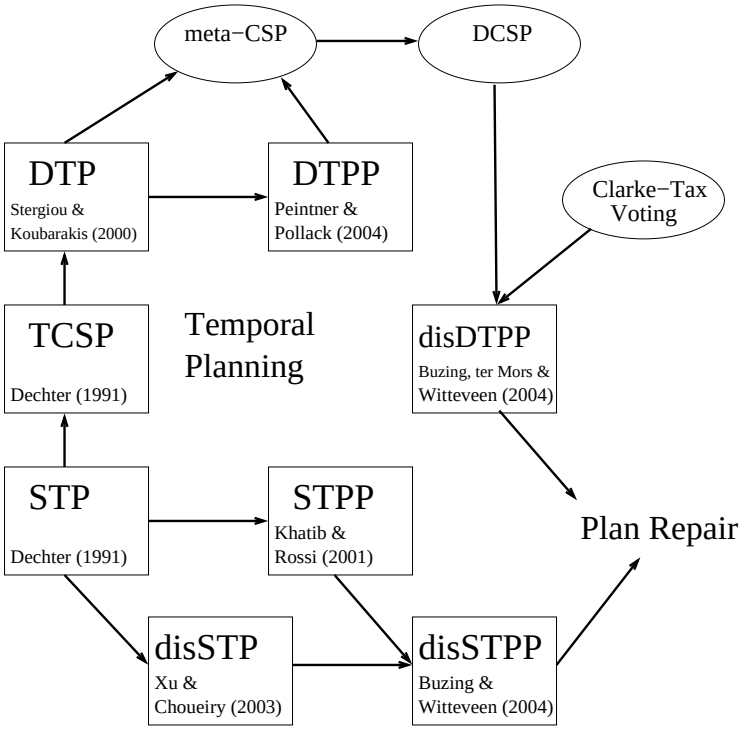


Figure 3: The relations between the different representations and techniques discussed in this paper. Our interest in this framework lies mainly in Plan Repair.

[7] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *17th International Joint Conference on AI*, pages 322–327, 2001.

[8] B. Peintner and M. E. Pollack. Low-cost addition of preferences to dtps and tcsps. In *the 19th National Conference on Artificial Intelligence*, July 2004.

[9] F. Rossi, K. Venable, L. Khatib, P. Morris, and R. Morris. Two solvers for tractable temporal constraints with preferences. In *Proc. AAAI 2002 workshop on preference in AI and CP*, Edmonton, Canada, July 2002.

[10] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[11] I. Tsamardinos and M. E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1-2):43–90, 2003.

[12] L. Xu and B. Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *TIME-ICTL*, pages 212–222, Cairns, Queensland, Australia, 2003. IEEE Computer Society Press.

[13] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.

[14] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.