A Variable Elimination Approach for Optimal Scheduling with Linear Preferences

Nicolas Meuleau[*] and **Robert A. Morris**
NASA Ames Research Center
Moffet Field, California 94035-1000, USA
{Nicolas.F.Meuleau,Robert.A.Morris}@nasa.gov

Neil Yorke-Smith
SRI International
Menlo Park, California 94025, USA
nysmith@ai.sri.com

Abstract

In many practical scheduling problems, feasible solutions can be partially ordered according to differences between the temporal objects in each solution. Often, these orderings can be computed from a compact value function that combines the local preference values of the temporal objects. However, in part because it is natural to view temporal domains as continuous, finding complete, preferred solutions to these problems is a challenging optimization task. Previous works achieve tractability by making restrictions on the model of temporal preferences, including limiting representations to binary and convex preferences. We propose an application of Bucket Elimination (BE) to solve problems with piecewise linear constraints on temporal preferences with continuous domains. The key technical hurdle is developing a tractable elimination function for such constraints. This proof of concept takes a step toward an ability to solve scheduling problems with richer models of preference than previously entertained. Further, it provides a complementary approach to existing techniques for restricted models, because the complexity of BE, while exponential in the treewidth of the problem, is polynomial in its size.

Introduction

Practical problems requiring the assignment of times to planned events are pervasive in the world, from mundane applications like calendar management (Moffitt, Peintner, & Yorke-Smith 2006), to the more exotic such as science observations for planetary exploration (Bresina, Khatib, & McGann 2006). In cases such as these, feasible temporal assignments can be ordered on the basis of the degree to which certain preferences are adhered to. Such preferences can sometimes be expressed as a simple function of times, (e.g., *it's preferable that an event start as early as possible*) but may also express complex relationships (e.g., *it's preferable that a set of events have roughly the same duration*).

There are numerous efforts in representing and solving problems in decision-making with preferences. Practical problems are often challenging because of they involve many preference criteria over continuous temporal variables. Much of the previous work in optimization of temporal preferences has sought to manage the potential complexity of such problems by limiting the expressive power of the language for expressing preferences. A common model, the *Simple Temporal Problem with Preferences* (STPP) (Morris *et al.* 2004) expands the *Simple Temporal Network* (Dechter, Meiri, & Pearl 1991) by allowing numerical preferences on the admissible values of temporal variables. STPPs combine both *hard* constraints that must be satisfied for a solution to be valid, and *soft* constraint that assign a preference value to each admissible assignment. Global preferences are obtained by summing the local preferences of temporal objects. STPPs inherit from STNs being limited to unary and binary constraints. Moreover, all existing efficient solution techniques for STPPs assume convex or semi-convex preferences. Notably, the most significant result in STPPs is the use of Linear Programing (LP) to solve problems with binary piecewise linear convex preferences as described in (Morris *et al.* 2004).

Despite such progress in solving planning and scheduling problems with temporal preferences, restrictions imposed by binary, convex preference functions result in limitations in the expressive power of the language of constraints. Consider a natural temporal preference such as *Maximize the temporal gap (duration) between A and B*. Graphically, this preference could be expressed as a V-shaped non-convex function from times to preference values that reaches 0 (the least preferred value) at time 0.

Removing the restriction on constraints imposed by the STPP paradigm requires an alternative approach to finding solutions. A natural course of action at this point is to consider general approaches to constraint optimization, either based on *conditioning* (backtracking, branch and bound) or Bucket Elimination (BE), which is a generalization of *dynamic programming* (Dechter 1999; 2003). The interest in this paper is in building a temporal solver based on BE for a model of time in which constraints are piecewise linear over continuous domains, but not necessarily binary or convex. The result of this work is the ability to solve scheduling problems with richer models of preference than previously entertained. Further, we provide a complementary approach to existing techniques for restricted models, because the complexity of BE, while exponential in the *treewidth* of the problem, is polynomial in its size. The main contribution of this paper is the proof of the soundness of an approach whose competitiveness will be explored in future works.

[*]Carnegie Mellon University

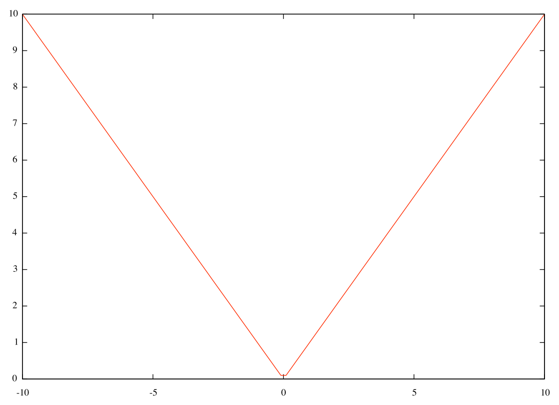

Figure 1: A non-convex function over $-10 \leq t \leq 10$ that expresses the preference: *maximize the temporal difference between two events*.

The next section survey previous work on BE. There follows the definition of the class of continuous temporal problem considered. It is based on *piecewise linear value functions* that encode both hard and soft constraints, and that are not necessarily binary or convex. We then show that BE can be applied to this domain, demonstrating by construction that elimination preserves piecewise linearity. To improve the efficiency of the elimination process, the notion of the 'witness' is applied to the elimination function, leading to our final algorithm. Finally, preliminary experimental results using the full BE algorithm are summarized.

Bucket Elimination

Bucket Elimination (BE) is a complete approach to solving constraint optimization problems that generalizes dynamic programming (Dechter 1999; 2003). Intuitively, BE solves a problem by a series of replacements of variables by constraints that summarize the effect of each variable on the best solution to the problem.

In this approach, hard and soft constraints on an ordered set of variables $X = \{x_1, x_2, \dots x_n\}$ are represented as numerical functions, here called *value functions* V, defined over a subset of X called the *scope* of V. A hard constraint is represented by a value function that takes the value 0 in each state where it is satisfied, and $-\infty$ elsewhere. A soft constraint is represented by a function taking finite values that reflects the preferences over the variables in its scope. A value function can take both finite and infinite values and represent both soft and hard constraints at the same time.

Given a set $\mathcal{V}$ of such preference functions, *bucket* $B(i)$ is defined as the set of constraints having x_i in their scope as their highest variable, according to the ordering of X. For a fixed i, we call $X_i = \{x_{j_1}, x_{j_2}, \dots x_{j_k}, x_i\}$ the smallest subset of variables such that the scope of each utility in $B(i)$ is included in X_i. We then define $X_{-i} = X_i \setminus \{x_i\} = \{x_{j_1}, x_{j_2}, \dots x_{j_k}\}$. Technically, X_{-i} regroups all parents of x_i in the induced constraint graph (Dechter 1999).

BE eliminates variables in reverse order by creating a constraint V_i for each x_i eliminated following:

$$V_i = \text{Elim}_i \left(\sum_{V \in B(i)} V \right)$$

which is defined as:

$$\forall (x_{j_1}, x_{j_2}, \dots x_{j_k}) : V_i(x_{j_1}, x_{j_2}, \dots x_{j_k}) = \max_{x_i} \left[\sum_{V \in B(i)} V(x_{j_1}, x_{j_2}, \dots x_{j_k}, x_i) \right] . \quad (1)$$

V_i is added to the bucket $B(j_k)$, where j_k is the index of the highest variable in X_{-i}. BE terminates when all the variables are eliminated (the resulting function V_0 is then a constant function returning the highest value of any solution to the problem). The optimal state x^* that yielded the highest value can be recovered easily based on this computation.

The complexity of BE strongly contrasts with that of standard *conditioning* algorithm such as backtracking search and branch and bound (Dechter 1999). The worst-case complexity of these algorithms is $O(m^n)$ where n is the number of variables and m is the size of the discrete domain of each variable. Therefore, it is exponential in the problem size (number of variable). In contrast, the complexity of BE is exponential in a parameter called the *treewidth* (or *induced width*) of the problem and, if the treewidth is fixed, polynomial in the problem size. The induced width (Dechter 1999) depends on the ordering of variables in X. Therefore, research as been directed towards finding variable orderings that minimize the treewidth. Unfortunately, it turns out to be an NP-hard problem; (Dow & Korf 2007) is an example of heuristic approach. It is important to note that all previous treewidth minimization techniques can be applied to the continuous framework described in this paper as well as to any other instance of BE.

BE has been used successfully to solve Constraint Optimization Problems (Dechter 2003). The key contribution of this paper is to apply it to scheduling problems with continuous domains.

Problem Formulation

We define a framework where piecewise linear value functions are used to represent both soft and hard constraints over a set of temporal variables $X = \{x_1, x_2, \dots x_n\}$ with continuous domains.

Definition 1 *For any subset of continuous variables $X = \{x_{i_1}, x_{i_2}, \dots x_{i_k}\}$, let $\mathbb{R}[X]$ denote the k-dimensional continuous space $\mathbb{R}^k$ where the axis are labeled with variables in X.*

Definition 2 *Given a subset of continuous variables $X = \{x_{i_1}, x_{i_2}, \dots x_{i_k}\}$, a $(k+1)$-tuple $(\alpha_0, \alpha_{i_1}, \alpha_{i_2}, \dots \alpha_{i_k})$ is used to represent several quantities:*

- *A linear constraint C that is true in all states $x = (x_{i_1}, x_{i_2}, \dots x_{i_k})$ such that $\alpha_0 + \sum_{l=1}^{k} \alpha_{i_l} x_{i_l} \leq 0$. We say that x* satisfies *C iff. C is true in x;*
- *A linear utility function u that associates the reward $u(x) = \alpha_0 + \sum_{l=1}^{k} \alpha_{i_l} x_{i_l}$ to each state $x = (x_{i_1}, x_{i_2}, \dots x_{i_k})$. To represent a hard constraint, the constant α_0 can take the value $-\infty$;*

- *A* linear policy μ *implementing the control law* $x_i \leftarrow \alpha_0 + \sum_{l=1}^{k} \alpha_{i_l} x_{i_l}$ *for a given index* $i \notin (i_1, i_2, \dots i_k)$.

The set X *is called the* scope *of the constraint, utility or policy.*

Definition 3 *A* convex polygon *is defined as a set of linear constraints,* $\mathcal{C} = \{C_i, i\}$. *Each constraint represents a* face *of the polygon (the term* face *is sometimes used in a two dimensional setting, and* singular point *in 1D). The points belonging to the polygon are those that satisfy all constraints in* $\mathcal{C}$. *The scope of a polygon is the union of the scopes of its faces.*

Note this is a loose definition of a polygon that encompasses unbounded pieces.

Definition 4 *A* value function piece *is defined as a pair* $P = (\mathcal{C}, u)$ *where* $\mathcal{C}$ *is a convex polygon and* u *is a linear utility function. To store the optimal policy, some pieces also carry a linear policy* μ. *The scope of a piece is the set of variables appearing in its edges, utility function, or policy.*

Definition 5 *A* piecewise linear value function *with scope* $X = \{x_{i_1}, x_{i_2}, \dots x_{i_k}\}$ *is defined as a collection of value function pieces* $V = \{P_i, i\}$ *such that the polygons of the pieces* P_i *constitute a partition of* $\mathbb{R}[X]$, *and the scope of each piece* P_i *is included in* X.

Definition 6 *A* linear constraint optimization problem *is a pair* $\mathcal{P} = (X, \mathcal{V})$ *where* $X = \{x_1, x_2, \dots x_n\}$ *is a set of continuous temporal variables and* $\mathcal{V}$ *is a set of piecewise linear value functions* $V = \{P_i, i\}$ *with scope included in* X. *A solution of* $\mathcal{P}$ *is a state* $x = (x_1 \dots, x_n) \in \mathbb{R}[X]$. *A solution* x^* *is optimal if it maximizes* $\sum_{V \in \mathcal{V}} V(x)$.

Solving Linear Constraint Optimization Problems using Bucket Elimination

Formally, BE is based on two operators over functions: summing and variable elimination (Elim_i). The soundness of BE for piecewise linear value functions is formally derived from Theorem 1, which we establish from two Lemmas.

Lemma 1 *The sum of a finite number of piecewise linear value functions is a piecewise linear value function.*

Proof: The sum of two value functions is computed by enumerating all non-empty intersections of two of their pieces, and summing the utility functions carried by these pieces. Because the intersection of two convex polygons is a convex polygon, and the sum of two linear utility functions is a linear utility function, it results in a well defined piecewise linear value function. □

Lemma 2 *If* Q *is a piecewise linear value function with scope* $X = \{x_{j_1}, x_{j_2}, \dots x_{j_k}, x_i\}$, *then* $\mathrm{Elim}_i(Q)$ *is a piecewise linear value function with scope* $X_{-i} = \{x_{j_1}, x_{j_2}, \dots x_{j_k}\}$.

Proof: To prove this lemma, we show how $\mathrm{Elim}_i(Q)$ can be computed and make clear that, by construction, the resulting function is a piecewise linear value function.

The first operation in the calculation of $\mathrm{Elim}_i(Q)$ is to *project* the partition of $\mathbb{R}[X]$ defined by Q on $\mathbb{R}[X_{-i}]$ (or, loosely speaking, to project Q on X_{-i}). It consists of computing a partition of $\mathbb{R}[X_{-i}]$ in convex polygons, as shown in Fig. 2 for $X = \{X_0, X_1\}$. This is achieved by first of projecting the pieces of Q on $\mathbb{R}[X_{-i}]$, which creates only convex polygonal pieces, and then computing all possible intersections of projected pieces. Therefore, the partition computed by projection contains only convex polygonal pieces.

Consider now a particular projected piece P^0 and call $\mathcal{P}_Q^0$ the set of pieces of Q that contributes to P^0 (intersecting the horizontal stripe associated with P_0 in Fig. 2). If Δ^0 as any translation of the x_i-axis that projects inside of P^0, then Δ^0 enters P through a well defined face C_P^- and exits through another face C_P^+ (see Fig. 2). Because utility is linear over P, maximum utility is attained on a face C_P^* that is either C_P^- or C_P^+. This allows deriving an analytical expression of the optimal decision rule for x_i. If the optimal of piece P is attained on the face represented by $C_P^* : \alpha_0^C + \sum_{l=1}^{k} \alpha_{j_l}^C x_{j_l} + \alpha_i^C x_i \leq 0$ ($C_P^* = C_P^-$ or C_P^+), then the optimal decision rule (policy) for piece P is μ_P:

$$x_i \leftarrow -\frac{\alpha_0^C}{\alpha_i^C} - \sum_{l=1}^{k} \frac{\alpha_{j_l}^C}{\alpha_i^C} x_{j_l} \ . \tag{2}$$

This rule guarantees that $\alpha_0^C + \sum_{l=1}^{k} \alpha_{j_l}^C x_{j_l} + \alpha_i^C x_i = 0$, and thus that we are choosing a point on the face represented by C_P^*. As long as parent variables $(x_{j_1}, x_{j_2}, \dots x_{j_k})$ are in P^0, the optimal value of Q over piece P is attained by μ_P, which is a linear policy with scope X_{-i}.

Now that the optimal policy for piece P is known, it is easy to derive an analytical expression of the optimal value of Q over P. If u_P is the linear utility associated with piece P and μ_P is the optimal policy in P, then the optimal value is the return of u_P when x_i is chosen following μ_P. In other words, if $u_P(x) = \alpha_0^u + \sum_{l=1}^{k} \alpha_{j_l}^u x_{j_l} + \alpha_i^u x_i$ and $\mu_P : x_i \leftarrow \alpha_0^\mu + \sum_{l=1}^{k} \alpha_{j_l}^\mu x_{j_l} + \alpha_i^\mu x_i$, then an analytical expression of V_p is obtained by substituting x_i for μ_P in u_P:

$$V_P(x) = \alpha_0^u + \alpha_i^u \cdot \alpha_0^\mu + \sum_{l=1}^{k} \left(\alpha_{j_l}^u + \alpha_i^u \cdot \alpha_{j_l}^\mu \right) x_{j_l} \ . \tag{3}$$

The resulting utility is linear with scope X_{-i}.

To compute $\mathrm{Elim}_i(Q)$, we need to maximize Q with respect to x_i. By definition:

$$\begin{aligned} &\forall (x_{j_1}^0, x_{j_2}^0, \dots x_{j_k}^0) \in P^0 : \\ &\quad \max_{x_i} \left[Q(x_{j_1}^0, x_{j_2}^0, \dots x_{j_k}^0, x_i) \right] = \\ &\qquad\qquad \max_{P \in \mathcal{P}_Q^0} \left[V_p(x_{j_1}^0, x_{j_2}^0, \dots x_{j_k}^0) \right] \ . \end{aligned}$$

However, the piece P that maximizes Q may vary over P^0. Therefore, P^0 may be split in at most as many pieces as in $\mathcal{P}_Q^0$. For each $P \in \mathcal{P}_Q^0$, we build a convex polygon $\mathcal{C}_P$ by first copying all faces of P^0 and then adding to this set the following constraints:

$$\forall P' \in \mathcal{P}_Q^0, P' \neq P : u_{P'}(x) - u_P(x) \leq 0 \ , \tag{4}$$

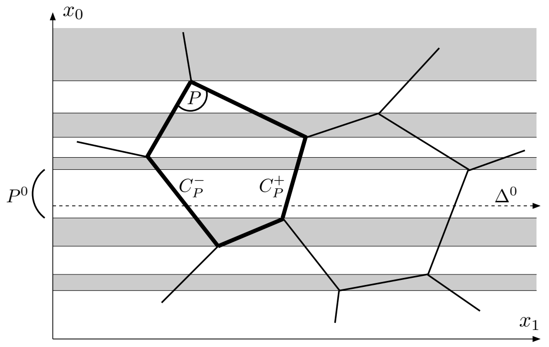

Figure 2: Projecting of Q on $X_{-i} = \{x_0\}$: The convex polygonal partition of the plane represents the pieces of Q. Horizontal stripes represents the partition of $\mathbb{R}[X_{-i}]$ obtained by projecting Q.

where u_P and $u_{P'}$ are the linear utility functions carried by P and P'. These constraints are well defined linear constraints with scope X_{-i}. They ensure that the optimal of Q over P^0 is carried by piece P. If the resulting set $\mathcal{C}_P$ is consistent, then $\mathcal{C}_P$ is a well defined polygon and the triple $(\mathcal{C}_P, V_P, \mu_P)$ is a well defined value function piece of $\mathrm{Elim}_i(Q)$. Because $\mathrm{Elim}_i(Q)$ is made only of such pieces, it is a well defined piecewise linear value function. □

Theorem 1 *If bucket $B(i)$ contains only piecewise linear value functions, then the function V_i resulting from an application of Eqn. 1 is also a piecewise linear value function.*

Proof: The claim results directly from Lemma 1 and 2. □

Bucket Elimination's fundamental equation (Eqn. 1) starts with a universal quantification over $(x_{j_1}, x_{j_2}, \dots x_{j_k})$. In the linear framework defined in the previous section, these variables have infinite continuous domains. Therefore, applying Eqn. 1 requires a continuous infinity of maximizations. Fortunately, Theorem 1 shows that the computation can be performed only once for each piece of V_i, and there is a finite number of such pieces.

A direct application of Th. 1 solves

$$\forall P \in V_i, \forall x = (x_{j_1}, x_{j_2}, \dots x_{j_k}) \in P :$$
$$V_i(x) = \max_{x_i} \left[\sum_{V \in B(i)} V(x_{j_1}, x_{j_2}, \dots x_{j_k}, x_i) \right] , \tag{5}$$

this computation being performed *analytically*. That is, an analytical expression of V_i (and the associated optimal policy) that applies to the whole piece P is computed instead of a single numerical value. In this way, Eqn. 1 is implicitly solved over the infinite set of values.

Algorithm 1 is a straightforward procedure computing Eqn. 5. It follows closely the proof of Lemma 1 and 2. It is important to note that, for a fixed treewidth, the complexity of this procedure is polynomial in the size of the problem, which is a vector four variables: (i) The number of temporal variables n; (ii) The number of value functions $V \in B(i)$; (iii) the maximum number of pieces in a function $V \in B(i)$; (iv) the maximum number of constraints in a piece of a function $V \in B(i)$. The algorithm is exponential in the treewidth because it has to build a piecewise linear value function V_i with dimension as high as the treewidth.

Algorithm 1 A straightforward implementation of Th. 1

```
 1: Compute an analytical representation of $Q = \sum_{V \in B(i)} V$.
 2: Project $Q$ on $X_{-i}$.
 3: for each projected piece $P^0$ do
 4:   for each piece $P$ of $Q$ that contributes to $P^0$ ($P \in \mathcal{P}_Q^0$) do
 5:     Compute the optimal return $V_P$ of $P$ following Eqn. 3.
 6:     Compute the constraint set $\mathcal{C}_P$ as in proof of Lemma 2.
 7:     if $\mathcal{C}_P$ is consistent then
 8:       Add the piece $(\mathcal{C}_P, V_P)$ to $V_i$.
 9:     end if
10:   end for
11: end for
```

While being conceptually simple, Alg. 1 presents two bottlenecks: (i) computing the multidimensional sum Q; (ii) projecting Q_i on X_{-i}. Although it is guaranteed to be polynomial in the problem size as long as the treewidth is bounded, these operations become expensive when the treewidth increases. This is because the size of the largest function V_i that the algorithm has to build is equal to the induced width of the problem. For instance, calculating Q requires computing all non-empty intersections of pieces of functions in $B(i)$. Computing non-empty intersections is done by considering all possible combinations of pieces, putting all the faces of these pieces in a single set, and testing the consistency of this set. If the set is consistent, then the pieces have a non-empty intersection. In high dimensions, this consistency test is expensive, making the summation an overly expensive process. In the next section, we present an algorithm that avoids these two pitfalls.

A Witness Algorithm

Algorithm 2 is a *witness* algorithm that efficiently implements Th. 1.[1] This algorithm associates with every piece P of V_i a witness state-vector $x^P = (x_{j_1}^P, x_{j_2}^P, \dots x_{j_k}^P)$ that testify of the existence of P in V_i. It then performs the finite calculation

$$\forall P \in V_i : V_i(x_{j_1}^P, x_{j_2}^P, \dots x_{j_k}^P) =$$
$$\max_{x_i} \left[\sum_{V \in B(i)} V(x_{j_1}^P, x_{j_2}^P, \dots x_{j_k}^P, x_i) \right] . \tag{6}$$

The result of this local computation is generalized to produce analytical expressions of the optimal utility and policy over P. The strength of this algorithm is that once $(x_{j_1}, x_{j_2}, \dots x_{j_k})$ has been substituted by $(x_{j_1}^P, x_{j_2}^P, \dots x_{j_k}^P)$, then all functions in $B(i)$ are piecewise linear functions of the single variable x_i, that is $V(x_i) = \alpha + \alpha_i x_i$. Therefore, summing them and maximizing them is extremely easy.

[1] We borrow the term *witness* from the POMDP literature. Littman's *witness algorithm* uses witness points to testify for the presence of an "α-vectors" in the optimal solution of a POMDP (Kaelbling, Littman, & Cassandra 1998).

Algorithm 2 A witness algorithm

```
1: while $V_i$ is not totally defined do
2:     Pick a state $x^0 = (x^0_{j_1}, x^0_{j_2}, \dots x^0_{j_k}) \in \mathbb{R}[X_{-i}]$ where $V_i$ is not defined yet.
3:     Determine the characteristics of the piece of $V_i$ containing $x^0$ (utility $u$ and policy $\mu$).
4:     Build a set of constraints ensuring that these characteristics remain. This constraint set represent the faces of the piece of $V_i$ containing $x^0$.
5:     Simplify the constraint set built at previous step.
6:     Add the new piece to $V_i$.
7: end while
```

This contrast strongly with Alg. 1 that endures very high cost manipulating multidimensional functions.

Selecting an unexplored state $x^0 \in \mathbb{R}[X_{-i}]$ (line 2 of Alg. 2): This step is a bottleneck where we face the multidimensional space avoided otherwise. It can be performed systematically by a costly and complex recursive procedure. Random and pseudo-random techniques might be more efficient despite having no guarantee of completeness. Our prototype implementation employs a simple but inefficient technique that checks only the points on a discrete grid covering the set of possible values for the variables in X_{-i}.

In the following we call P^0_V the piece of V_i containing x^0. x^0 is the witness of P^0_V. The next steps aim at building P^0_V .

Determining the characteristics (μ and u) of P^0_V (line 3 of Alg. 2)): Our algorithm performs these operations in a particular way that minimizes computation time.

Given $x^0 = (x^0_{j_1}, x^0_{j_2}, \dots x^0_{j_k})$, the first operation performed is to substitute $(x_{j_1}, x_{j_2}, \dots x_{j_k})$ for x^0 in every function $V \in B(i)$. The resulting function called $V[x^0]$ is a piecewise linear function of the single variable x_i: $V[x^0](x_i) = \alpha_0 + \alpha_i x_i$. We later call this operation *instantiating* V as x^0.

Next all $V[x^0]$ are summed to produce the function $Q[x^0]$ that is the instantiation of $Q = \sum_{V \in B(i)} V$ in x^0. In Fig. 2, $Q[x^0]$ represents the variation of Q along the Δ^0 axis that passes through x^0. Again, these operations are straightforward and the result is a linear function of x_i only.

Finally we determine the maximum of $Q[x^0]$ w.r.t. x_i. This is a straightforward optimization of a 1-dimensional piecewise linear function. We then have solved BE fundamental equation in the particular point x^0 with minimum effort. We obtain an analytical expression of the optimal utility u and the optimal policy μ in P^0_V by generalizing this local computation as explained below.

Because the $V[x^0]$'s and $Q[x^0]$ are unidimensional functions, their faces are *singular points* between two adjacent pieces. One can show that each singular point of $V[x^0]$'s and $Q[x^0]$ is inherited from an face of some function $V \in B(i)$. First, $V[x^0]$ is obtained by traversing the piecewise linear value function V with the axis Δ^0 defined by $(x_{j_1}, x_{j_2}, \dots x_{j_k}) = x^0$, as in Fig. 1. The singular point of $V[x^0]$ are the points where Δ^0 intersects an face of V. Therefore each singular point is inherited from a constraint C of V. Second, $Q[x^0]$ is obtained by intersecting and summing the unidimensional piecewise linear functions $V[x^0]$. Therefore, each of its singular point is inherited from a singular point of some function $V[x^0]$, and so it is inherited from a constraint C of V. In order to determine the analytical solution, we must keep track of the relationships between singular points of $Q[x^0]$ and constraints of V's; fortunately, maintaining this information is relatively inexpensive.

Since $Q[x^0]$ is piecewise linear, its optimum is attained in one of its singular points. The constraint at the origin of this singular point determines the optimal policy over P^0_V. If the constraint is $C : \alpha^C_0 + \sum_{l=1}^k \alpha^C_{j_l} x_{j_l} + \alpha^C_i x_i \leq 0$ then the optimal policy of V^0_P is given by Eqn. 2. This is a well defined linear policy with scope in X_{-i}.

Now that we know the policy μ attached to V^0_P, we want to determine an analytical expression of the utility u attached to this piece of V_i. We cannot carry on the same computation as in the proof of Lemma 2 because we do not have an explicit representation of Q. However, again, keeping track of simple information achieves the same result. Indeed we only need to memorize the piece of each V that contributes to each piece of $Q[x^0]$, which is straightforward and very cheap. Then, if we denote P^* as the piece of $Q[x^0]$ containing the optimal value, we can trace back the piece P_V of each $V \in B(i)$ that contributes to P^*. By summing the utility functions carried by these pieces, we get an analytical expression of the utility of the piece of Q containing P^*. As in the proof of Lemma 2, we know the utility and the optimal policy associated with the optimal piece of Q. Therefore, we can get an analytical expression of the utility attached to V^0_P by substituting x_i by μ in u following Eqn. 3.

The preceding discussion shows that there is no need to employ a complex multidimensional representation of Q to determine the optimal policy and utility associated with V^0_P. In this calculation, the most costly step is the instantiation of each function V in x^0. We show below how this operation can be performed efficiently.

Instantiating $V \in B(i)$ in $x^0 = (x^0_{j_1}, x^0_{j_2}, \dots x^0_{j_k})$ (used at line 3 of Alg. 2): The procedure used for instantiating a function V in x^0 loops through all pieces of V and, for each piece $P = (\mathcal{C}, u)$, checks whether the Δ^0-axis passing through x^0 traverses P or not. This check is performed by substituting $(x_{j_1}, x_{j_2}, \dots x_{j_k})$ by x^0 in every constraint $C \in \mathcal{C}$. It transforms a constraint $C : \alpha_0 + \sum_{l=1}^k \alpha_{i_l} x_{i_l} \leq 0$ into unidimensional constraint $C[x^0]$ that is is either

$$x_i \leq -\frac{\alpha_0}{\alpha_i} - \sum_{l=1}^{k} \frac{\alpha_{j_l}}{\alpha_i} x^0_{j_l} \qquad \text{if } \alpha_{\mathrm{i}} > 0 \ ,$$

or

$$x_i \geq -\frac{\alpha_0}{\alpha_i} - \sum_{l=1}^{k} \frac{\alpha_{j_l}}{\alpha_i} x^0_{j_l} \qquad \text{if } \alpha_{\mathrm{i}} < 0 \ .$$

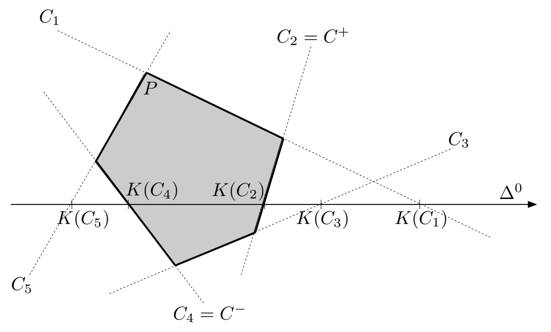

Figure 3: Instantiating V: Δ^0 (the axis defined by $(x_{j_1}, x_{j_2}, \dots x_{j_k}) = x^0$) enters P through face $C^- = C_4$ and leaves through face $C^+ = C_2$. For each constraint C, $K(C)$ is the coordinate (on Δ^0) of the point where Δ^0 traverses face C.

We call $\mathcal{C}^+$ the set of constraints of the first type and $\mathcal{C}^-$ the set of constraints of the second type. Then we define

$$K(C) = -\frac{\alpha_0}{\alpha_i} - \sum_{l=1}^{k} \frac{\alpha_{j_l}}{\alpha_i} x^0_{j_l}$$

as the coordinate (on Δ^0) of the point where Δ^0 traverses face C (see Fig. 3), and $C^+ = \arg\min_{C\in\mathcal{C}^+}[K(C)]$ and $C^- = \arg\max_{C\in\mathcal{C}^-}[K(C)]$. Then Δ^0 traverses $\mathcal{P}$ iff. $K(C^-) < K(C^+)$. If Δ^0 traverses $\mathcal{P}$ then the segment of Δ^0 contained into P defines a piece $[K(C^-), K(C^+)]$ of $V[x^0]$. The constraint associated with singular point $K(C^-)$ is C^- and the constraint associated with $K(C^+)$ C^+. The detail of this computation is presented in Alg. 3.

Computing the faces of the new piece (line 4 of Alg. 2): In the proof of Lemma 2, the faces of a new piece are computed by adding to the faces of P^0 (see Fig. 2) the constraints defined by Eqn. 4. We follow the same general principle to derive the set of faces of P^0_V.

First, because we do not have an explicit representation of Q, we do not have an easy access to the faces of P^0. Fortunately, an equivalent constraint set can be recovered using a simple set of pointers. The constraints of a piece P^0 of the projection of Q on X_{-i} express the fact that the axis Δ^0 passing through $x = (x_{j_1}, x_{j_2}, \dots x_{j_k})$ traverses the same sequence of pieces of Q for all x in P^0 (cf. Fig. 2). A necessary condition for this holding is that Δ^0 traverses the same sequence of pieces of each $V \in B(i)$. Given a function $V \in B(i)$, we can build a set of constraints implying that Δ^0 traverses a fixed sequence of pieces of V in two steps:

- For each piece P of V that is traversed by Δ^0, add a constraint implying that P is still traversed by Δ^0;
- For each piece P of V that is *not* traversed by Δ^0, add a constraint implying that P is still *not* traversed by Δ^0.

This procedure is linear in the number of pieces in V. Since it is performed once for each piece P^0, the algorithm is quadratic in the maximum number of pieces in a value function. Therefore, we adopt the cheaper alternative that builds a single set of constraints:

Algorithm 3 Instantiating V in $x^0 = \left(x^0_{j_1}, x^0_{j_2}, \dots x^0_{j_k}\right)$.

```
1: for each piece $\mathcal{P} = (\mathcal{C}, u)$ of $V$ do
2:   $\mathcal{C}^- \leftarrow \emptyset, \mathcal{C}^+ \leftarrow \emptyset$.
3:   for each face $C = (\alpha_0, \alpha_{j_1}, \alpha_{j_2}, \dots \alpha_{j_k}, \alpha_i)$ of $\mathcal{C}$ do
4:     if $x_i$ is in the scope of $C$ ($\alpha_i \neq 0$) then
5:       if $\alpha_i > 0$ then
6:         $\mathcal{C}^+ \leftarrow \mathcal{C}^+ \cup \{C\}$.
7:       else
8:         $\mathcal{C}^- \leftarrow \mathcal{C}^- \cup \{C\}$.
9:       end if
10:      Compute $K(C) = -\frac{\alpha_0}{\alpha_i} - \sum_{l=1}^k \frac{\alpha_{j_k}}{\alpha_i} x^0_{j_k}$ .
11:    else
12:      // $x_i$ is not in the scope of $C$ ($\alpha_i = 0$)
13:      Substitute $(x_{j_1}, x_{j_2}, \dots x_{j_k})$ for $x^0$ in $C$ and check for
         consistency: $\alpha_0 + \sum_{l=1}^k \alpha_{j_l} x^0_{j_l} \leq 0$ ?
14:      if Consistent then
15:        Skip to next face of $\mathcal{C}$ (go to line 3).
16:      else
17:        Skip to next piece of $V$ (go to line 1).
18:      end if
19:    end if
20:  end for
21:  Compute $C^- = \arg\max_{C\in\mathcal{C}^-}[K(C)]$ and $C^+ =$
     $\arg\min_{C\in\mathcal{C}^+}[K(C)]$.
22:  if $K(C^+) <= K(C^-)$ then
23:    // Inconsistency detected
24:    Skip to next piece of $V$ (go to line 1).
25:  else
26:    Substitute $(x_{j_1}, x_{j_2}, \dots x_{j_k})$ for $x^0$ in $u$ to get a function
       $u[x^0](x_i)$.
27:    Add the piece $\left([K(C^-), K(C^+)], u[x^0]\right)$ to $V[x^0]$. At-
       tach to singular point $K(C^-)$ the constraint $C^-$ and at-
       tach $C^+$ to $K(C^+)$.
28:  end if
29:  // Continue to next piece of $V$
30: end for
31: Return $V[x^0]$
```

- For each piece P of V that is traversed by Δ^0, add a constraint implying that Δ^0 still enters and exits P through the same faces.

This creates an equivalent set of constraints, but the complexity is linear in the number of pieces traversed by Δ^0 which is generally much smaller than the total number of pieces. Now, given a piece P of V, we ensure Δ^0 enters and exit through the same face of P using two types of constraint:

$$\forall C \in \mathcal{C}^-, C \neq C^- : K(C) \leq K(C^-) \ ,$$

and

$$\forall C \in \mathcal{C}^+, C \neq C^+ : K(C^+) \leq K(C) \ .$$

We add a following constraint to ensure Δ^0 traverses P:

$$K(C^-) \leq K(C^+) \ .$$

Formally, if $C^- : \alpha^-_0 + \sum_{l=1}^k \alpha^-_{i_l} x_{i_l} \leq 0$, and $C^+ : \alpha^+_0 + \sum_{l=1}^k \alpha^+_{i_l} x_{i_l} \leq 0$, and $C = \alpha_0 + \sum_{l=1}^k \alpha_{i_l} x_{i_l} \leq 0$, then these constraints constraints are computed as (respectively):

$$\frac{\alpha^-_0}{\alpha^-_i} - \frac{\alpha_0}{\alpha_i} + \sum_{l=1}^{k} \left(\frac{\alpha^-_{j_l}}{\alpha^-_i} - \frac{\alpha_{j_l}}{\alpha_i} \right) x_{j_l} \leq 0 \ , \qquad (7)$$

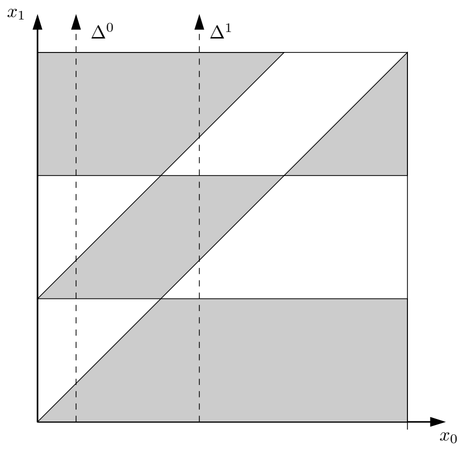

Figure 4: The partition of $\mathbb{R}^2$ defined by the pieces of Q. In this example, there are two value functions in B_i: A reward function $R_1(x_1)$ at the origin of the horizontal faces in the figure, and a function $R_{01}(x_1 - x_0)$ that represents piecewise linear preferences on the difference between x_0 and x_1 and that induces the diagonal constraints. This is a standard situation encountered in many STPPs. The axis Δ^0 and Δ^1 traverse the same sequence of pieces of both R_1 and R_{01}. However, they traverse different pieces of Q.

$$\frac{\alpha_0}{\alpha_i} - \frac{\alpha_0^+}{\alpha_i^+} + \sum_{l=1}^{k} \left(\frac{\alpha_{j_l}}{\alpha_i} - \frac{\alpha_{j_l}^+}{\alpha_i^+} \right) x_{j_l} \leq 0 \ , \qquad (8)$$

$$\frac{\alpha_0^+}{\alpha_i^+} - \frac{\alpha_0^-}{\alpha_i^-} + \sum_{l=1}^{k} \left(\frac{\alpha_{j_l}^+}{\alpha_i^+} - \frac{\alpha_{j_l}^-}{\alpha_i^-} \right) x_{j_l} \leq 0 \ . \qquad (9)$$

Equations 7 to 9 allow for the building of a constraint set that guarantees that Δ^0 traverses the a fixed sequence of pieces of each $V \in B(i)$. This condition is necessary to ensure that we stay inside of a single piece P^0, but it is not sufficient. A counter-example is provided in Fig. 4. Fortunately, this issue can be addressed by adding a small number of simple constraints. In the example of Fig. 4, Δ^0 and Δ^1 traverse the same sequence of pieces of each $V \in B(i)$, but they traverse different sequences of pieces *all functions taken together*. Both axis start in the first piece of R_1 and the first piece of R_{01}, Δ^0 enters the second piece of R_1 before entering the second piece of R_{01}, while Δ^1 does the opposite. Therefore, we get a complete set of constraints to characterize the faces of P^0 in the following way: For any two adjacent pieces $[a, b]$ and $[b, c]$ of $Q[x^0]$, such that C_a is the constraint associated to singular point a during the construction of $Q[x^0]$ and C_b is the constraint associated with b, we add the constraint:

$$K(C_a) < K(C_b) \ .$$

If $C_a : \alpha_0^a + \sum_{l=1}^{k} \alpha_{i_l}^a x_{i_l} \leq 0$ and $C_b : \alpha_0^b + \sum_{l=1}^{k} \alpha_{i_l}^b x_{i_l} \leq 0$, then it translates as:

$$\frac{\alpha_0^b}{\alpha_i^b} - \frac{\alpha_0^a}{\alpha_i^a} + \sum_{l=1}^{k} \left(\frac{\alpha_{j_l}^b}{\alpha_i^b} - \frac{\alpha_{j_l}^a}{\alpha_i^a} \right) x_{j_l} \leq 0 \ . \qquad (10)$$

The constraints defined by Eqn. 7 to 10 are the faces defining a unique piece P^0 of the projection of Q on X_{-i}. In the proof of Lemma 2, this piece is split in as many sub-pieces as there are pieces P of Q that satisfy Eqn. 4 for some x. Here, we are not interested in all these pieces but only the one containing the current witness x^0. Thus, we complete the set of faces of P_V^0 by adding the constraints defined by Eqn. 4 for the particular piece P that carries the optimum of of $Q[x^0]$ as computed at step 1. If $u_P(x) = \alpha_0 + \sum_{j=1}^{k} \alpha_{j_l} x_{j_l}$ and $u_{P'}(x) = \alpha'_0 + \sum_{j=1}^{k} \alpha'_{j_l} x_{j_l}$, Eqn. 4 becomes

$$\alpha'_0 - \alpha_0 + \sum_{l=1}^{k} \left(\alpha'_{j_l} - \alpha_{j_l} \right) x_{j_l} \leq 0 \ . \qquad (11)$$

Equations 7 to 11 generate the set of all faces of P_V^0.

Simplifying Constraint Sets (line 5 of Alg. 2): The constraint set built by the procedure described above may contain redundant constraints, or constraints that are useless because they are *dominated* by other constraints. For instance, if $C : x_0 \leq 0$ is in the constraint set, then any constraint $C' : x_0 \leq K$ for some $K > 0$ can be omitted in the representation of the piece. Indeed, our experiments show that the witness algorithm produces many such constraints.

The algorithm complexity is sensitive to the number of constraints in the description of each piece. Although the maximum number of constraints in a piece is guaranteed to be polynomial in the problem size, it is preferable to keep this number as low as possible. Therefore, in line 5 of Alg. 2 we try to purge the set from useless constraint.

Simplifying a set $\mathcal{C}$ of linear constraints with scope X is a difficult problem. At this point we are considering several approaches to address it:

• A similar problem arises in the optimization of POMDPs where we must determine a minimal set of linear faces to represent a piecewise linear convex function. Here the problem is addressed by solving multiple LPs. This approach seems relatively easily transferable to our sub-problem.

• Solving the dual program of an LP M allows determining the constraints that are binding in the optimal solution of M (Schrijver 1986). Defining $M = (X, \mathcal{V}, 0)$ where 0 is the objective function that returns 0 in all states, then any point inside of the polygon defined by $\mathcal{C}$ is an optimal solution of M. Solving the dual problem of M can thus compute a minimum set of faces.

• Our current implementation of the witness algorithm just prunes the set $\mathcal{C}$ from unary constraints that are dominated by other unary constraints. Useless k-ary constraints, $k > 1$ are kept in the description of the piece, which can hurt the algorithm. In general, there is a trade-off between the time spent pruning the set of constraints, and the impact of useless constraints on complexity.

Note that the problem of simplifying $\mathcal{C}$ is exponential in the number of variables in its scope X. However, in our case, the scope of the functions is bounded by the treewidth of the problem. Therefore, with a bounded treewidth, complexity is polynomial in the size of the problem.

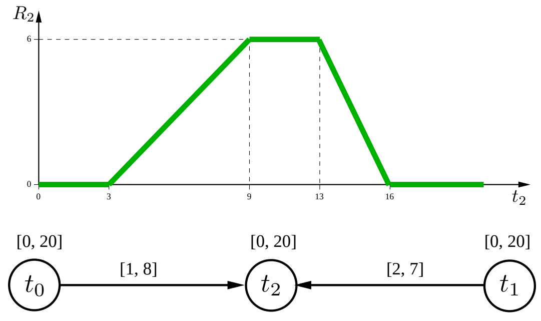

Figure 5: A simple STPP with non-convex preferences. Interval $[l_i, u_i]$ attached to vertex t_i represents the hard constraint $l_i \leq t_i \leq u_i$. Interval $[l_{ij}, u_{ij}]$ on edge $t_i \rightarrow t_j$ represents the hard constraint $l_{ij} \leq t_j - t_i \leq u_{ij}$. There is a total of 4 value functions in $B(2)$: one for the hard constraint on t_2, two for the hard constraints on $t_2 - t_0$ and $t_2 - t_1$, and the reward $R2(t_2)$ represented at the top.

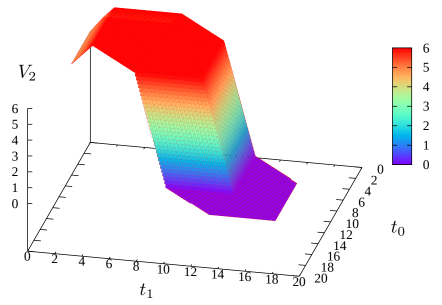

Figure 6: The 3D value-function V_2 obtained by eliminating t_2 in the problem of Fig. 5. Points not assigned a value in this graph are those that break a hard constraint (hence they have $-\infty$ value).

Summary and Future Work

This paper provides proof of concept that bucket elimination can be used to optimize a rich model of temporal preferences. The potential of this concept for scheduling with preferences is twofold. First, a richer preference model opens up generation of schedules that better reflect the desires of the modeler; second, the alternate position of BE in the space–time trade-off opens up a complementary approach compared to prior art.

After showing the relevance and soundness of the approach, we provided a grounded algorithm. A prototype implementation is capable of solving a richer class of problems than previous approaches. It has been run successfully on problems involving non convex and k-ary constraints. Fig. 6 shows the result of one iteration of the witness algorithm on the STPP of Fig. 5.

At this point, our implementation of the witness algorithm suffers from two weaknesses: (i) the simplistic grid-base procedure used to find a new witness x^0 where V_i is not yet defined; (ii) the procedure used to simplify constraint sets cannot handle more than unary constraints.

Our ongoing work is to improve the efficiency of the prototype implementation, and to compare the BE approach both theoretically and empirically to alternate approaches, such as the use of repeated Linear Programming with a branch-and-bound search.

In particular, because binary and convex STPPs is a sub-class of problems that can be represented as linear constraint optimization problems, it would be interesting to compare our algorithm with the results in (Morris *et al.* 2004). BE and LP approaches can both solve this sub-class of problems, but there complexity bears on different parameters. Our future work will aim at showing experimentally that BE can be a competitive approach for large problems with small treewidth.

Acknowledgments. The work of the author from SRI International was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA, or the Air Force Research Laboratory (AFRL).

References

Bresina, J. L.; Khatib, L.; and McGann, C. 2006. Mission operations planning with preferences: An empirical study. *International Workshop on Planning and Scheduling for Space.*

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artifi cial Intelligence* 49:61–95.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artifi cial Intelligence* 113(1-2):41–85.

Dechter, R. 2003. *Constraint Processing*. San Francisco, CA: Morgan Kaufmann.

Dow, P., and Korf, R. 2007. Best-first search for treewidth. In *Proceedings of the Twenty Second National Conference on Artifi - cial Intelligence*, 1146–1151. Menlo Park, CA: AAAI Press.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artifi cial Intelligence* 101:99–134.

Moffitt, M. D.; Peintner, B.; and Yorke-Smith, N. 2006. Multi-criteria optimization of temporal preferences. In *Proc. of CP'06 Workshop on Preferences and Soft Constraints (Soft'06)*, 79–93.

Morris, P.; Morris, R.; Khatib, L.; Ramakrishnan, S.; and Bachmann, A. 2004. Strategies for global optimization of temporal preferences. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, 408–422. Berlin, Germany: Springer-Verlag.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. New York, NY: John Wiley and sons.