Learning and Solving Soft Temporal Constraints: An Experimental Study

F. Rossi, A. Sperduti, K.B. Venable
Department of Pure and Applied Mathematics,
University of Padova, Italy
{frossi,perso,kvenable}@math.unipd.it

L. Khatib[(*)], P. Morris, R. Morris
(*) Kestrel Technology
NASA Ames Research Center, Moffett Field, CA, USA
{lina,pmorris,morris}@ptolemy.arc.nasa.gov

Abstract

Soft temporal constraints problems allow for a natural description of scenarios where events happen over time and preferences are associated with event distances and durations. However, sometimes such *local* preferences are difficult to set, and it may be easier instead to associate preferences to some complete solutions of the problem, and then to *learn* from them suitable preferences over distances and durations.

In this paper, we describe our learning algorithm and we show its behaviour on classes of randomly generated problems. Moreover, we also describe two solvers (one more general and the other one more efficient) for tractable subclasses of soft temporal problems, and we give experimental results to compare them.

1 Introduction and motivation

Several real world problems involve the manipulation of temporal information in order to find an assignment of times to a set of activities or events. These problems usually have preferences associated with how long a single activity should last, when it should occur, or how it should be ordered with respect to other activities. For example, an antenna on an earth orbiting satellite such as Landsat 7 must be slewed so that it is pointing at a ground station in order for recorded science or telemetry data to be downlinked to earth. Antenna slewing on Landsat 7 has been shown to occasionally cause a slight vibration to the satellite, which in turn might affect the quality of the image taken by the scanning instrument if the scanner is in use during slewing. Consequently, it is preferable for the slewing activity not to overlap any scanning activity, although because the detrimental effect on image quality occurs only intermittently, this disjointness is best not expressed as a hard constraint. This is only one of the many real world problems that can be cast and, under certain assumptions, solved in our framework.

Reasoning simultaneously with hard temporal constraints and preferences, as illustrated in the example just given, is crucial in many situations. However, in many temporal reasoning problems it is difficult or impossible to specify some local preferences on durations. In real world scheduling problems, for example, it is sometimes easier to see how preferrable a solution is, but it may be virtually impossible to specify how specific ordering choices between pairs of events contribute to such global preference value.

This scenario is typical in many cases. For example, it occurs when there is no precise function which describes the assignment of a preference value to a global solution. This may happen for example when we just have an expert, whose knowledge is difficult to code as local preferences, but who can immediately recognize a good (or bad) global solution. Another typical case occurs when the environment in which the solver will work presents some level of uncertainty. In this case, we could have the local preferences, but their effect on a global solution could depend on events which are not modeled within the problem.

On the other hand, if all the knowledge could be coded as local preferences, it could be used as heuristics to guide the scheduler to prefer local assignments that were found to yield better solutions.

We solve this problem by automatically generating local temporal preference information, from global preferences, via a machine learning approach, and using a representation of local preferences in terms of soft constraints.

This paper recalls the current formalism and tractability results for soft temporal constraint problems, and describes the new machinery we have developed for both learning and solving a subclass of soft problems which uses fuzzy constraints [12][1]. The main results of the paper can thus be listed as follows: (1) a learning algorithm which induces local preferences from solution ratings (in Section 4); (2) some new theoretical results on the use of path-consistency in soft temporal constraints (in Section 5.1); (3) a solver based on these results, which solves temporal fuzzy constraint problems from a tractable class (in Section 5.1); (4) another solver based on a "chopping" approach, which is less general but much more efficient (in Section 5.2); (5) a random-problem generator, which extends the usual ones for hard and soft CSPs to deal with temporal constraints (in Section 3); (6) experimental results to show the behaviour of the learning module and of the two solvers, and to compare the solvers (in Section 5.3).

A learning approach related to what we use in this paper has already been used for soft CSPs without temporal information [11, 1], and is here adapted to soft temporal CSPs. Notice that this adaptation is not straightforward, since here we focus on a class of tractable temporal problems, and we build both the solvers and the learning module to work with this class of problems.

[1] Although they could be used also for other classes of soft temporal constraints.

This poses some restrictions on the learning algorithm which could in principle undermine its ability to learn the preferences over the constraints. Some other preliminary ideas which have led to the development described in this paper have been presented in [9]. However, in [9] the algorithm on which our learning module is based was not present, nor the experimental scenario and results. Another related paper is [8], which contains some of the tractability results for classes of soft temporal problems, which we use in this paper to define both our solvers. However, additional theoretical results are needed to define one of the two solvers we present here. The solving approach we use in our experiments is related to the literature of fuzzy temporal constraints [5, 10, 15]. Moreover, the translation of a classical CSP into a fuzzy CSP has been explored also in [6].

2 Background

2.1 Temporal constraint problems with preferences

In the Temporal CSP framework (TCSP) [4], variables represent events happening over time, and each constraint gives an allowed range for the distances or durations, expressed as a set of intervals over the time line. Satisfying such a constraint means choosing any of the allowed distances. A solution for a TCSP is an assignment of values to its variables such that all temporal constraints are satisfied.

General TCSPs are NP-hard. However, TCSPs with just one interval for each constraint, called STPs, are polynomially solvable [4].

TCSPs are able to model just *hard* temporal constraints. This means that all constraints have to be satisfied, and that the solutions of a constraint are all equally satisfying. However, in many real-life some solutions are preferred with respect to others. Therefore the global problem is not to find a way to satisfy all constraints, but to find a way to satisfy them optimally, according to the preferences specified. To address such problems, recently [8] a new framework has been proposed, where each temporal constraint is associated with a preference function, which specifies the preference for each distance. This framework is based on a simple merging of TCSPs and soft constraints, where for soft constraints we have taken a general framework based on semirings [3]. The result is a class of problems called Temporal Constraint Satisfaction Problems with Preferences (TCSPPs).

A *soft temporal constraint* in a TCSPP is represented by a pair consisting of a set of disjoint intervals and a preference function: $\langle I = \{[a_1, b_1], \ldots, [a_n, b_n]\}, f\rangle$, where $f : I \rightarrow A$ is a mapping of the elements of I[2] into preference values taken from a set A.

Such a set A is ordered (totally or partially), and, following the style of semiring-based soft constraints [3], such an ordering $\leq$ is defined via an operation on elements of A, denoted by $+$, as follows: $a \leq b$ iff $a + b = b$. Informally, $a \leq b$ means that b is better than a.

[2]Here by I we mean the set of all elements appearing in the intervals of I.

A *solution* to a TCSPP is a complete assignment to all the variables that satisfies the distance constraints. Each solution has a *global preference value*, obtained by combining the local preference values found in the constraints. This combination operation is usually denoted by $\times$. The two operations ($+$ and $\times$), together with the set of preference values A, form a mathematical structure called a semiring [3], and usually written as a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, where 0 and 1 are, respectively, the worst and best preference in the ordering.

The two semiring operations allow for complete solutions to be evaluated in terms of the preference values assigned locally. More precisely, given a solution t in a TCSPP with associated semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, let $T_{ij} = \langle I_{i,j}, f_{i,j}\rangle$ be a soft constraint over variables X_i, X_j and (v_i, v_j) be the projection of t over the values assigned to variables X_i and X_j (abbreviated as $(v_i, v_j) = t_{\downarrow X_i, X_j}$). Then, the corresponding preference value given by f_{ij} is $f_{ij}(v_j - v_i)$, where $v_j - v_i \in I_{i,j}$. Then the global preference value of t, $val(t)$, is defined to be $val(t) = \times\{f_{ij}(v_j - v_i) \mid (v_i, v_j) = t_{\downarrow X_i, X_j}\}$. The optimal solutions of a TCSPP are those solutions which have the best global preference value, where "best" is determined by the ordering $\leq_S$ of the values in the semiring.

For example, the semiring $S_{fuzzy} = \langle [0,1], max, min, 0, 1\rangle$ is usually used to model fuzzy constraints [12], which we will use in this paper. Preferences are between 0 and 1, the global preference of a solution is the *minimum* of all the preferences associated with the distances selected by this solution in all constraints, and the best solutions are those with the *maximal* preference.

A special case of TCSPPs occurs when each constraint contains a single interval. We call such problems *Simple Temporal Problems with Preferences* (STPPs). These problems are not as general as TCSPPs but can however model many real situations.

We can perform two operations on soft simple temporal constraints: *intersection* and *composition*. Given two such constraints $C_1 = \langle I_1, f_1\rangle$ and $C_2 = \langle I_2, f_2\rangle$ the intersection is the constraint $C_1 \oplus C_2 = \langle I_1 \cap I_2, f\rangle$, where $\cap$ is the usual intersection of intervals and $f(a) = f_1(a) \times f_2(a), \forall a \in I_1 \cap I_2$. The composition of the two constraints is again a constraint $C_1 \otimes C_2 = \langle \tilde{I}, \tilde{f}\rangle$, where $\tilde{I} = \{r | \exists r_1 \in I_1, \exists r_2 \in I_2 r = r_1 + r_2\}$ and $\tilde{f}(r) = \sum\{f_1(r_1) \times f_2(r_2) | r = r_1 + r_2, r_1 \in I_1, r2 \in I_2\}$. We can use these two operations to perform constraint propagation over STPPs. In particular, we can achieve a local consistency notion similar to path-consistency, but adapted to deal with temporal soft constraints. Applying path consistency to an STPP means considering all triangles of constraints, say (C_1, C_2, C_3), composing any two of them, say C_1 and C_2, and then intersecting the resulting constraint with the other, i.e. $(C_1 \otimes C_2) \oplus C_3$. This is performed until stability is reached, that is, until one sweep of path consistency wouldn't result in any changes.

While in general TCSPPs are NP-hard, under certain restrictions on the "shape" of the preference functions and on the semiring, STPPs are tractable. In particular, this happens when the preference functions are semi-convex. A *semi-convex* function f is one such that, for all Y, the set $\{X \text{ such that } f(X) \geq Y\}$ forms an interval. It is easy to see that semi-convex functions include linear ones, as well as convex and some step functions. For example, the *close to k* criterion (which gives a higher preference to time points which are closer to k) cannot be coded into a linear preference function, but it can be easily specified by a semi-convex preference function.

STPPs with semi-convex preference functions and a semiring with a total order of preference values and an idempotent multiplicative operation can be solved in polynomial time [8]. Moreover, semi-convex preference functions are closed with respect to path-consistency.

2.2 Learning soft temporal constraints

It is not always easy to specify the preference function in each temporal constraint; sometimes it is easier to specify global preference functions, to be associated to entire solutions. For this reason, and since the whole TCSPP machinery is based on local preference functions, we developed a methodology to induce local preferences from global ones. Here we focus on STPPs, which, as noted above, are tractable, rather than general TCSPPs.

The problem of learning preferences in STPPs, from examples of solutions ratings, can be formally described as an inductive learning problem [14]. Inductive learning can be defined as the ability of a system to induce the correct structure of a map d which is known only for particular inputs. More formally, defining an example as a pair $(x, d(x))$, the computational task is as follows: given a collection of examples of d, i.e., the *training set*, return a function h that approximates d. Function h is called a hypothesis.

A common approach to inductive learning, especially in the context of neural networks, is to evaluate the quality of a hypothesis h (on the training set) through an *error function* [7]. An example of popular error function, that can be used over the reals, is the sum of squares error [7]: $E = \frac{1}{2}\sum_{i=1}^{n}(d(x_i) - h(x_i))^2$, where $(x_i, d(x_i))$ is the i-th example of the training set.

Given a starting hypothesis h_0, the goal of learning is to minimize the error function E by modifying h_0. One popular way of doing this is to use a technique called *gradient descent* [7]. Specifically, the set of parameters W, which define the current hypothesis, is initialized to small random values at time $\tau = 0$ and updated at time $\tau + 1$ according to the following equation: $W(\tau + 1) = W(\tau) + \Delta W(\tau)$, where $\Delta W(\tau) = -\eta \frac{\partial E}{\partial W(\tau)}$, and η is the step size used for the gradient descent. Learning is stopped when a minimum of E is reached. Note that, in general, there is no guarantee that the found minimum is global.

Learning in our context can be used to find suitable preference functions to be associated to the constraints of a given STP. More precisely, let $P = (V, C)$ be an STP where V is a set of variables with domains consisting of time instants, and C is a set of distance constraints of the form $l \leq X - Y \leq u$, where $X, Y, \in V$ and l, u are time points. Let f be a function $f : S \rightarrow U$, where S is the set of solutions to P and U is a set of values indicating the "quality" of the solution. The learning task consists of transforming the STP into an STPP, with each constraint $c_{i,j} \in C$ replaced by a soft constraint $\langle c_{i,j}, f_{i,j} \rangle$, where $f_{i,j}$ is the local preference function for $c_{i,j}$.

The examples to be used in the learning task consist of pairs $(s, f(s))$, where s is a solution to the original STP and $f(s)$ is its "score". In the following, we use P to denote an STP and P' to denote a corresponding STPP. Also, $val_{P'}(t)$ is used to indicate the value of a solution t over P'.

Let P and f be as defined above, and suppose a set of examples $TR = \{(t_1, r(t_1)) \ , \ldots, \ (t_m, r(t_m))\}$ is given. To infer the local preferences, we must also be given the following: a semiring whose element set A contains the values $r(t_i)$ in the examples; and a distance function over such a semiring. For example, if the score values are positive real numbers, we could choose the semiring $\langle \Re^+, min, +, +\infty, 0 \rangle$ and, as distance function, the usual one over reals: $dist(val_{P'}(t), r(t)) = \mid val_{P'}(t) - r(t) \mid$. Given all the above, the goal is to define a corresponding STPP P' over the same semiring such that P and P' have the same set of variables, domains and interval constraints, and for each t such that $(t, r(t))$ is an example, $dist(val_{P'}(t), r(t)) < \epsilon$, where $\epsilon > 0$ and small.

If the first condition is true, the only free parameters that can be arbitrarily chosen in order to satisfy the other conditions are the values to be associated to each distance. For each constraint $c_{ij} = \{[a_1, b_1], \ldots, [a_n, b_n]\}$ in P,the idea is to associate, in P', a free parameter w_d, where $d = X_j - X_i$ (note that such a parameter must belong to the set of the chosen semiring), to each element d in $I = \bigcup_i [a_i, b_i]$. This parameter will represent the preference over that specific distance. With the other distances, those outside I, we associate the constant $\mathbf{0}$, (the lowest value of the semiring (w.r.t. $\leq_S$)).

If I contains many time points, we would need a great number of parameters. To avoid this problem, we can restrict the class of preference functions to a subset which can be described via a small number of parameters. For example, linear functions just need two parameters a and b, since they can be expressed as $a \cdot (X_j - X_i) + b$. In general, we will have a function which depends on a set of parameters W, thus we will denote it as $f_W : (W \times I) \to A$.

The value assigned to each solution t in P' is

$$val_{P'}(t) = \prod_{c_{ij} \in P'} [\sum_{d \in \bigcup_{D \in I_{ij}} D} \text{check}(d, t, i, j) \times f_W(d)] \tag{1}$$

where $\prod$ generalizes the $\times$ operation, $\sum$ generalizes $+$, I_{ij} is the set of intervals associated to constraint c_{ij}, and $\text{check}(d, t, i, j) = \mathbf{1}$ if $d = t \downarrow_{X_j} - t \downarrow_{X_i}$ and $\mathbf{0}$ otherwise. Note that, for each constraint c_{ij}, there is exactly one distance d such that $\text{check}(d, t, i, j) = \mathbf{1}$, namely $d = t \downarrow_{X_j} - t \downarrow_{X_i}$. Thus, $val_{P'}(t) = \prod_{c_{ij} \in P'} f_W(t \downarrow_{X_j} - t \downarrow_{X_i})$. The values of the free parameters in W may be obtained via a minimization of the error function, which will be defined according to the distance function of the semiring.

To force the learning framework to produce semi-convex functions, we can specialize it for a specific class of functions with this property. For example, we could choose convex quadratic functions of the form $f(d) = a \cdot d^2 + b \cdot d + c$, where $a \leq 0$. In this case we just have three parameters to consider: $W = \{a, b, c\}$. Of course, by choosing a specific class of semi-convex functions, not all local preference shapes will be representable.

3 The random problem generator

The random generator we have developed focuses on a particular subclass of semi-convex preference functions: convex quadratic functions of the form $ax^2 + bx + c$, with $a \leq 0$. This choice has been suggested by the expressiveness of such a class of functions. In fact, we can notice that this class of functions includes constant, linear, and semi-convex quadratic functions. Moreover, it is easy to express functions in this class: we just need to specify three parameters.

Moreover, the generator generates fuzzy STPPs, thus preference values are between 0 and 1, and they are combined using the max-min approach. A reason for this choice is the fact that the min operator is idempotent, thus the generated problems, according to the results in [8], are tractable. Moreover, the fuzzy approach has been shown to be useful in many real-life problems, as it is demonstrated by the interest in fuzzy theory and by several arguments for its generality.

An STPP is generated according to the value of the following parameters: (1) number n of variables; (2) range r for the initial solution: to assure that the generated problem has at least one solution, we first generate such a solution, by giving to each variable a random value within the range $[0, r]$; (3) density: percentage of constraints that are not universal (that is, with the maximum range and preference 1 for all interval values); (4) maximum expansion from initial solution (max): for each constraint, the bounds of its interval are set by using a random value between 0 and max, to be added to and subtracted from the timepoint identified for this constraint by the initial solution; (5) perturbation of preference functions (pa, pb, pc): we recall that each preference function can be described by three values (a, b, and c); to set such values for each constraint, the generator starts from a standard quadratic function which passes through the end points of the interval, with value 0, and the middlepoint, with value 0.5, and then modifies it according to the percentages specified for a, b, and c.

For example, if we call the generator with the parameters $\langle 10, 20, 30, 40, 20, 25, 30\rangle$, it will generate a fuzzy STPP with 10 variables. Moreover, the initial solution will be chosen by giving to each variable a value between 0 and 20. Among all the constraints, 70% of them will be universal, while the other 30% will be specified as follows: for each constraint, consider the timepoint specified by the initial solution, say t; then the interval will be $[t - t1, t + t2]$, where $t1$ and $t2$ are random numbers between 0 and 40. Finally, the preference function in each constraint is specified by taking the default one and changing its three parameters a, b, and c, by, respectively, 20%, 25%, and 30%.

To compare our generator with the usual one for classical CSPs, we notice that the maximum expansion (max) for the constraint intervals roughly corresponds to the tightness. However, we do not have the same tightness for all constraints, because we just set an upper bound to the number of values allowed in a constraint. Also, we do not explicitly set the domain of the variables, but we just set the constraints. This is in line with other temporal CSP generators, like the one in [13].

4 The learning module

We have developed a learning module which can learn fuzzy STPPs where the preference functions are quadratic functions of the form ax^2+bx+c with $a \leq 0$, which are exactly those functions generated by our random generator.

The input is a set of pairs consisting of a solution and its preference. Part of this set will be used as the training set, and the rest as the test set. The hard version of the problem, that is, the temporal constraints between the variables, is given as input as well.

To each temporal constraint, we then associate three parameters a, b, c, which will specify its preference function. These parameters are initialized to $a = 0$, $b = 0$, and $c = 1$, that is, a constant function.

Learning is performed via a gradient descent technique using an approximated version of the min operation which is continuous and derivable. In words, we approximate the derivative of the min function over each of its arguments, which is a step function, via two sigmoid functions. More precisely, the approximation depends on the value of a parameter, called β: $min^{\beta}(x_1, \cdots, x_n) = -\frac{1}{\beta} ln(\frac{1}{n}\sum_{i=1}^{n} e^{-\beta x_i})$. Higher beta values give better approximations, but yields an overall slower learning algorithm and with a higher chance of getting into local minima. In our experiments we have used $\beta = 8$, which showed a good trade-off between efficiency and accuracy.

At each learning step, we consider an example from the training set, say $(s, f(s))$, where s is a solution and $f(s)$ is the preference for such a solution. Then we compute the guess for the preference of s, say $g(s)$, according to the current set of temporal soft constraints. Then the error, is computed using the true and the guessed preferences: $E = \frac{1}{2}\sum_{s \in Tr}(f(s) - g(s))^2$. In our fuzzy case this formula becomes: $E = \frac{1}{2}\sum_{s \in Tr}(f(s) - [min_{i=1,\cdots,\nu}\{a_i s_i^2 + b_i s_i + c_i\}])^2$, where ν is the number of constraints. And, considering the approximation of the min, we have: $E = \frac{1}{2}\sum_{s \in Tr}(f(s) + \frac{1}{\beta}\ln(\frac{1}{\nu}\sum_{i=1}^{\nu} \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)})])^2$.

The a, b, and c parameters of the function on each constraint are then modified following the gradient descent rule. That is:

$$\tilde{a_i} = a_i - \eta[(f(s) - g(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(-s_i^2 \ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}))]$$

$$\tilde{b_i} = b_i - \eta[(f(s) - g(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(-s_i \ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}))]$$

$$\tilde{c_i} = c_i - \eta[(f(s) - g(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(- \ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}))]$$

We recall that η is the step size for the gradient descent, which in our case was set to 5×10^{-5}. Smaller η values yield smaller changes of the parameters, which usually makes local minima less probable.

Max	Mean error (min,max)	Number of examples
20	0.03 (0.02,0.04)	500
30	0.03 (0.02,0.04)	600
40	0.0333 (0.02,0.05)	700

Table 1: Mean absolute error and number of examples for learning preferences in some STPPs.

After the updating of the parameters, we compute the absolute mean error over the training set, that is, the mean over the set of differences between $f(s)$ and $g(s)$, for all s. We stop when for 100 consecutive times, the error didn't improve of at least 20% of its previous value.

Once the chosen stopping criterion is met, the constraint network computed by the algorithm is an STPP with preference functions in the shape of convex quadratic functions, whose solutions are ranked very similarly to the original examples in the input. The similarity achieved by the learning algorithm is measured by comparing the solution ratings of the obtained problem with the examples contained in the test set, and by computing the absolute mean error on this set.

Our learning module has been tested on randomly generated problems. Every test involves the generation of an STPP via our generator, and then the generation of some examples of solutions and their rating. Then the STPP, without its preference functions, is given to the learning module, which, starting from the examples, learns new preference functions over the constraints, until the stopping criterion is met.

Table 1 shows the number of examples in the training (and also in the test) set, and the mean error (computed as the average of the mean error for three problems), for learning preferences in STPPs with 20 variables, range = 40, density = 40%, and function perturbation parameters 10, 10, and 5. The maximum expansion, which, we recall, is related to the tightness notion, is 20, 30, and 40. What can be noticed is that the mean error increases as the parameter max increases, even if one uses more examples (which should ease the learning process). This is due to the fact that a larger value for max may yields larger intervals, and thus preference functions with larger domains and a larger number of solutions, which require more work from the learning algorithm. This trend seems to be confirmed also by other experimental results not reported here. These results show that it is feasible to learn local preferences from solution ratings, even when no local preference is given. However, it is probable that in many scenarios we will have some local preferences given by the user, and others to be learnt by the system.

5 Solving soft temporal constraints

Once we have an STPP, either after a learning phase or not, we need to solve it, that is, to find an optimal solution according to its preferences. In this section we will describe two solvers for tractable classes of soft temporal constraint problems. The first one, path-solver, is more general, while the other one, called chop-solver, is more efficient. We will also present experimental results over randomly-generated problems to show the difference in performance between the two solvers.

5.1 A solver based on path consistency

The tractability results for STPPs that are contained in [8] can be translated in practice as follows: to find an optimal solution for an STPP, we can first apply path consistency and then use a search procedure to find a solution without the need to backtrack. More precisely, we can add to the results in [8] (and summarized in Section 2) the following results, whose proofs are omitted for lack of space.

Theorem 1 *Given an STPP P, let us call P' the STPP obtained by applying path-consistency to P. Then, all preference functions in P' have the same best preference level, which is lower than or equal to the original one.*

Theorem 2 *Consider the STP obtained from the STPP P' by taking, for each constraint, the sub-interval corresponding to the best preference level. Then, the solutions of such an STP coincide with the best solutions of the original P (and also of P'). Therefore, finding a solution of this STP means finding an optimal solution of P.*

Our first solver, which we call path-solver, relies on these results. In fact, the STPP solver takes as input an STPP with semi-convex preference functions and fuzzy temporal constraints, and returns an optimal solution of the given problem, working as follows and as shown in Figure 1: first, path consistency is applied to the given problem, by function STPP_PC-2, producing a new problem P'; then, an STP corresponding to P' is constructed, applying REDUCE_TO_BEST to P', by taking the subintervals corresponding to the best preference level and forgetting about the preference functions; finally, a backtrack-free search is performed to find a solution of the STP, specifically the earliest one is returned by function EARLIEST_BEST. All these steps are polynomial, so the overall complexity of solving an STPP with the above assumptions is polynomial. In Figure 1 we show the pseudocode for this solver.

In Figure 2 we show some results for finding an optimal solution for STPPs generated by our generator. Path-solver has been developed in C++ and tested on a Pentium III 1GHz.

As it can be seen, this solver is very slow. The main reason is that it uses a pointwise representation of the constraint intervals and the preference functions. This makes the solver more general, since it can represent any kind of preference functions, even those that don't have an analytical representation via a small set of parameters. In fact, even starting from convex quadratic functions, which need just three parameters, the first solving phase, which applies path consistency, can yield new preference functions which are not representable via

Pseudocode for path-solver
1. **input** STPP P;
2. STPP P'=STPP_PC-2(P);
3. **if** P' inconsistent **then** exit;
4. STP P"=REDUCE_TO_BEST(P');
5. **return** EARLIEST_BEST(P").

Figure 1: Path-solver.

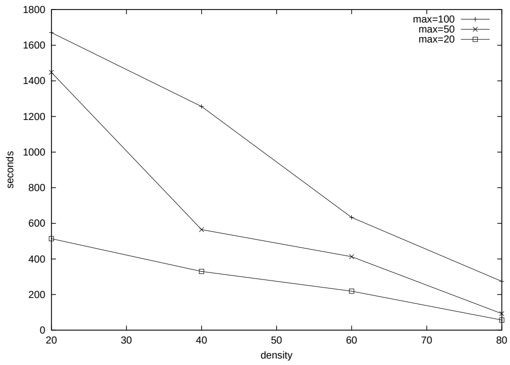

Figure 2: Time needed to find an optimal solution (in seconds), as a function of density (d). The other parameters are: n=50, r=100, pa=20, pb=20, and pc=30. Mean on 3 examples.

three parameters only. For example, we could get semi-convex functions which are generic step functions, and thus not representable by giving new values to the initial three parameters.

5.2 A solver based on a chopping procedure

The second solver for STPPs that we have implemented, and that we will call 'chop-solver', is based on the proof of tractability for STPPs, with semi-convex preference functions and idempotent multiplicative operator of the underlying semiring, described in [8]. Let's briefly recall the main argument of that proof. The first step is to obtain an STP from a given STPP. In order to do this, we reduce each soft constraint of the STPP, say $\langle I, f\rangle$, to a simple temporal constraint without preferences. Consider $y \in A$, a value in the set of preferences. Then, since the function f on the soft constraint is semi-convex, the set $\{x : x \in I, f(x) \geq y\}$ forms an interval, i.e. a simple temporal constraint. Performing this transformation on each soft constraint of the original STPP we get an STP, wich we refer to as STP_y. The proof states that the set of solutions of the STP_{opt}, where opt represents the highest level at which the derived STP is consistent, coincides with the set of optimal solutions of the STPP.

```
Pseudocode for chop-solver
input STPP P;
input precision;
integer n=0;
real lb=0, ub=1, y=0;
if(CONSISTENCY(P,y))
   y=0.5, n=n+1;
   while (n<=precision)
    if(CONSISTENCY(P,y))
      lb=y, y=y+(ub-lb)/2, n=n+1;
     else
       ub=y, y=y-(ub-lb)/2, n=n+1;
    end of while;
    return solution;
else exit.
```

Figure 3: Chop-solver.

This solver works with STPPs with semi-convex quadratic functions (lines and convex parabolas), thus it is less general than path-solver. Preferences are handled via the fuzzy semiring. This means that the set of preferences we are considering is the interval [0,1].

The solver finds an optimal solution of the STPP identifying first STP_{opt} and returning its earliest or latest solution. *Opt* is found by performing a binary search in $[0, 1]$. The bound on the precision of a number, that is the maximum number of decimal coded digits, implies that the number of search steps is always finite. Moreover, our implementation allows the user to specify at the beginning of the solving process the number n of digits he or she wants for the optimal solution's preference level.

The search for the optimal preference level starts with $y = 0$. Since STP_0 is the STP we would obtain by considering all the soft constraints as hard constraints, that is, with preference function equal to 1 on the elements of the interval and to 0 everywhere else, the algorithm first checks if the hard version of the problem is consistent. If it is found not to be consistent, the algorithm stops, informing the user that the whole problem is inconsistent. Otherwise the search goes on. Three variables are maintained during the search: ub, which contains the lowest level at which an inconsistent STP was found, lb, which contains the highest level at which a consistent STP was found, and y, which is the current level at which we need to perform the "chopping". It is easy to see that ub and lb are the upper and lower bound of the portion of the [0,1] interval to which we can restrict our search. The three values are updated depending on the outcome of the consistency test. In Figure 3 we show the pseudocode for this solver.

The actual chopping and the consistency test on the STP obtained are performed by function CONSISTENCY. It receives, as input, the level at which the

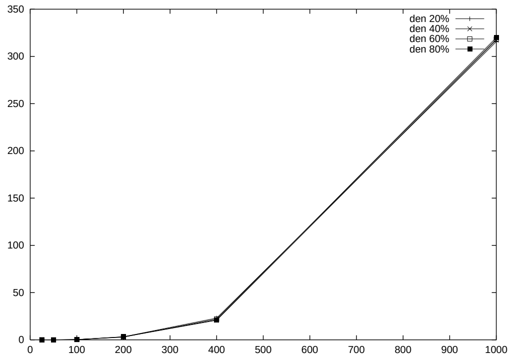

Figure 4: Time, in seconds, (y-axis) required by chop-solver to solve, varying the number of variables (x-axis) and the density, with r=100000 max=50000, pa=5, pb=5 e pc=5. Mean on 10 examples. (The curves all overlap.)

chop must be performed and the STPP. For each constraint of P, it considers three types of functions: a constant, a line or a semi-convex parabola. It then finds the intersection of the function with the constant function at the chopping level. As it finds the intersection, for each constraint it fills in the distance matrix F. This matrix is $N \times N$, where N is the number of variables of the problem. It represents the distance graph of the STP [4]. This means that if the constraint between variable i and variable j is the interval $[a, b]$, then $F[i][j] = b$ and $F[j][i] = -a$. At this point, we use the fact that an STP is consistent if and only if its distance graph has no negative cycles [4]. In order to accomplish this, we run Floyd-Warshall's all-shortest-paths algorithm on F and then check the diagonal elements. If no diagonal elements are negative, we can conclude that STP_y is consistent. If we have already reached the number of decimal digits the user wanted, then we return either the earliest or the latest solution, respectively corresponding to the assignments $x_i = -F[i][0]$ and $x_i = F[0][i]$. If instead one or more diagonal elements are negative, we can conclude that the STP_y is inconsistent and either return the solution of the last consistent STP or keep searching at lower levels of preference. The solution we return is always made of integers, that is, in the case of the earliest solution, the real numbers found intersecting the preference functions with the chopping level are approximated to the first larger integer while, for the latest, the approximation is to the largest smaller integer.

Figure 4 shows some experimental results for chop-solver. We have used the same random generator used to test path-solver, but with a different representation of the constraints generated, since chop-solver uses a parametric representation while path-solver needs a point-wise representation.

We have tested chop-solver by varying the number of variables, from a minimum of 25 up to a maximum of 1000, and the density from 20% to 80%.

From Figure 4 we can conclude that chop-solver is only slightly sensitive to variations in the density, and it is very sensitive to the number of variables, since a higher number of variables yields an increase of the number of constraints on which the intersection procedure must be performed.

The choice of maintaining a fixed maximum enlargement of the intervals, that can be interpreted as a fixed tightness, is justified by the continuous representation of the constraint this solver uses. In fact, each constraint is represented by only two integers for the left and right ends of the interval and three real numbers as parameters of the function. Increasing max affects this kind of representation of a constraint only making these values bigger in modulo. This change however does not affect any of the operations performed by chop-solver.

5.3 Path-solver vs. chop-solver

In Table 2 we can see a comparison between chop-solver and path-solver.

	D=20	D=40	D=60	D=80
path-solver (n=30)	515.95	235.57	170.18	113.58
chop-solver (n=30)	0.01	0.01	0.02	0.02
path-solver (n=40)	1019.44	516.24	356.71	320.28
chop-solver (n=40)	0.03	0.03	0.03	0.03
path-solver (n=50)	2077.59	1101.43	720.79	569.47
chop-solver (n=50)	0.05	0.05	0.06	0.07

Table 2: Time in seconds, used by path-solver and chop-solver to solve problems with $n = 30, 40, 50$, $r = 100$, $max = 50$, $pa = 10$, $pb = 10$, and $pc = 5$ and varying density D. Results are mean on 3 examples.

It appears clear that chop-solver is always much faster than path-solver. It can be noted, however, that chop-solver finds more constrained problems a little more difficult. This fact can be partially explained by the way problems are generated: having a higher density means having more constraints with non trivial parabolas, i.e. $a \neq 0$. The intersection procedure in this case is a little more complicated than in the case of constants or lines. On the other hand, with a higher density, path-solver has to deal with smaller constraints (w.r.t. the default ones), and thus the pointwise representation is less of a problem.

Chop-solver is also more precise, since it can find an optimal solution with a higher precision. It must be kept in mind, though, that path-solver is more general. In fact, the point-to-point representation of the constraints needed by path-solver, to be blamed for its poor performance, allows one to use any kind of semi-convex function, e.g. step functions, that cannot be easily compactly parametrized. Moreover, even wanting to extend the types of parametrized functions in the continuous representation for chop-solver, we must remember that the system deriving from intersecting the constant at chopping level and the function must be solvable in order to find the possible intersections.

6 Future work

We plan to further test the overall system, composed of the solvers and the learning module, using other classes of randomly generated STPPs and also real-life problem instances such as satellite event scheduling. We also plan to extend our solver to deal with soft temporal problems which are not tractable. Moreover, we believe that the ideas underlying chop-solver can be used also for solving soft constraints in general, not just temporal ones. This would allow for the choice of the precision with which an optimal solution is found. This approach is related to hybrid algorithms based on abstraction of soft constraints, where a series of abstraction and concretization mappings can improve the bounds over an optimal solution [2].

References

[1] A. Biso, F. Rossi, and A. Sperduti. Experimental Results on Learning Soft Constraints. *Proc. KR 2000*, Morgan Kaufmann, 2000.

[2] Abstracting Soft Constraints. S. Bistarelli, P. Codognet, Y. Georget, F. Rossi. *Proc. ERCIM/Compulog Net work. on constraints*, Springer, LNAI 1865, 2000.

[3] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

[4] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. Artificial Intelligence, Vol. 49, 1991, pp. 61-95.

[5] D. Dubois, H. Prade. Processing Fuzzy Temporal Knowledge. *IEEE Trans. On systems, Man, and Cybernetics*, 19:4,1989.

[6] M. Giacomin. From Crisp to Fuzzy Constraints Networks. *Proc. CP '01 Workshop on Modelling and Solving Problems with Soft Constraints*, 2001.

[7] S. Haykin. *Neural Networks: a comprehensive Foundation.* IEEE Press, 1994.

[8] L. Khatib, P. Morris, R. Morris, F. Rossi. Temporal Constraint Reasoning With Preferences. *Proc. IJCAI 2001.*

[9] L. Khatib, P. Morris, R. Morris, F. Rossi, A. Sperduti. Learning Preferences on Temporal Constraints: A Preliminary Report. *Proc. TIME 2001*, IEEE Comp. Soc. Press, 2001.

[10] R. Marin, M.A. Cardenas, M.Balsa, J.L.Sanchez. Obtaining solutions in Fuzzy Constraint Networks. *Int. Jour. of Approximate Reasoning*, 16:261-288, 1997.

[11] F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of Experimental and Theoretical Computer Science*, 1998. Vol 10.

[12] T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.

[13] E. Schwalb, R. Dechter. Coping with disjunctions in temporal constraint satisfaction problems. *Proc. AAAI-93*, 1993.

[14] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[15] L. Vila, L. Godo. On fuzzy temporal constraint networks. *Mathware and Soft Computing*, 3:315-334,1994.