Research Interests

Brent Venable, University of Padova

My interests, as a Ph.D. student in computer Science, are mainly in Constraints and Machine Learning. I have graduated in mathematichs with a thesis that combines elements from both fields since it presents conceptual description and implementation of a solver for a special class of constraint satisfaction problems and a learning module that allows to build such problems when only global knowledge is available. My latest work has been devoted to the implementation of another solver. I'll now present a description of my work on solving and learning soft temporal constraints problems, wich has been supervised by Prof. Francesca Rossi and Prof. Alessandro Sperduti (University of Padova). Lina Khatib, Robert Morris and Paul Morris (NASA Ames San Francisco) have also contributed to this research.

1 Learning and solving soft temporal constraints problems

Soft temporal constraints problems allow to describe in a natural way scenarios where events happen over time and preferences are associated to 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. Machine learning techniques can be useful in this respect.

Several real world problems involving the manipulation of temporal information in order to find an assignment of times to a set of activities or events can naturally be viewed as having preferences associated with local temporal decisions, where by a local temporal decision we mean one 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 a local preference on durations. In real world scheduling problems, for example, it is sometimes easier to see how preferable is a solution, 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.

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.

Although very expressive, 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 [6] 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 merge of TCSPs and soft constraints, where for soft constraints we have taken a general framework based on semirings [2]. 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 $f : I^1 \rightarrow A$, is a mapping of the elements of I into preference values, taken from a set A.

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. To formalize the process of combining local preferences into a global preference, and comparing solutions, we impose a c-semiring structure on the TCSPP framework.

C-semirings allow for a partial order relation $\leq_S$ over A to be defined as $a \leq_S b$ iff $a+b = b$ where $+$ is the additive operator of the semiring. Informally, $\leq_S$ gives us a way to compare tuples of values and constraints, and $a \leq_S b$ can be read *b is better than a*. Moreover, one can prove that given a semiring[2] with a set of values A, each preference function f associated with a soft constraint $\langle I, f\rangle$ of a TCSPP takes an element from I and returns an element of A, where A is the carrier of a semiring. This allows us to associate a preference with a duration or a distance. 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 $v_j - v_i \in I_{i,j}$. Finally, where $F = \{x_1, \ldots, x_k\}$ is a set, and $\times$ is the multiplicative operator on the semiring, let $\times F$ abbreviate $x_1 \times \ldots \times x_k$. 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.

The semiring underlying the problems targeted here is $S_{fuzzy} = \langle [0,1], max, min, 0, 1\rangle$, used for fuzzy constraint solving [9]. The global preference value of a solution will be the minimum of all the preference values associated with the distances selected by this solution in all constraints, and the best solutions will be those with the maximal value.

A special case occurs when each constraint of a TCSPP contains a single interval. We call such problems *Simple Temporal Problems with Preferences* (STPPs). 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 combination 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 $\tilde{f}(r) = \sum\{f_1(r_1) \times f_2(r_2) | r = r_1 + r_2, r_1 \in I_1, r2 \in I_2\}$.

In [6] it has been shown that, while in general TCSPPs are NP-hard, under certain restrictions on the "shape" of the preference functions and on the semiring, STPPs are tractable.

A *semi-convex* function f is one such that, for all Y, the set $\{X$ such that $f(X) \geq Y\}$ forms an interval. It is easy to see that semi-convex functions include linear ones, as well

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

[2] For simplicity, from now on we will write *semiring* meaning *c-semiring*.

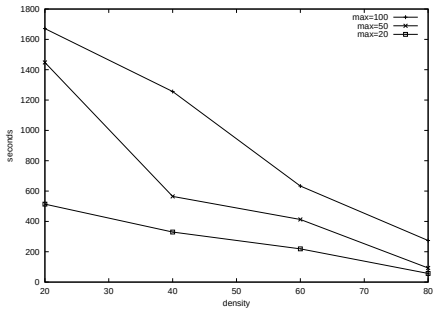

Figure 1: Time needed to find an optimal solution (in seconds), as a function of density (d). The other parameters are: n=50, r=100, pa=20, pb=20, and pc=30. Mean on 3 examples.

as convex and some step functions. For example, the *close to k* criterion cannot be coded into a linear preference function, but it can be easily specified by a semi-convex preference function.

It is proven in [6] that 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.

The tractability results for STPPs can be translated in practice as follows: to find an optimal solution for an STPP, we can first apply path consistency (suitably adapted to STPPs, see [6]) and then use a search procedure to find a solution without the need to backtrack. More in details, it is possible to show that: (1) applying path consistency to an STPP means considering all triangles of constraints (C_1, C_2, C_3) composing two of them 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); (2) semi-convex functions are closed w.r.t. path consistency: if we start from an STPP P with semi-convex functions, and we apply path consistency, we get a new STPP P' with semi-convex functions (see [6]); (3) after applying path consistency, all preference functions in P' have the same best preference level; (4) 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 solving module, 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 returns an optimal solution of the given problem, working as follows: first, path consistency is applied to the given problem producing a new problem P'; then, an STP corresponding to P' is constructed 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. All these steps are polynomial, so the overall complexity of solving an STPP with the above assumptions is polynomial. This STPP solver has been tested both on toy problems and on randomly-generated problems. 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 $a \leq 0$. The choice has been suggested both by the expressiveness of such a class of functions and also by the facility of expressing functions in this class (just three parameters). Moreover, it generates fuzzy STPPs, thus preference values are between 0 and 1. In Figure 1 we show some results for finding an optimal solution for STPPs generated by our generator.

The second solver for STPPs that we have implemented, and that we will call **chop-solver**, is based on the fact that the set of optimal solutions of the STPP coincides with the set of solutions of an STP, STP_{opt}, that is obtained considering only the intervals mapped in values higher than $y = opt$ where opt is the highest level at which the induced STP is consistent[6]. This solver works with STPPs with semi-convex quadratic functions (lines and convex parabolas) based on 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

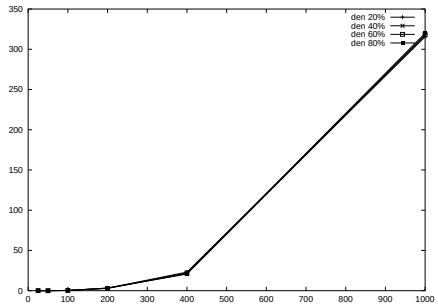

Figure 2: Time, in seconds, (y-axis) required by chop-solver to solve, varying the number of variables (x-axis) and the density, with r=100000 max=50000, pa=5, pb=5 e pc=5. Mean on 10 examples. (The curves all overlap.)

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.

Figure 2 shows some experimental results for chop-solver.

In Table 1, we can see a comparison between chop-solver and path-solver.

	D=20	D=40	D=60	D=80
path-solver	515.95	235.57	170.18	113.58
chop-solver	0.01	0.01	0.02	0.02

Table 1: Time in seconds, used by path-solver and chop-solver to solve problems with $n = 30$, $r = 100$, $max = 50$, $pa = 10$, $pb = 10$, and $pc = 5$ and varying density D. Results are mean on 3 examples.

It appears clear that chop-solver is much faster. It is also true that, in a sense, it's 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, 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. Keep in mind that such a pointwise representation is required in order to be able to apply path consistency. It is also true that, in general, time is dealt with as a discretized quantity, which means that, once the measuring unit that is most significant for the involved events is fixed, the problem can be automatically cast in the point-to-point representation. 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. However, the continuous representation used by chop-solver is, undoubtly, more natural because it reflects the most obvious idea, the idea we all have in mind, of such constraints, that is an interval plus a function over it. The improvement in terms of speed are impressive but the loss in generality is, on the other hand, considerable.

As noted in the introduction, it is not always easy to specify the preference functions in each temporal constraint in a way that the real-life problem at hand is faithfully modeled. For this reason, and since the whole TCSPP machinery is based on local preference functions, we propose here a method for **learning soft temporal constraints**.

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

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

to evaluate the quality of a hypothesis h (on the training set) through an *error function* [5]. An example of popular error function, that can be used over the reals, is the sum of squares error [5]: $E = \frac{1}{2}\sum_{i=1}^{n}(d(x_i) - h(x_i))^2$, 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. This can be done by using a definition of h which depends on a set of internal parameters W, i.e., $h \equiv h_W$, and then adjusting these parameters. This adjustment can be formulated in different ways, depending on whether the domain is isomorphic to the reals or not. The usual way to be used over the reals, and if h_W is continuous and derivable, is to follow the negative of the *gradient* of E with respect to W. This technique is called *gradient descent* [5].

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 $a \leq 0$. Notice that the class of such functions includes constant, linear, and semi-convex quadratic functions.

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 constraints between the variables with only intervals specified, is given as input as well. Learning is performed via a gradient descent technique using an approximated version of the min operation which is continuous and derivable. This means that at each step an example from the training set is considered and the current network of constraints computes its guess for the preference. Then the error, e.g. sum square errors, is computed using the true and the guessed preferences. The a, b, and c parameters of the function on each constraint are then modified following the gradient descent rule. After the updating of the parameters several errors are computed considering the whole training set: maximum absolute error, absolute mean error and sum of squares error. This allows to try different stopping criteria, setting thresholds on the different errors. Once the 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 solution are ranked very similarly to the original examples in the input. The performance of the learning algorithm is measured testing it on new examples contained in the test set and computing the same errors, computed during the learning phase on the training set, on this set.

The learning module has been tested on some 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 error (that is, the difference between the solution ratings in the test set and in the new problem) is small enough.

Table 2 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.

2 Additional topics of research

What has just been described is the topic to which I have devoted most of my time, however there are other research areas I'm exploring. With Francesca Rossi and Toby Walsh, I'm considering how to reason with **ceteris paribus statements** within the constraint satisfaction framework. Ceteris Paribus (all else being equal) statements allow to express preferences on certain attributes of features when they are independent of others [3]. This framework can

be very useful for preference elicitation. In fact this kind of statement provides an elegant and user-friendly way to express preferences on attributes of some feature pointing out there dependence from other features and independence from others. Their main use has been to compare complete assignments of attributes involved in the preference elicitation process, which means reasoning with the ordering introduced by the ceteris paribus semantics. We are considering the problem of reasoning efficiently with such ordering from various points of view. We have analyzed possible approximations of ceteris paribus networks with soft constraints networks based on some known semi rings, as the one for Fuzzy CSPs, the one for Weighted CSPs and the one obtained combining the two previous ones. The results we have obtained is an approximation of the partial order, in some cases with a total order, that allows very efficient procedures for reasoning with complete assignments. We are also considering a new semi ring that will describe exactly the same ordering in order to show that this problem can be indeed modeled with the semiringframework. On the other side we have tried to overcome the tractability problem proposing new semantics that remain faithful to the main idea of ceteris paribus. Although we have several ideas on this very promising topic, this work is still in a very preliminary stage.

In the future I'll be involved in continuing the research on combined systems of learning and constraint solving. Right now I'm starting to look at the possibility of actually learning the multiplicative operator of the semi ring, given some solutions and there preferences. This summer I have been accepted for the second time at the SSRP Program at NASA Ames, CA. During my stay, among other things, I'll be working at the implementation of an algorithm for Pareto Optimization of Temporal Decisions.

References

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

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

[3] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with ceteris paribus preference statements

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

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

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

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

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

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

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

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