International Journal on Artificial Intelligence Tools

A HOPFIELD-TYPE NEURAL NETWORK BASED MODEL FOR TEMPORAL CONSTRAINTS

MALEK MOUHOUB

Department of Computer Science, University of Regina
3737 Waskana Parkway,
Regina Saskatchewan, Canada, S4S 0A2
email : mouhoubm@cs.uregina.ca

In this paper we present an approximation method based on discrete Hopfield neural network (DHNN) for solving temporal constraint satisfaction problems. This method is of interest for problems involving numeric and symbolic temporal constraints and where a solution satisfying the constraints of the problem needs to be found within a given deadline. More precisely the method has the ability to provide a solution with a quality proportional to the allocated process time. The quality of the solution corresponds here to the number of satisfied constraints. This property is very important for real world applications including reactive scheduling and planning and also for over constrained problems where a complete solution cannot be found. Experimental study, in terms of time cost and quality of the solution provided, of the DHNN based method we propose provides promising results comparing to the other exact methods based on branch and bound and approximation methods based on stochastic local search.

Keywords: Temporal Reasoning; Neural Networks; Hopfield Model; Optimization Problems; Planning and Scheduling.

1. Introduction

In 1985, John Hopfield and David Tank first attempted using discrete Hopfield neural networks (DHNN) as an approximation method to solve optimization problems, mainly the Traveling Salesman Problem[1]. Since then, there has been wide spread interest in applying neural nets to solve a large variety of combinatorial optimization problems[2,3].

In this paper we will use the Hopfield model as an approximation method to solve the binary constraint satisfaction problem (CSP)[a] involving numeric and symbolic temporal constraints. We call it temporal constraint satisfaction problem (TCSP)[b]. The goal here is to look for a solution that satisfies the temporal constraints of the

[a]A CSP (Constraint Satisfaction Problem) [4,5,6,7] involves a list of variables defined on finite domains of values and a list of relations (constraints) between variables. A binary CSP is a CSP where the relations are binary.
[b]Note that this name and the corresponding acronym was used in [8]. This latter approach is different from our method in the way numeric (and also symbolic) constraints are represented. See [9] for a comparison of the two methods

problem within a given deadline. More precisely the resolution method we propose has the ability to provide a solution with a quality proportional to the allocated process time. The quality corresponds here to the number of solved constraints. This property is of interest in many real world applications such as reactive scheduling and planning where the resolution method can be interrupted at any time and provides the optimal solution at that time. We talk then about anytime method (see [10] for more details about anytime algorithms). Also the method can be used to solve over constrained problems (problems where a complete solution cannot be found) by providing a partial solution satisfying the maximal number of temporal constraints.

In a previous work[11,12], we have proposed two type of anytime methods for solving TCSPs. The first one is an exact algorithm based on partial constraint satisfaction techniques[13]. Local consistency techniques and backtrack search methods we have used to solve TCSPs in general [9] were adapted to cope with, and take advantage of, the differences between partial and complete constraint satisfaction. The exact method is based on branch and bound techniques and has the advantage to provide a solution that is guaranteed to be optimal [11]. However, as we mentioned in [11,12], this method is impractical for large size problems and is in general useful to verify the optimality and, therefore, the quality of the solution returned by the approximation methods. The second type of methods are approximation algorithms based on local search techniques (Min-Conflict Random Walk (MCRW), Steepest Descent Random Walk (STRW) and Tabu Search). This second type of methods does not guarantee the optimality of the solution provided (as it is the case of the exact method) but is obviously of interest when it provides near optimal solutions.

In order to evaluate the performance in time of the method based on DHNN we propose, experimental comparisons with the exact and approximation methods we mentioned above have been performed on randomly generated temporal constraint problems. This study shows that the method based on DHNN presents promising results in the case of over-constrained problems.

The rest of the paper is organized as follows. In the next section we will present, through an example, our temporal model TemPro which represents symbolic and numeric information in the form of temporal constraints. Section 3 and 4 are dedicated respectively to the representation of numeric and symbolic time information using Neural Networks (the Hopfield model). Experimental comparison of the DHNN based method with the other approximation methods are reported in section 5. Finally, concluding remarks are presented in section 6.

2. CSP-based Representation of Numeric and Symbolic Constraints : the model TemPro

One important issue when dealing with problems involving temporal information is the ability to manage both the symbolic and numeric aspects of time. This motivates us to develop the model TemPro[9], extending the Interval Algebra defined by Allen[14] to handle numeric constraints. TemPro transforms any problem under

Table 1. Allen primitives

Relation	Symbol	Inverse	Meaning
X precedes Y	P	$P^{\smile}$	XXX YYY
X equals Y	E	E	XXX YYY
X meets Y	M	$M^{\smile}$	XXXYYY
X overlaps Y	O	$O^{\smile}$	XXXX YYYY
X during y	D	$D^{\smile}$	XXX YYYYYY
X starts Y	S	$S^{\smile}$	XXX YYYYY
X finishes Y	F	$F^{\smile}$	XXX YYYYY

qualitative and quantitative constraints into a binary CSP where constraints are disjunctions of Allen primitives[14] (see table 1 for the definition of the 13 Allen primitives) and variables, representing temporal events, are defined on domains of time intervals. Each event domain (called also temporal window) contains the Set of Possible Occurrences (SOPO) of numeric intervals the corresponding event can take. The SOPO is the numeric constraint of the event. It is expressed by the fourfold [*earliest_start*, *latest_end*, *duration*, *step*] where :

- *earliest_start* is the earliest start time of the event.
- *latest_end* is the latest end time of the event.
- *duration* is the duration of the event.
- *step* is the discretization step corresponding to the number of time units between the start time of two adjacent intervals belonging to the event domain.

To illustrate the different components of the model TemPro let us consider the following scheduling problem[c].

Example 1

The production of two items A and B requires three mono processor machines M_1, M_2 and M_3. Each of the two items can be produced using two different ways depending on the order in which the machines are used. The process time of each machine is variable and depends on the task to be processed. The following lists the different ways to produce each of the two items (the process time for each machine is mentioned in brackets) :

item A: $M_2(3), M_1(3), M_3(6)$ *or*
$M_2(3), M_3(6), M_1(3)$

[c]This problem is taken from. [15]

item B: $M_2(2), M_1(5), M_2(2), M_3(7)$ *or*
$M_2(2), M_3(7), M_2(2), M_1(5)$

The goal here is to find a possible schedule of the different machines to produce the two items and respecting all the constraints of the problem. We also assume that items A and B should be produced within 25 and 30 units of time respectively.

In the following we will describe how is the above problem transformed into a TCSP using our model TemPro. Figure 1 illustrates the graph representation of the TCSP corresponding the the scheduling problem. A temporal event corresponds here to the contribution of a given machine to produce a certain item. For example, the event AM_1 corresponds to the use of machine M_1 to produce the item A, ..., etc. Seven events are needed in total to produce the two items as follows:

item A: $AM_2(3), AM_1(3), AM_3(6)$ or
$AM_2(3), AM_3(6), AM_1(3)$
item B: $BM_{21}(2), BM_1(5), BM_{22}(2), BM_3(7)$ or
$BM_{21}(2), BM_3(7), BM_{22}(2), BM_1(5)$

The translation to Allen primitives of the disjunction of the two sequences required to produce item B needs a 3-ary relation involving BM_1, BM_{22} and BM_3. This relation states that BM_{22} should occur between BM_1 and BM_3. Since our temporal network handles only binary relations, the way we use to represent this kind of 3-ary relations is as follows: we create an additional event (EVT_1) and represent the constraints for producing item B as shown in figure 1. The duration X of EVT_1 is greater(or equal) than the sum of the durations of BM_1, BM_{22} and BM_3.

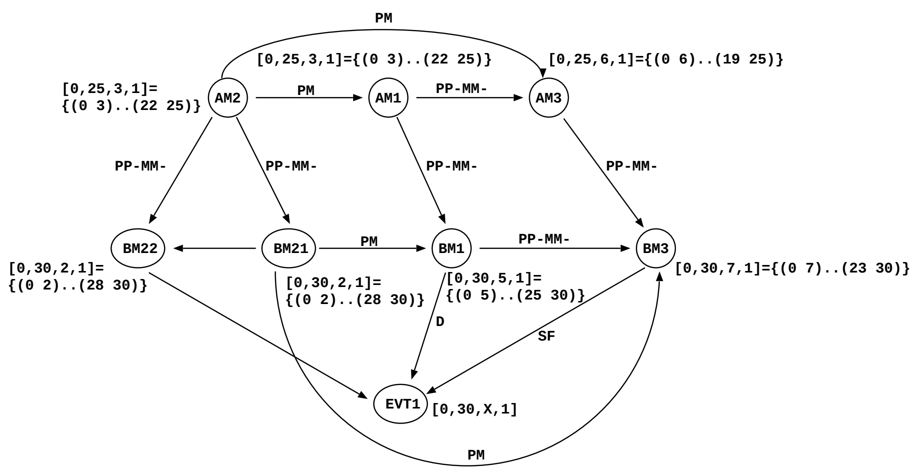

Fig. 1. TCSP corresponding to the problem presented in example 1.

3. Neural Representation of Numeric Constraints

To represent the event SOPO in terms of neurons, a two-dimensional array is adequate. One axis depicts the time line; one neuron equates to one discretization step. The other axis represents an individual SOPO event.

To represent an assignment of an interval to a particular event in terms of neural states, there should only be two neurons 'on' while the rest of the neurons 'off' for each given event. The two 'on' neurons represent the start and end times of the event interval. The distance between them (number of neurons between the start and end time neurons) represents the duration.

When dealing with the constraint and energy functions, the 'on' state is considered to be 1 and the 'off' state is considered to be 0. Each neuron a_{ij} represents a time point j for a given event i. a_{ij} can have only 2 possible values 0 or 1.

The approach we use to satisfy the different temporal constraints of the problem is essentially a Lagrangian relaxation of the constraints, i.e minimize the following energy function:

$$F = \alpha_1 C_1 + \alpha_2 C_2 + \cdots + \alpha_n C_n \tag{1}$$

where:

- each C_i is a nonnegative penalty function representing a given constraint,
- and $\forall i \ \alpha_i > 0$.

The energy function F representing the numeric constraints will then be:

$$F = \alpha F_1 + \beta F_2 \tag{2}$$

where:

- F_1 represents the fact that there should be exactly 2 activated neurons (equal to 1) per row. F_1 includes also the constraint forcing each neuron to have a digital value (0 or 1).

$$F_1 = \sum_{i=1}^{n}[\sum_{j=1}^{h} a_{ij} - 2]^2 + \sum_{i=1}^{n}\sum_{j=1}^{h} a_{ij}(1 - a_{ij}) \tag{3}$$

  where n is the number of events and h is the constant horizon (time before which all events should be processed).
- F_2 states that there are exactly 2 activated neurons per row within the temporal window of the event (EV_i) and distant by d_i.

$$F_2 = \sum_{i=1}^{n}[(\sum_{s=inf_i}^{sup_i - d_i} a_{is} + a_{i,s+d_i}) - 2]^2 \tag{4}$$

  inf_i, sup_i and d_i are respectively the earliest start time, latest end time and duration of a given event evt_i.

By comparing F_1 and F_2 with the following energy function of the Hopfield net H, we are able to find proper weights w and thresholds wo for the network.

$$H = -\frac{1}{2} \sum_{i,j,k,l} \mathbf{w_{ij,kl}}\ a_{ij} a_{kl} - \sum_{i,j} \mathbf{wo_{ij}}\ a_{ij} \tag{5}$$

If we multiply out the summations in (2) we obtain constant terms, linear terms proportional to one a_{ij} and quadratic terms with two a_{ij}'s. The quadratic terms can be represented by connections $w_{ij,kl}$ between the units, while the linear terms can be considered as thresholds.

4. Neural Representation of Symbolic Constraints

As we mentioned in section 2, in our model TemPro qualitative constraints are disjunctions of Allen primitives. Thus, to represent qualitative constraints using neural networks, we need to find a neural representation for each of the 13 primitives (see table 1 for the definition of the 13 Allen primitives). To do so we need first to define the Allen primitives in the form of a list of equations involving the end points of the intervals and their durations. Table 2 defines the equations corresponding the Allen primitives E, S, M, D, O, F and S. The inverse relations are defined in a similar way. *begin*, *end* and *dur* are functions returning respectively the begin time, end time and duration of a given numeric interval. In the following we will present the conversion of the equations defined in table 2 to energy functions.

$I\ E\ J$

$F_E = \sum_{inf_i \le k \le sup_i - d_i} (n_{i,k+d_i} + n_{j,k+d_j} + n_{i,k+d_i} + n_{j,k+d_j} - 4)$

$I\ F\ J$

$F_F = \sum_{inf_j \le k \le sup_j - d_j} (n_{i,k+d_j-d_i} + n_{j,k} + n_{i,k+d_i} + n_{j,k+d_j} - 4)$

$I\ M\ J$

$F_M = \sum_{inf_i \le k \le sup_i - d_i} (n_{i,k+d_i} + n_{j,k} - 2)$

$I\ P\ J$

$F_P = \sum_{1 \le p \le max} \sum_{inf_i \le k \le sup_i - d_i} (n_{i,k+p} + n_{j,k} - 2)$

$I\ D\ J$

$F_D = \sum_{inf_j \le k \le sup_j - d_j} (n_{i,k} + n_{j,k+a} + n_{i,k+d_i+b} + n_{j,k+d_j} - 4) + \sum_{1 \le dur_j - dur_i} \sum_{1 \le dur_j - dur_i} (a + b - dur_i - dur_j)$

Table 2. Definitions of Allen primitives.

Allen Primitive	*Corresponding Equation*
$X\ P\ Y$	$begin(X) + p = begin(Y),\ p > 0$
$X\ E\ Y$	$begin(X) = begin(Y)$ $end(X) = end(Y)$
$X\ M\ Y$	$end(X) = begin(Y)$
$X\ D\ Y$	$begin(Y) + a = begin(X)$ $end(X) + b = end(Y),$ $a + b = dur(Y) - dur(X)$
$X\ O\ Y$	$begin(X) + a = begin(Y)$ $end(X) + b = end(Y)$ $a + dur(Y) = b + dur(X)$
$X\ F\ Y$	$end(X) = end(Y)$ $begin(X) = begin(Y) + dur(Y) - dur(X)$
$X\ S\ Y$	$begin(X) = begin(Y)$ $end(X) + dur(Y) = end(Y) + dur(X)$

$I\ O\ J$

$F_O = \sum_{inf_i \le k \le sup_i - d_i} (n_{i,k+a} + n_{j,k} + n_{i,k+d_i+b} + n_{j,k+d_j} - 4) + \sum_{1 \le dur_i} (a - b + dur_j - dur_i)$

5. Experimentation

In this section, we present comparative tests concerning different approximation algorithms based on local search, namely the Min-Conflict-Random-walk (MCRW), the Steepest-Descent-Random-Walk (SDRW) and the Tabu Search methods; and the method based on DHNN we propose. Tests are performed on consistent and inconsistent temporal constraint problems, each having 200 variables and randomly generated as shown in subsection 5.3. The experiments are performed on a SUN SPARC Ultra 5 station. All the procedures are coded in C/C++.

5.1. *Comparison Criteria*

We use two criteria to compare the different approximation methods. The first one is the quality of the solution, i.e the minimum number of violated constraints of the solution provided by the method. The second criterion is the computing effort needed by an algorithm to find its best solution. This last criterion is measured by the running time in seconds required by each algorithm.

5.2. *Parameter tuning for MCRW, SDRW and Tabu Search*

The performance of the approximation algorithms we use is greatly influenced by the following parameters : the size of the Tabu list, tl_size, in the case of Tabu Search; and the random walk probability, p, in the case of MCRW and SDRW. Preliminary tests determined the following ranges of parameter values : $10 \leq tl\_size \leq 20$ and $0.05 \leq p \leq 0.15$.

5.3. *Generated Instances*

Each generated problem is characterized by two parameters : N the number of events and $Horizon$ the parameter before which all events must be processed. In the following we will describe the generation of consistent and inconsistent problems.

1. Generation of Consistent Problems

Consistent problems of size N are those having at least one complete numeric solution (set of N numeric intervals satisfying all the constraints of the problem). Thus, to generate a consistent problem we first randomly generate a numeric solution and then add other other numeric and symbolic information to it. More precisely the generation is performed using the following steps.

1. Generation of the numeric solution

Randomly pick N pairs (x, y) of integers such that $x < y$ and $x, y \in [0, \ldots, Horizon]$ ($Horizon$ is the parameter before which all events must be processed). This set of N pairs forms the initial solution where each pair corresponds to a time interval.

2. Generation of the numeric constraints

For each interval (x, y) randomly pick an interval contained within $[0..Horizon]$ and containing the interval (x, y). This newly generated interval defines the SOPO of the corresponding variable.

3. Generation of the symbolic constraints

Compute the basic Allen primitives that can hold between each interval pair of the initial solution. Add to each relation a random number in the interval

$[0, Nr]$ $(1 \leq Nr \leq 13)$ of chosen Allen primitives.

Example 2

Let us assume we want to generate a consistent problem with $N = 3$ and $Horizon = 10$.

(1) First a numeric solution is generated: S = $\{(1\ 4), (2\ 8), (5\ 7)\}$.
(2) Numeric constraints (SOPOs) are then randomly generated from the numeric solution.

Numeric Interval		Corresponding SOPO
(1 4)	→	[2 9]
(2 8)	→	[2 10]
(5 7)	→	[3 8]

(3) The Allen primitives are then computed from the pairs of intervals of the numeric solution:

(14) *and* (28)	→	Overlaps (O)
(14) *and* (57)	→	Overlaps (O)
(28) *and* (57)	→	During inverse ($D^{\smile}$)

And finally the symbolic constraints are generated from the above Allen primitives.

O	→	POM
O	→	$DD^{\smile}EO$
$D^{\smile}$	→	$FSDD^{\smile}PE$

5.3.1. *Generation of Inconsistent Problems*

Each inconsistent problem of size N (N is the number of variables) is generated using the following steps.

1. Generation of numeric constraints

Randomly pick N pairs of ordered values (x, y) such that $x, y \in [0, \ldots, Horizon]$. x and y are respectively considered the earliest start time and the latest end time of a given event. For each pair of value (x, y), randomly pick a number $d \in [1 \ldots y - x]$. d is considered the duration of the event.

2. Generation of symbolic constraints

Randomly generate C constraints between the N events where $C \in [1 \ldots \frac{N(N-1)}{2}]$ ($C = \frac{N(N-1)}{2}$ in the case of a complete graph of constraints). Each constraint C is a disjunction of a random number Nb ($Nb \in [1 \ldots 13]$) of relations chosen randomly from the set of the 13 Allen primitives.

3. Consistency check of the generated problem

Perform the branch and bound method on the generated problem. If the quality q of the problem (number of non solved constraints) is equal to zero (the problem is consistent) **goto 1** otherwise ($q > 0$) the problem is inconsistent.

The generated problems are characterized by their tightness, which can be measured, as shown in [16], using the following definition :

> *The tightness of a CSP problem is the fraction of all possible pairs of values from the domain of two variables that are not allowed by the constraint.*

The tightness depends in our case on the parameters $Horizon$ (time before which all tasks should be processed), Nr (the maximal number of Allen primitives per symbolic constraint) and the density of the problem ($\frac{2C}{N(N-1)}$ where C is the number of constraints of the problem).

5.4. *Results*

Table 3 presents the results of the tests performed on randomly generated temporal consistent problems. It gives a summary of the best results of MCRW, SDRW, Tabu Search and the DHNN based method for the chosen instances in terms of quality of the solutions. The results correspond to the average running time and the quality of the solution provided by each method. To obtain these results, the algorithms were run 100 times on each instance, each run being given a maximum of 100,000 moves in the case of MCRW and 10,000 moves in the case of SDRW and Tabu Search. The parameter of each algorithm (the size of the Tabu list tl_size and the random-walk probability p) is fixed according to the best value found during the parametric study. Note that, as mentioned in [12], the cost in time of a move in the case of Tabu Search and SDRW is equal to N times the cost of a move in the case of the MCRW method, where N is the number of variables (events).

Table 3. Comparative results for consistent problems.

Tightness	MCRW				SDRW				Tabu Search				DHNN	
of the problem	p	qual	time	# moves	p	qual	time	# moves	tl_size	qual	time	# moves	qual	time
0.0002	5	0	0.12	5	15	0	2.67	80	10	0	0.17	4	0	10.55
0.0004	5	0	0.28	18	15	0	4.95	136	10	1	185	5000	0	21.25
0.001	5	0	0.46	28	5	0	8.24	193	10	0	0.6	16	0	32.34
0.002	5	0	0.95	68	5	0	11.22	212	10	2	294	10000	0	46
0.0037	5	0	1.74	145	10	0	126	712	10	1	270	10000	0	54
0.006	5	0	4	255	10	0	33	336	10	3	286	10000	0	56
0.03	15	0	86	3713	10	33	33802	10000	10	12	349	10000	0	77
0.044	5	0	73	1633	5	4	9595	10000	15	25	355	10000	0	66
0.045	5	0	72	1633	5	4	9614	10000	10	16	376	10000	0	65
0.058	5	0	15	433	5	74	12333	10000	15	12	364	10000	0	60
0.1	5	0	12	332	5	0	34	225	10	0	112	211	0	33
0.14	5	0	8.47	304	5	0	39	243	10	0	112	193	0	27
0.35	5	0	181	2009	15	0	66	210	20	68	714	10000	0	32
0.44	5	0	137	1291	5	220	8346	10000	10	63	646	10000	0	38
0.55	5	0	315	2505	5	0	66	210	10	0	262	190	0	34
0.67	5	372	13945	100000	5	0	130	297	10	0	422	224	20	112

From the data of Table 3, when comparing the methods based on local search we can make the following observations. For under-constrained and middle-constrained problems, the MCRW method always provides the best results. It always founds a complete solution within a reasonable amount of time which is not the case of the other two methods. It is also faster than the other two methods to find solutions of the same quality. However for over-constrained problems (see last row of table 3) SDRW and Tabu Search have better performance. We can explain this by the fact that, for under constrained problems the initial configuration is in general of good quality. A complete solution can be obtained in this case by only changing the values of some conflicting variables (case of MCRW) instead of looking for the best neighbor which is much more expensive.
When comparing the DHNN based method with the methods based on Local search, we notice that the former one provides better results for over-constrained problems.

Table 4 presents tests performed on randomly generated inconsistent temporal problems. For each instance, the exact method based on branch and bound techniques[11] was first performed in order to get the optimal solution (solution with the minimum number of violated constraints). The three algorithms are then run 100 times on each instance, each run being given a maximum of 100,000 moves in the case of MCRW and 10,000 moves in the case of SDRW and Tabu search. Quality values in boldface correspond to the best quality (optimal solution) found by the exact method.

Table 4. Comparative results for non consistent problems.

Tightness	MCRW				SDRW				Tabu Search				DHNN		B Bound
of the problem	p	qual	time	# moves	p	qual	time	# moves	tl_size	qual	time	# moves	qual	time	qual
0.0002	5	**8**	0.44	32	15	8	4.5	107	10	8	0.28	6	12	120	8
0.001	5	**10**	0.7	53	5	10	10.26	199	10	11	242	5000	15	88	10
0.002	5	**2**	0.68	43	5	3	7.77	183	10	2	194	5000	8	49	2
0.0037	5	**14**	1237	9100	10	14	14.62	238	10	18	230	5000	20	22	14
0.006	5	**20**	5.83	425	10	20	33	336	10	22	377	10000	24	27	20
0.03	15	**21**	190	5406	10	32	3663	10000	10	85	341	10000	25	105	21
0.044	5	**43**	853	25	5	46	4827	10000	15	45	255	10000	43	120	43
0.1	5	**41**	10	318	5	106	41	233	10	91	25	230	41	22	41
0.14	5	**208**	10.14	279	5	208	37	215	10	230	22	197	208	22	208
0.35	5	**141**	259	3015	15	141	439	554	20	141	201	415	141	34	141
0.44	5	**531**	105	271	5	531	82	216	10	531	48	195	531	22	531
0.67	5	**858**	156	315	5	858	98	206	10	924	58	224	858	27	858

From table 4 we can make the same observations we made for table 4 i.e the MCRW method is the algorithm of choice if we have to deal with under-constrained or middle-constrained problems. The effort made by SDRW and Tabu Search methods to look for the best neighbor helps only in the case of over constrained problems. As we can easily see, the DHNN based method is the best one for over-constrained problems. The Branch and Bound method is used here to check the goodness of solution provided by the approximation method.

6. Conclusion

In this paper we have presented an approximation method based on discrete Hopfield neural network (DHNN) for solving problems involving numeric and symbolic temporal constraints. This approximation method has the property to provide a solution with a quality proportional to the allocated running time. This is very relevant since when dealing with these kind of problems in the real world we often look a solution that solves the maximal number of temporal constraints instead of a complete one. This can be the case of over constrained problems or those problems where a solution needs to be found within a given deadline.

In order to evaluate the performance of the DHNN based method we propose, experimental comparison with approximation methods based on randomized local search have been performed. Results show that the DHNN based method presents better results for over constrained problems.

References

1. J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.
2. G. A. Tagliarini, J. F. Christ, and E.W. Page. Optimization using neural networks. *IEEE transactions on computers*, 40:1347–1358, 1991.
3. Kate A. Smith. Neural networks for combinatorial optimization : A review of more than a decade of research. *INFORMS Journal of Computing*, 11(1):15–33, 1999.
4. E. Tsang. *Foundation of Constraint Satisfaction.* Academic Press, 1994.
5. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
6. A. K. Mackworth and E. Freuder. The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.
7. R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.
8. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
9. M. Mouhoub, F. Charpillet, and J.P. Haton. Experimental Analysis of Numeric and Symbolic Constraint Satisfaction Techniques for Temporal Reasoning. *Constraints: An International Journal*, 2:151–164, Kluwer Academic Publishers, 1998.
10. S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.
11. M. Mouhoub. Reasoning about Numeric and Symbolic Time Information. In *the Twelfth IEEE International Conference on Tools with Artificial Intelligence(ICTAI'2000)*, pages 164–172, Vancouver, 2000. IEEE Computer Society.
12. M. Mouhoub. Analysis of Approximation Algorithms for Maximal Temporal Constraint Satisfaction Problems. In *The 2001 International Conference on Artificial Intelligence (IC-AI'2001)*, pages 165–171, Las Vegas, 2001.
13. R. J. Wallace. Partial constraint satisfaction. *Lecture Notes in Computer Science*, 923:121–138, 1995.
14. J.F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.
15. P. Laborie. *Une approche intégrée pour la gestion de ressources et la synthèse de plans.* PhD thesis, École Nationale Supérieure des Télécommunications, 1995.
16. D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proc. 11th ECAI*, pages 125–129, Amsterdam, Holland, 1994.