Stochastic Search versus Genetic Algorithms for Real Time and Over-Constrained Temporal Constraint Problems

Malek Mouhoub
Department of Computer Science
University of Regina
3737 Waskana Parkway,
Regina SK, Canada, S4S 0A2
phone : +1 (306) 585 4700 fax : +1 (306) 585 4745
email : mouhoubm@cs.uregina.ca

ABSTRACT

The aim of this work is to study the applicability of Genetic Algorithms (GAs) and stochastic local search methods to solve real time and over constrained temporal constraint problems. Solving these two type of problems consists of finding a possible scenario (solution) satisfying the temporal constraints within a given deadline. In the case where a complete scenario satisfying the constraints cannot be found, a partial one maximizing the number of solved constraints should be returned. This is an optimization problem where the objective function corresponds to the number of solved constraints. Experimental comparison of genetic algorithms and three stochastic local search methods have been conducted on randomly generated temporal constraint problems. The results of the experimentation favour the Min-Conflicts-Random-Walk (MCRW) local search method for under constrained problems while the GA base method is the technique of choice for middle constrained and over constrained problems.

KEY WORDS
Genetic Algorithms, Local Search, Temporal Reasoning, Constraint Satisfaction.

1 Introduction

Genetic algorithms and local search methods are among the techniques of choice for solving optimization problems. In an optimization problem, the goal consists of finding an optimal solution given some objective function and a list of constraints. These kind of problems are tackled differently depending on the type of the resolution method. Algorithms based on local search, branch and bound or dynamic programming, for example, rely on a single solution as the basis for future exploration with each iteration. Each of these algorithms works on or builds a single solution at a time. Dynamic programming, for example, obtains a complete solution after solving a list of subproblems. A branch and bound algorithm uses a cost function and a lower bound on this function to eliminate some branches of the search space in a systematic manner. Local search methods keep the best solution found so far and try to improve it in the next iteration. In each of the above three techniques, we can use one of the following two rules.

- Deterministic rules : in the case of local search, for example, if an examined neighbor is better, choose that neighbor and continue the search from there.
- Probabilistic rules : we talk here about randomized algorithms where, depending on a random parameter, the next iteration does not necessarily improve the current one (sometimes a weaker neighbor is chosen for the next step). The goal here is to avoid being trapped in a local optima.

On the other hand, genetic algorithms maintain a set (population) of solutions instead of a single one. This has the advantage to allow the competition between solutions of the same population which simulates the natural process of evolution.

Our goal in this paper is to study the applicability of genetic algorithms and stochastic local search methods for managing numeric and symbolic time information. Dealing with these two type of information is very important in many application areas such as planning [1], scheduling [2, 3], computational linguistics [4, 5], database design [6], and computational models for molecular biology [7]. In the case of scheduling problems, for instance, we can have qualitative information such as the ordering between tasks and quantitative information describing the temporal windows of the tasks i.e earliest start time, latest end time and the duration of each task. Thus having a model that handles both types of information is practically important.

In [8] we have defined the model TemPro, based on the interval algebra of Allen [9] and a discrete representation of time, to convert a problem involving a list of numeric and symbolic constraints into a unique constraint-based model that we call Temporal Constraint Satisfaction Problem (TCSP). A TCSP is a binary Constraint Satisfaction Problem (CSP) involving temporal constraints. A binary CSP[10, 11] involves a list of variables defined on finite domains of values and a list of binary relations between variables. Solving a TCSP consists of finding a complete solution (scenario) that satisfies all the temporal constraints. Note that the name Temporal Constraint Satisfaction Problem and its corresponding acronym, TCSP, was used in [12]. The TCSP, as defined by Dechter et al, is a quantitative temporal network used to represent only numeric temporal information. Nodes represent time points while arcs are labeled by a set of disjoint intervals denoting a disjunction of bounded differences between each pair of time points.

In order to deal with real time applications where a solution should be provided within a given deadline or those applications where it is impossible or impractical to solve these problems completely, we may have to look for a partial solution that solves most constraints instead of a complete one. This is an optimization problem where the objective function to maximize corresponds to the number of solved constraints. We will show in this paper how to solve these kind of optimization problems using genetic algorithms and stochastic local search methods.

The rest of the paper is structured as follows. In the next section we present the TCSP representation of temporal constraints using our model TemPro. Section 3 describes the method based on genetic algorithms we use to solve a TCSP. In section 4 we will show the way to solve TCSPs using local search algorithms. Experimental comparison of the different methods we present in this paper for solving randomly generated TCSPs is then presented in section 5. Finally, concluding remarks are presented in section 6.

2 CSP-Based Representation of Temporal Constraints

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 [8], extending the Interval Algebra defined by Allen [9] to handle numeric constraints. TemPro transforms any problem under qualitative and quantitative constraints into a binary CSP where constraints are disjunctions of Allen primitives [9] (see table 1 for the definition of the 13 Allen primitives) and variables, representing temporal events, are defined on domains of time intervals. We call this later a Temporal Constraint Satisfaction Problem (TCSP). 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 quadruple $[earliest\_start,\ latest\_end,\ duration,\ step]$ 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 and $step$ 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[1].

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) :*

[1]This problem is taken from [13]

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

3 Solving TCSPs using Genetic Algorithms

In the previous section we have presented the way to convert temporal constraints into a particular case of CSPs that we call TCSP. In this section we will present an approximation search method based on genetic algorithms (GAs) for solving a TCSP i.e. looking for a scenario (solution) that satisfies all (or most) constraints. Our choice of GAs as a solving method is motivated by the fact that GAs are very efficient for solving CSPs in general as shown in [14].

Let us define the concepts of Genetic Algorithms in general and in the case of TCSPs. Genetic Algorithms perform multi-directional searches by maintaining a population of individuals (called also potential solutions) and encouraging information formation and exchange between these directions. It is an iterative procedure that maintains a constant size population of candidate solutions. Each iteration is called a generation and it undergoes some changes. *Crossover* and *mutation* are the two primary genetic operators that generate or exchange information in GAs. In general, a genetic algorithm for any particular problem must have the five components [15] :

1. a genetic representation for potential solutions to the problem,
2. a way to create an initial population of potential solutions,
3. an evolution function that plays the role of the environment, to evaluate the solutions in terms of their fitness,
4. genetic operators that alter the composition of children during reproduction,
5. and values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, . . . etc).

Under each generation, *good solutions* are expected to be produced and *bad solutions* die. It is the role of the objective (evaluation or fitness) function to distinguish the goodness of the solution.

The idea of crossover operators is to combine the information from parents and to produce a child that obtains the characteristics of its ancestors. For example, parent A (in a binary representation) has a value of 00101000 and parent B has a value of 10010111. If the crossover operator combines the first four bits of parent A with the last four bits of parent B, then the child will obtain the value of 00100111. Crossover operators have a special characteristic that they require two or more inputs in order to perform the crossover. In contrast, mutation is a unary operator that only needs one input. During the process, mutation operators produce a child by selecting some bad genes from the parent and replacing them with the good genes. For example, parent C has a value of 01011101. If the second and third bits are determined to be the bad genes, the mutation operators will replace them with the good genes (they may be the opposite value in the case of binary representation). The child will obtain the value of 00111101. The two operators may behave differently but they both follow the characteristic of GAs in that the next generation is expected to perform better than the ancestors.

In the case of TCSPs, we define the following.

Individual (potential solution) : one possible assignment of numeric intervals to all events i.e set of couples (ev_i, occ_j), where ev_i is an event and occ_j is a possible interval belonging to the domain of ev_i. In other words the individual represents a potential solution to the problem.

Random individual : random assignment of intervals to all events.

Population : a set of individuals (potential solutions).

Mutation : unary operator that returns a new individual (child) by assigning new values (numeric intervals) to some events of a given individual (parent).

Crossover : n-ary operator that takes as arguments two or more individuals and returns a new individual with assignments belonging to parent individuals.

Fitness (evaluation) function : returns a measure of an individual. The measure corresponds here to the quality of the solution. The quality is defined by the number of satisfied constraints.

The pseudo code of the GA based method is illustrated in figure 2. The method starts from a population of p random individuals and iterates until the termination condition is satisfied. The method maintains a population of n individuals, $P(1) = \{ind_1^1, \ldots, ind_n^1\}$ for iteration 1, ... $P(t) = \{ind_1^t, \ldots, ind_n^t\}$ for iteration t, ... etc. Each individual (potential solution) ind_i^t is evaluated using the fitness function. A new population at iteration $t+1$ is then formed by selecting the more fit individuals (*select* step in line 9) from the population of iteration t. Some of the selected individuals will be transformed (*alter* step in line 10) by the mutation and crossover operators. The algorithm is executed until it is running out of time or a solution with the best (or acceptable) quality is found.

Let us illustrate the GA based method on the example we have seen is section 2. Since the problem involves 8 variables, an individual corresponds here to a vector of size 8. Each entry of the vector will contain a possible value of the corresponding variable domain. According to our example, a possible individual can be :

$< AM_1 = (2 \quad 5)$, $AM_2 = (3 \quad 6)$, $AM_3 = (6 \quad 12)$, $BM_1 = (17 \quad 22)$, $BM_{21} = (12 \ \ 14)$, $BM_{22} = (14 \ \ 16)$, $BM_3 = (7 \ \ 14)$, $EVT_1 = (2 \ \ 18) >$

Our method starts with a randomized population of n of these individuals. Each individual will be generated by randomly selecting a value from each variable domain. The next step will be to evaluate each individual to see if we have discovered the optimum solution (satisfying all the constraints of the problem). If not we will apply the *select* function on the individuals of the population. The way we will proceed here consists of assigning a probability of being selected to each individual in proportion of their relative fitness (number of solved constraints). An individual with 10 solved constraints is 10 times more likely to be chosen than an individual with one solved constraint. Since we have to maintain a population of n individuals, we will randomly pick n individuals from the initial population. Note that we may obtain multiple copies of individuals that happened to be chosen more than once (case, for example, of individuals with good fitness function) and some individuals very likely would not be selected at all. Note also that even the individual with the best fitness function might not be selected, just by random chance. After the *select* function, we will perform the *alter* function in which some chosen individuals will recombine using the *crossover* operator or mutate using the *mutation* operator. The new population is then evaluated to perform the *select* and *alter* functions at the next iteration.

4 Solving TCSPs using Local Search Methods

In this section we present the way to solve a TCSP using stochastic local search methods. We will use the following terms:

State : one possible assignment of all events i.e set of couples (ev_i, occ_j), where ev_i is an event and occ_j is a possible interval belonging to the domain of ev_i; the number of states is equal to the product of domains sizes.

State or solution quality : the number of constraint violations of the state or the solution.

Neighbor : the state which is obtained from the current state by changing one event value.

Local-minimum : the state that is not a solution and the evaluation values of all of its neighbors are larger than or equal to the evaluation value of this state.

Strict local-minimum : the state that is not a solution and the quality of all of its neighbors are larger than the evaluation value of this state.

The different algorithms that we will consider in the following are based on a common idea known under the notion of local search. In local search, an initial configuration (assignment of events) is generated randomly and the algorithm moves from the current configuration to a neighborhood configurations until a complete solution or a good one has been found or the resources available are exhausted.

4.1 Min-Conflict-Random-Walk method (MCRW)

After an initial configuration is randomly generated, the Min-conflicts method chooses randomly any conflicting event, i.e., the event that is involved in any unsatisfied constraint, and then picks a value which minimizes the number of violated constraints. If no such value exists, it picks randomly one value that does not increase the number of violated constraints. The problem of this method is that it is not able to escape a local-minimum. In addition, if the algorithm achieves a strict local-minimum it does not perform any move at all and, consequently, it does not terminate. Thus, noise strategies should be introduced. Among them, the random-walk strategy that works as follows : for a given conflicting event, the random-walk strategy picks randomly a value with probability p, and apply the Min Conflict heuristic with probability $1 - p$. In the worst case, the time cost required in each move corresponds to the time needed to determine the value that minimizes the number of violated constraints. This time is of order $O(N\ Max_{1 \leq i \leq N}(\frac{sup_i - inf_i - d_i}{s_i}))$ where N is the number of variables and sup_i, inf_i, s_i and d_i are respectively the latest end time, earliest start time, duration and step of a given event evt_i. Max is the function that returns the maximum of a list of numbers. Figure 3 presents the pseudo-code of the MCRW method for solving TCSPs.

4.2 Steepest-Descent-Random-Walk (SDRW)

In the Steepest-Descent method, instead of selecting the event in conflict randomly as it is the case of MCRW, this algorithm explores the whole neighborhood of the current configuration and selects the best neighbor (neighbor with the best quality). This algorithm can be randomized by using the random-walk strategy in the same manner as for Min-Conflicts to avoid getting stuck at ”local optima”. The time cost required in each iteration corresponds to the time needed to find the best neighbor and is of order $O(N^2 Max_{1 \leq i \leq N}(\frac{sup_i - inf_i - d_i}{s_i}))$ in the worst case. The pseudo-code of the SDRW method is presented in figure 4.

4.3 Tabu-Search (TS)

The pseudo-code of Tabu search method is illustrated in figure 5. This method is based on the notion of Tabu list used to maintain a selective history, composed of previously encountered configurations in order to prevent Tabu from being trapped in short term cycling and allows the search process to go beyond local optima. In each iteration of the algorithm, a couple $< event, intv >$ that does not belong to the Tabu list and corresponding to the best performance is selected and considered as an assignment of the current configuration. $< event, intv >$ will then replace the oldest move in the Tabu list. The time cost required in each iteration is the same as for SDRW, i.e $O(N^2 Max_{1 \leq i \leq N}(\frac{sup_i - inf_i - d_i}{s_i}))$ in the worst case.

5 Experimentation

In order to compare the performance of the GA based method and the three local search techniques, tests were performed on randomly generated consistent and inconsistent temporal constraint problems, each having 200 variables. The goal here is to return a complete solution if the problem is consistent or a partial one maximizing the number of solved constraints otherwise. The experiments were performed on a SUN SPARC Ultra 5 station. All the procedures are coded in C/C++. The advantage of performing the tests on randomly generated TCSPs is that we can control the tightness of the generated problems. The tightness, as defined later in this section, is a measure that tells how constrained is a given problem. This will allow us to generate under constrained, middle constrained and over constrained problems and test the different search methods on each type of problems.

5.1 Comparison Criteria

We use two criteria to compare the different 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 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.

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 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 $[0, \ldots, Horizon]$ 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.

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 the earliest start time and the latest end time of a given event. For each pair of values (x, y), randomly pick a number $d \in [1 \ldots y - x]$. d is 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 constraint graph). 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 a backtrack search method on the generated problem. If a solution is found **goto 1** otherwise 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).

5.3 Results

Table 2 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 GA 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 iterations in the case of MCRW and the GA based method, and 10,000 iterations in the case of SDRW and Tabu Search. Note that, as we mentioned in section 4, 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.

From the data of table 2 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 methods. It is also faster than the other methods to find solutions of the same quality. However, for over-constrained problems 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 (case of SDRW and Tabu) which is much more expensive. When comparing the GA based method with the methods based on local search, we notice that GA provides better results for over-constrained and middle constrained problems and comparable results for under constrained problems. It also obtains, in each case, the solution with the best quality.

Table 3 presents tests performed on randomly generated inconsistent temporal problems. For each instance, an exact method based on branch and bound techniques [18] is 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 iterations in the case of MCRW and the GA based method, and 10,000 iterations in the case of SDRW and Tabu search. From table 3 we can make the same observations we made for table 2 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 GA based method presents comparable and sometimes better results (in the case of over-constrained problems) than MCRW. An exact algorithm based on the branch and bound techniques, we have proposed in [18], is used here to check the goodness of the solution provided by the approximation methods.

6 Conclusion

In this paper we have presented two type of methods for solving numeric and symbolic temporal constraints. The first type is based on genetic algorithms while the second one uses randomized local search techniques. Both methods have 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 for 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 applications where a solution needs to be found within a given deadline. In order to evaluate the performance of the two type of methods, experimental comparison has been performed on randomly generated TCSPs. The results of the tests show that the GA based method is the technique of choice for middle constrained and over constrained problems while the MCRW local search method is the best technique for under constrained problems.

References

[1] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *IJCAI-95*, pages 1643–1649, 1995.

[2] C. Le Pape and S. Smith. Management of Temporal Constraints for Factory Scheduling. In *Temporal Aspects in Information Systems Conference*, pages 165–176, Sophia Antipolis, France, Mai 1987.

[3] P. Baptispte and C. Le Pape. Disjunctive constraints for manufacturing scheduling : Principles and extensions. In *Third International Conference on Computer Integrated Manufacturing*, Singapore, 1995.

[4] F. Song and R. Cohen. Tense interpretation in the context of narrative. In *AAAI'91*, pages 131–136, 1991.

[5] C. Hwang and L. Shubert. Interpreting tense, aspect, and time adverbials: a compositional, unified approach. In *Proceedings of the first International Conference on Temporal Logic, LNAI, vol 827*, pages 237–264, Berlin, 1994.

[6] M. Orgun. On temporal deductive databases. *Computational Intelligence*, 12(2):235–259, 1996.

[7] C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: a graphic-theoretic approach. *Journal of the Association for Computing Machinery*, 40(5):1108–1133, 1993.

[8] 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.

[9] J.F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.

[10] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[11] V. Kumar. Algorithms for Constraint Satisfaction Problems: A survey. *AI Magazine*, 1992.

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

[13] P. Laborie. *Une approche intégrée pour la gestion de ressources et la synthèse de plans*. PhD thesis, École Nationale Supérieure des Télécommunications, 1995.

[14] B. Craenen and A.E. Eiben. Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 7(5):424–444, 2003.

[15] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evaluation Program*. Springer-Verlag, 1992.

[16] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proc. 11th ECAI*, pages 125–129, Amsterdam, Holland, 1994.

[17] 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.

[18] M. Mouhoub. Reasoning with numeric and symbolic time information. *Artificial Intelligence Review*, 21:25–56, 2004.

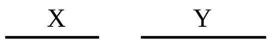
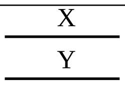
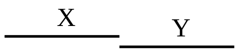
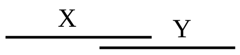
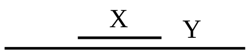
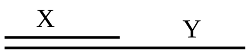
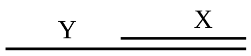
Relation	Symbol	Inverse	Meaning
X precedes Y	P	P-	
X equals Y	E	E	
X meets Y	M	M-	
X overlaps Y	O	O-	
X during Y	D	D-	
X starts Y	S	S-	
X finishes Y	F	F-	

Table 1: Allen Primitives.

Tightness	MCRW			SDRW			Tabu Search			GA		
of the problem	qual	time	# iterations	qual	time	# iterations	qual	time	# iterations	qual	time	# iterations
0.0002	0	0.12	5	0	2.67	80	0	0.17	4	0	5.44	8
0.0004	0	0.28	18	0	4.95	136	1	19.25	10000	0	5.58	22
0.001	0	0.46	28	0	8.24	193	0	0.6	16	0	7.22	44
0.002	0	0.95	68	0	11.22	212	2	294	10000	0	10.8	76
0.0037	0	1.74	145	0	126	712	1	270	10000	0	6.24	287
0.006	0	4	255	0	33	336	3	286	10000	0	12.2	412
0.03	0	86	3713	33	33802	10000	12	349	10000	0	17.18	2612
0.044	0	73	1633	4	9595	10000	25	355	10000	0	38.5	817
0.045	0	72	1633	4	9614	10000	16	376	10000	0	34.2	911
0.058	0	15	433	74	12333	10000	12	364	10000	0	17.22	548
0.1	0	12	332	0	34	225	0	112	211	0	14	784
0.14	0	8.47	304	0	39	243	0	112	193	0	12.5	546
0.35	0	181	2009	0	66	210	68	714	10000	0	34	1255
0.44	0	137	1291	220	8346	10000	63	646	10000	0	38.5	623
0.55	0	315	2505	0	66	210	0	262	190	0	40	711
0.67	372	13945	100000	0	130	297	0	422	224	20	89	7615

Table 2: Comparative results of Tabu Search, MCRW, SDRW and the GA based method for consistent problems.

Tightness	MCRW			SDRW			Tabu Search			GA			B Bound
of the problem	qual	time	# iterations	qual	time	# iterations	qual	time	# iterations	qual	time	# iterations	qual
0.0002	8	0.44	32	8	4.5	107	8	0.28	6	12	13	32	8
0.001	10	0.7	53	10	10.26	199	11	242	10000	15	17	178	10
0.002	2	0.68	43	3	7.77	10000	2	194	5432	8	21	256	2
0.0037	14	1237	9100	14	14.62	238	18	230	10000	20	22	1321	14
0.006	20	5.83	425	20	33	336	22	377	10000	24	15.12	1239	20
0.03	21	190	5406	32	3663	10000	85	341	10000	25	177	4934	21
0.044	43	853	25	46	4827	10000	45	255	10000	43	120	2655	43
0.1	41	10	318	106	41	10000	91	25	10000	41	21	311	41
0.14	208	10.14	279	208	37	215	230	22	10000	208	17	612	208
0.35	141	259	3015	141	439	554	141	201	415	141	34	314	141
0.44	531	105	271	531	82	216	531	48	195	531	21	89	531
0.67	858	156	315	858	98	206	924	58	10000	858	25.5	117	858

Table 3: Comparative results of Tabu Search, MCRW, SDRW and the GA based method for non consistent problems.

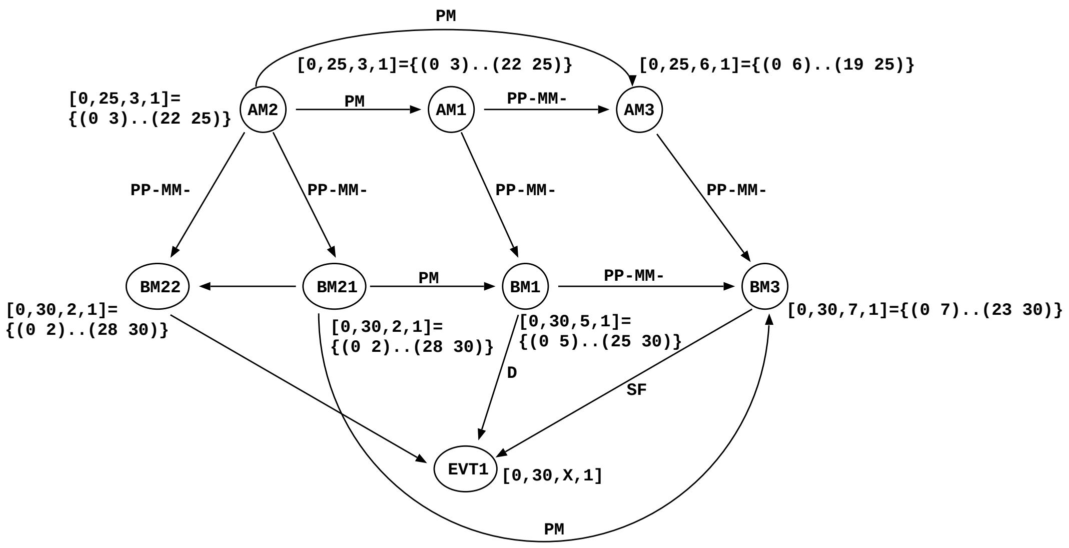


Figure 1: TCSP corresponding to the problem presented in example 1.

```
1.  **begin**
2.      $t \leftarrow 1$
3.      // $P(t)$ denotes a population at iteration $t$
4.      $P(t) \leftarrow n$ randomly generated individuals
5.      $eval \leftarrow\ evaluate\ P(t)$
6.      **while** termination condition is not satisfied **do**
7.          **begin**
8.              $t \leftarrow t + 1$
9.              select $P(t)$ from $P(t-1)$
10.             alter $P(t)$
11.             evaluate $P(t)$
12.         **end**
13. **end**
```

Figure 2: Genetic Algorithm.

```
procedure MCRW(Max_Moves,p)
begin
  s ← random valuation of events;
  nb_moves ← 0;
  while eval(s)$>$ 0 and nb_moves $<$ Max_Moves do
    if probability p verified then
      choose randomly an event evt in conflict;
      choose randomly an interval intv for evt;
    else
      choose randomly an event evt in conflict;
      choose an interval intv that minimizes
      the number of conflicts for evt;
    endif
    if intv $\neq$ current value of evt then
      assign intv to evt;
      nb_moves ← nb_moves+1;
    endif
  endwhile
  return s
end
```

Figure 3: Pseudo-code of the MCRW method.

```
procedure SDRW(Max_Moves,p)
begin
  s $\leftarrow$ random valuation of variables;
  nb_moves $\leftarrow$ 0;
  while eval(s) $>$ 0 and nb_moves $<$ Max_Moves do
    if probability p verified then
      choose randomly a variable evt in conflict;
      choose randomly a value intv for evt;
    else
      choose a move $<$evt,intv$>$ with the best performance
    endif
    if intv $\neq$ current value of evt then
      assign intv to evt;
      nb_moves $\leftarrow$ nb_moves+1;
      endif
  endwhile
  return s
end
```

Figure 4: Pseudo-code of the SDRW method.

```
procedure Tabu-Search(Max_Iter)
begin
    s ← random valuation of variables;
    nb_iter ← 0;
    initialize randomly the tabu list of size tl_size;
    while eval(s) > 0 and nb_iter < Max_Iter do
        choose a move <evt,intv> with the best performance
        among the non-tabu moves;
        remove the oldest move from the tabu list;
        introduce <evt,intv> in the tabu list,
        where intv is the current values of evt;
        assign intv to evt;
        nb_iter ← nb_iter+1;
    endwhile
    return s
end
```

Figure 5: Pseudo-code of the Tabu Search method.