TimeNetManager – A Software Tool for Generating Random Temporal Networks

Amedeo Cesta, Angelo Oddi and Angelo Susi

IP-CNR, National Research Council of Italy
{*cesta, oddi, susi*}*@pst.ip.rm.cnr.it*

Abstract. This paper describes a system, named *TimeNetManager* or *TNM*, that generates and manipulates random constraint networks corresponding to the so called Simple Temporal Problem (STP). The software tool satisfies both the requirements to build some common benchmarks useful to compare different research results, and to create a tool for supporting intensive test of new algorithms for temporal constraints management. The paper gives an overview of the functionalities of the software system and describes in detail the random generator able to fast generate sets of temporal networks controlled by a set of macro-parameters that characterize the topology and the temporal flexibility of the networks.

1 Introduction

The problem of a correct experimental evaluation is crucial in several areas of AI research, in particular when a computationally intensive test of specific algorithms is needed. This paper describes a system, named *TimeNetManager*, or *TNM*, that generates and manipulates random constraint networks corresponding to the so called Simple Temporal Problem (STP) defined in [8]. An STP is a constraint network where nodes represent temporal variables and edges represent temporal duration constraints. All constraints are binary (they concern pairs of variables) and the specification of disjunctive constraints between a pair of variables is not allowed.

Our attention to the problem generates from previous work on the synthesis of specialized dynamic algorithms for managing STPs [2,3]. In that work the need arose for reliable benchmarks, like a library of problems having different characteristics, and some preliminary solutions were given. Such solutions found difficult acceptance for their supposed lack of generality, their limited reproducibility and usability by people different from the authors. In this paper we address the problem of creating a software infrastructure to specifically cope with those limitations. In particular we address the problem of defining a set of random networks according to a number of well designed parameters and their management by a "facilitating" software environment.

When in need of benchmarks two approaches can be followed. A first possibility consists of collecting problems made public by other people (typically on

E. Lamma and P. Mello (Eds.): AI*IA 99, LNAI 1792, pp. 143−154, 2000.

a Web page). This approach, quite common nowadays, is very useful to gather significant examples of real world problems (see for example the "Planning and Scheduling Benchmarks" page [1]). A second possibility is to create random benchmarks. Such a rather different goal, pursued also in this paper, is required in the experimental analysis of algorithms for a well defined research problem, where the need exists of identifying particular structural cases in which one algorithm can be better then another, etc. (see [11, 12] for discussions of several aspects related to the empirical evaluations of algorithms). The generation of random problems is seldom seen in AI as a separate research problem but it is usually described in subparagraphs of papers mostly aimed at presenting technical results about algorithms that solve those random problems. This paper addresses the generation of random STPs as a separate research problem. Such an approach is often followed by the operation research community (see for example [18] for job-shop scheduling or [13] for project scheduling) and is less frequent in AI.

A further question the reader may have is the following "why to address such an effort for the quantitative scheduling problem STP which seems very specific?". The study of the efficiency of STP manipulation is relevant because the management of this temporal problem is the basis of all the approaches to complex scheduling problems [6, 7, 16, 4] not only the ones based on constraint satisfaction. STPs are also concern of attention for the creation of future generation multimedia presentation systems and workflow management tools. It is worth observing that most of the experimental analysis for temporal constraint networks have concerned qualitative constraints (see for example [19, 15]) while no specific work exists for the STPs. It is worth mentioning that some work exists that deals with "general CSP" random generation (see for example [9, 10]).

This paper is organized as follows: Sections 2 shortly introduces the Simple Temporal Problem and the reasoning it requires; Section 3 presents the technical details of the random network generator proposed in this paper; Section 4 describes the general software architecture TNM, its capabilities to offer an experimental workbench and gives some example of randomly generated STPs. Some concluding remarks end the paper.

2 A Basic Definition of STP

The temporal constraint problem named *Simple Temporal Problem (STP)* is defined in [8] and involves a set of integer temporal variables $\{X_1, \ldots, X_n\}$, having convex domains $[lb_i, ub_i]$ and a set of constraints $a \leq X_j - X_i \leq b$, where $b \geq a \geq 0$. A special variable X_0 is added to represent the *Reference-Point* (the beginning of the considered temporal horizon) and its domain is fixed to $[0, 0]$. A solution of the STP is a tuple $(x_1 \ldots x_n)$ such that $x_i \in [lb_i, ub_i]$ and every constraint $a \leq X_j - X_i \leq b$ is satisfied. An STP is *inconsistent* if no solution exists.

[1] `http://www.neosoft.com/~benchmrx/`

A STP can be also represented by a *time-map*. A time-map is defined as a direct graph $TM = \langle V, E, L \rangle$, where V is the set of time points i which represents the variables X_i; E is the set of edges, such that $\langle i, j \rangle \in E$ if the constraint $a \leq X_j - X_i \leq b$ exists; L is a labeling function defined on the set E, such that $L(\langle i, j \rangle) = [a, b]$ for each constraint $a \leq X_j - X_i \leq b$.

2.1 Reasoning Capabilities for an STP

STP networks are used as subcomponents in software systems in several application areas. Such subcomponents can be seen as separate representation and reasoning modules that offer specialized services. An STP stores information about temporal events (time-points) and temporal constraints between events. The possibility may exist of also retracting previous information. In general we can list the main *Tell* and *Ask* primitives of the reasoning service. Among the *Tell* functions of an STP module are the following:

- *Create_time_point* that adds a new time point to the net and links it to the reference-point t_0;
- *Delete_time_point*(i) that deletes the time point specified in the signature of the function;
- *Add_constraint*$(i, j, [a, b])$ that adds the constraint of label $[a, b]$ between time-points i and j;
- *Remove_constraint*$(i, j, [a, b])$ that removes the constraint of label $[a, b]$ between i and j.

Given a certain configuration of a time-map the basic operation is realized by a *propagation function* that computes the domains $[lb_i, ub_i]$ for every time variable i according to the current constraints. The propagation is usually automatically called after any activation of the Tell functions to leave the information on the network in a consistent state (Figure 1 shows an example of time-map where the domains $[lb_i, ub_i]$ have been computed. The boxes labelled "Ref" and "Hor" represent the beginning and end of the temporal horizon).

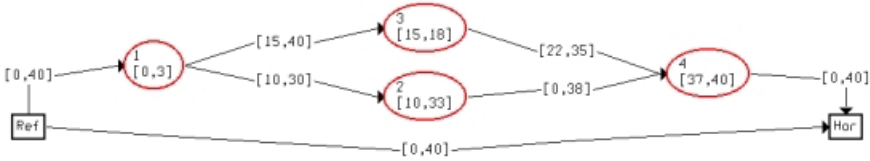

Fig. 1. A propagated time-map

Several *Ask* primitives may be used to extract particular information from a time-map. Two of the more frequent are:

- *time_point_bounds*(i) that returns the bounds associated to a time point i;
- *temporal_distance*(i, j) that returns the interval describing the minimum and maximum distance between the specified time points.

Further primitives can be added for specific tasks according to application requirements. As a service module called into play quite frequently a time-map manager should be rather efficient but addressing this aspect is out of the scope of this paper (for examples of managing algorithms see [8, 2, 3]). We are here interested in two aspects: creating random time-maps as we describe in the next section, and realizing a software environment for easily managing time-maps for different aims as described in Section 4.

3 Generating Random Temporal Networks

In this section we describe the basic ideas followed to generate random temporal networks based on the STP model. The procedure builds temporal networks by using a set of parameters to control the structure of the temporal graphs.

The generation algorithm controls four main graph's characteristics: (1) the number of nodes n, (2) the edges density (also called *connection degree*) D, (3) the *network topology* and (4) the *temporal flexibility*. We believe that by changing these characteristics it is possible to generate a significantly representative subset of all possible STP networks. The proposed procedure is able to generate either a large benchmark set with several levels of difficulty to test the average performance of a reasoning algorithm or to create a particular instance to test such an algorithm in "extreme situations".

The meaning of the first two parameters is straightforward: they determine the number of nodes and edges of the graph. The temporal flexibility is defined as a measure of how the time points can be moved with respect to each other without generating temporal inconsistency. The greater the temporal flexibility, the greater will be the set of the possible solutions of the associated Simple Temporal Problem. As far as the network topology is concerned, we introduce a measure which in practice is the opposite of the definition of *graph diameter* given in [1]. Given a graph G, we consider its undirected version and define the *average minimal diameter* d_m as the average minimal distance between all the couple of nodes in the graph. The distance between two nodes along a path is considered to be the number of path's edges. It is worth noting that d_m is maximal when we have a tree which is a simple sequence of connected nodes and is minimal when we have a complete graph.

The Grid. In order to control both the network topology and the temporal flexibility, we use a reference structure called *grid*. This is a matrix of points with discrete dimensions $T \times P$, (see Figure 2), where the horizontal dimension represents the *time* and the vertical one represents a quantity called *degree of parallelism* which intuitively represents the maximum number of contemporaneous events (time points) in the same time period.

The basic idea is to randomly map the set V of time points on the set of grid points, in this way we fix the average position of all the time points along the time (hence the number of grid points is always greater or equal of the number of graph's nodes n). The random mapping is controlled by two parameters: the *grid ratio* α and the *grid density* β. The first one is the ratio between the dimensions of the grid $\alpha = P/T$ and is one of the basic parameters to control the *average minimal diameter* d_m of the random graph. The second parameters gives the ratio between the number of grid points and the number of nodes $|V|$. In this case the greater β the greater will be the probability of a large scattering of the time points over the time line.

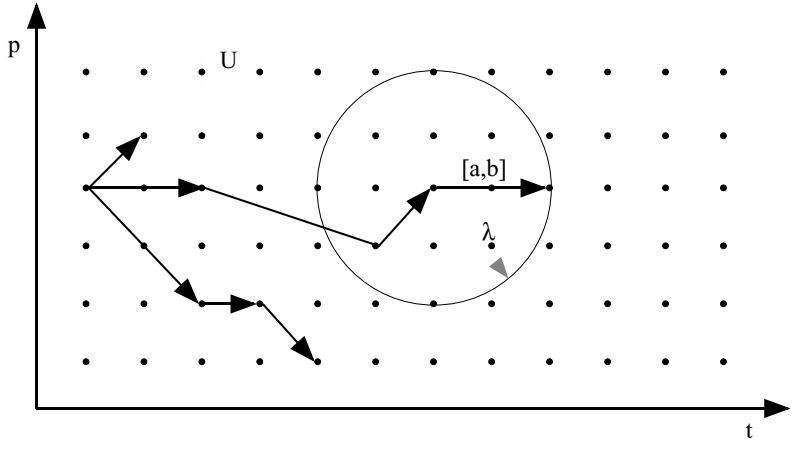

Fig. 2. The grid

The other aspect to control is the temporal flexibility (and also the consistency of the network). As we know, each edge (i, j) represents a temporal constraint between the time points i and j that describes the minimum and the maximum temporal distance between them. The generator randomly produces a set of intervals $[a, b]$ that represents a set of consistent temporal constraints. To this aim we use the metric defined on the grid to generate consistent constraints. That is, we select an interval which contains the distance between i and j computed on the horizontal dimension of the grid. In addition, in order to control the amplitude of the interval (which is directly related to the temporal flexibility) we introduce two real and nonnegative slack parameters S_{min} and S_{max}. The actual generation algorithm can produce only sets of consistent constraints; we are now modifying it for producing also inconsistent time networks.

3.1 The Generation Procedure

The algorithm is shown in Figure 3. It takes as input the parameters n, α, β, D, S_{min}, S_{max} and U and produces a temporal network $TN = \langle V, E, L \rangle$. The algorithm can be divided in three main blocks (or phases): *initialization* (Steps 1-3), *tree generation* (Steps 4-11) and *graph completion* (Steps 12-22).

Initialization. During the *initialization* phase the random mapping is realized between the set of nodes V and the set of points in the grid (Step 1). At Step 2 the neighborhood sets N_i of each time points i are initialized with a set of time points N_i^o. The definition of these sets is related to the grid. In fact, for each node $i \in V$ mapped on the grid, we find a "circular area", with i in the center and having a radius λ, called the *locality*. Here the idea is that it is possible to put only edges (i, j) with $j \in N_i$; λ is defined as the minimum value, such that, the set of nodes contained in the neighborhoods N_i is enough to build a graph of density D. Hence, with regard to the value α and a fixed (and low) value of the edge density D, the smaller the value α, the greater will be the average minimal diameter d_m of the generated graph. Finally at Step 3, a variable F_T (fathers set) is initialized which is used during the *tree generation* phase.

Tree Generation. In order to guarantee the connectivity of the final time map TN, a tree is created and used as basic graph for the completion phase. Steps 4 to 11 implement a quite simple idea. A set F_T (initialized to the origin time point 0), contains the set of nodes currently included in the tree. A father node i and a child node are randomly selected respectively from sets F_T and $N_i \setminus F_T$; hence an edge (i, j) is posted between the nodes i and j with a label $[a, b]$. After that, the node i is included in the set F_T (i becomes a father) and the neighborhoods N_i and N_j are updated. The previous steps continue until a random tree connects the set of nodes V.

It is worth observing the difference between the selection of a child or a father node. The first one is randomly selected with respect to a uniform probability distribution ($Rnd\_Selection()$). The second one is selected by using a dynamic probability distribution ($Tree\_Rnd\_Selection()$) which assigns to each node a variable probability in a way that the more often a point has been involved in a selection as a father the smaller is the probability for it to be selected.

The function $Add\_Constraint(i, j)$ is used to add a labeled edge to the graph. We use two values of time slacks, S_{min} and S_{max}, to generate a random label $[a, b]$ that contains the distance d_{ij} along the grid time line. The distance d_{ij} is computed as $d_{ij} = U|t_j - t_i|$. The edge is always posted according to increasing temporal order. U is an input parameter that gives the number of temporal units per each grid interval and t_j and t_i are the projection of the grid points on the time axis. Hence, the label $[a, b]$ (that contains the distance d_{ij}) are computed as $a = max\{0, \lceil d_{ij}(1 - S_{min}r) \rceil\}$ and $b = \lfloor d_{ij}(1 + S_{max}r) \rfloor$, where r is random number in the interval $[0, 1]$. In the special case of two points selected on the same column of the grid ($d_{ij} = 0$), a specific random function adds either the label $[0, \infty]$ (simple precedence constraint) or the label $[0, 0]$ with equal probability.

INPUT: n, α, β, D, S_{min}, S_{max}, U
OUTPUT: $TN = \langle V, E, L \rangle$

1. $Rnd\_Mapping(V)$
2. $N_i \leftarrow N_i^o$ (for $i = 1 \ldots n$)
3. $F_T \leftarrow \{0\}$
4. **for** $k = 1$ **to** $(n-1)$ **do begin**
5. $\quad i \leftarrow Tree\_Rnd\_Selection(F_T)$
6. $\quad j \leftarrow Rnd\_Selection(N_i \setminus F_T)$
7. $\quad F_T \leftarrow F_T \cup \{j\}$
8. $\quad Add\_Constraint(i, j)$
9. $\quad N_i \leftarrow N_i \setminus \{j\}$
10. $\quad N_j \leftarrow N_j \setminus \{i\}$
11. **end**
12. **while** $(Current\_Density(TN) < D)$ **do begin**
13. $\quad$ **if** $(\exists N_i \neq \emptyset)$
14. $\quad\quad$ **then begin**
15. $\quad\quad\quad i \leftarrow Rnd\_Selection(\{k | N_k \neq \emptyset\})$
16. $\quad\quad\quad j \leftarrow Rnd\_Selection(N_i)$
17. $\quad\quad\quad Add\_Constraint(i, j)$
18. $\quad\quad\quad N_i \leftarrow N_i \setminus \{j\}$
19. $\quad\quad\quad N_j \leftarrow N_j \setminus \{i\}$
20. $\quad\quad$ **end**
21. $\quad$ **else** $N_i \leftarrow Ns\_Updating(N_i)$ (for $i = 1 \ldots n$)
22. **end**

Fig. 3. Generation Algorithm

Graph Completion. In the last phase other edges are added to the graph until the density D is reached. The algorithm proceeds similarly to the tree generation phase with two notable differences. First, we use only uniform probability distribution to make random selections. Second, we have to consider the possibility that all the neighborhood sets became empty and at the same time the value of edges density D is not reached. This is possible, because at Step 2 we generate the sets N_i^o with respect to an estimation of the locality λ. At Step 13 we check if at least a neighborhood is not empty, if not we update the locality to a larger value such that at least one neighborhood N_i becomes not empty (Step 21); otherwise we insert edges as previously explained.

4 The TNM Software Architecture

To exploit the capabilities of the time-map generator we have inserted it in a software tool that allows a flexible interaction between a user, an STP reasoning system and the random generator. The software environment TNM has been designed as a building block for two main goals:

- experimental comparative testing of algorithms;
- intensive debugging of time-map reasoning systems.

It turns out that the system we have built is very useful to be used also as a didactic tool to describe a quantitative constraint manager based on STP.

The architecture of TNM is shown in Figure 4. We can see the three basic modules: (1) the TN Generator (TN-G) that is responsible for creating random networks according to the previous description; (2) a TN Representation and Reasoning Module (TN-RR) that gives the basic service of constraint management; (3) a TN Interaction Module that allows a user to interact and use all the features of the two basic modules. In the figure dashed lines remark that two components can be substituted by or integrated with others. In particular the software architecture is designed to allow the use of different TN-RR modules. We have built the TNM system for the aim of testing the efficiency of the time-map manager of the scheduling architecture O-OSCAR [5] but the careful object-oriented design allows the easy integration of a different representation module and/or of new management algorithms. A further aspect considered as "TNM independent" is the visualization of the time-map. As well known graph drawing is a research topic *per se*, so we are using for this task a public domain system VCG [17] [2] that guarantees quite an amount of useful features. The whole TNM is implemented in C++, the graphical user interface has been developed using the Amulet toolkit [14] [3].

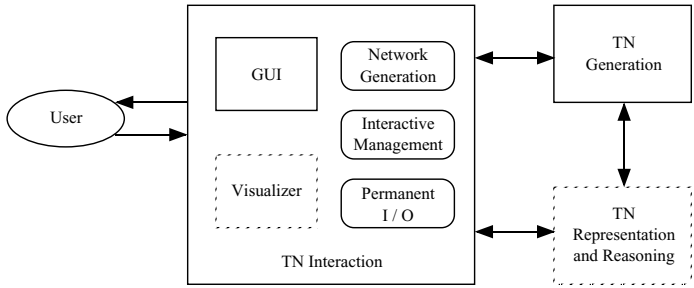

Fig. 4. Overview of the software architecture

4.1 The TNM Interaction Services

As depicted in Figure 4 the TNM Interaction Module (TN-IM) is composed by a graphical user interface (GUI) that allows to call the visualizer and to interact with three specific groups of functionalities:

[2] `http://www.cs.uni-sb.de/RW/users/sander/html/gsvcg1.html`
[3] `http://www.cs.cmu.edu/afs/cs/project/amulet/www/amulet-home.html`

- *Network Generation* is the modality that allows to interact with TN-G to produce sets of random networks giving in input a set of appropriate parameters as previously described;
- *Interaction Manager* allows the user to directly manipulate a time-map executing single Tell or Ask functions towards the TN-RR. In this way it is possible to either modify a pre-existent random network, or specify a network step by step from scratch. This functionality is particularly useful while debugging new reasoning modules because it allows the user to execute particular operations on the network after inspecting the visualization of the network for example.
- *Permanent I/O* allows to load and save in several file formats the networks and their characteristics. The possibility is included to produce files readable from a graph visualizer. This modality is deeply used to reproduce experiments because allows to save whole networks, the random seeds, etc.

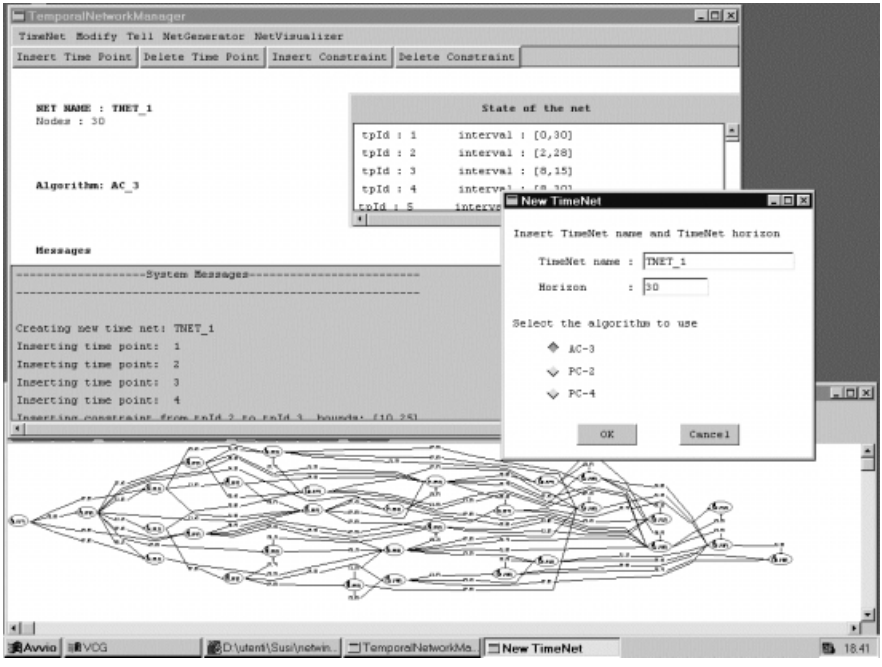

Fig. 5. Typical screen of TNM in operation

Figure 5 shows a dump screen of TNM at work. It is possible to recognize: the general menu bar that allow the selection of basic functionalities and/or interaction modalities; a specialized menu-bar of the Interaction Manager that allows to perform the basic Tell functions specified above on a give time-map (visual-

ized in the background). It is to be noted the constant possibility of naming and saving a current network, of choosing particular propagation algorithms to be used, of inspecting single aspects of the active network, etc.

4.2 Using the Random Network Generation

We now show examples of the ability of TN-G at creating time-maps of different shapes. In Figure 6 we have omitted from the drawing the information of distances because we aim at showing the possibility of producing different topologies.

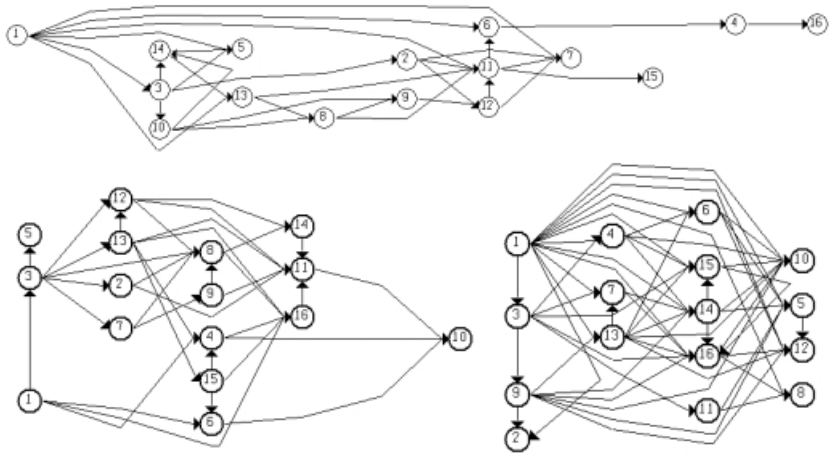

Fig. 6. Examples of generated random networks

The graph in the first row of the figure is an example of network "almost serial and loosely connected" and has been produced with the parameters $\langle \alpha = 0.33, \beta = 3.5, D = 0.18 \rangle$. In the left of the second row a graph is shown "parallel and loosely connected" with the parameters $\langle \alpha = 1, \beta = 3.5, D = 0.18 \rangle$. In the right side of the second row a graph "parallel and strongly connected" is shown (generation parameters: $\langle \alpha = 1, \beta = 4, D = 0.4 \rangle$). All the networks in the examples have 16 time-points.

It is not possible here to show the effectiveness of the work performed by the generation algorithm in deciding random durations, we just report the fact that first experimentation we are performing is quite satisfactory.

It is worth noting that TN-G opens the possibility to create a number of random network with similar parameters and all different on a random basis, and also the possibility of creating benchmarks with and equilibrate presence of networks of different shape.

By using a TN-IM dialogue window it is possible to ask for the creation of a set of random problems within certain generation parameters, set a name to each of them and store them on separate directories. It is alternatively possible to create a random temporal network for immediate use in the TN-RR and the subsequent store of it.

5 Conclusions

This paper has addressed the problem of producing random STP networks for experimental analysis of STP reasoning functionalities. This is motivated from one side by the need to build some common benchmarks to compare different results as in other research fields like scheduling; from the other by the need to produce tools for doing tests during the development of new algorithms for temporal constraints managing. We have developed a software tool called *Temporal Networks Manager* (TNM) that can satisfies both this requirements. The system has a temporal network generator that allows the user to fast generate sets of random temporal networks controlling this process through a set of macroscopic parameters that characterize the topology and the temporal flexibility of the networks. This way is possible to produce collections of networks for testing the algorithms used in the temporal representation managing. The system also allows the user to manually modify these networks or to produce from scratch other networks with particular properties, useful to tests the algorithms in critical situations. It is possible to interact with the system trough graphical user interface that allows to load, save, generate and edit the networks. The system has a modular design and this made easy to expand or modify single parts of it. In particular it is possible to redefine the temporal representation module allowing the user to put its own representation of temporal domains in the system only respecting the predefined interface between modules.

Acknowledgments

Thanks to the AI*IA reviewers for their detailed comments. This research has been developed in the framework of the project "A toolkit for the synthesis of interactive planners for complex space systems" supported by ASI (Italian Space Agency).

References

1. B. Bollobás. *Random Graphs*. Academic Press, 1985.
2. R. Cervoni, A. Cesta, and A. Oddi. Managing Dynamic Temporal Constraint Networks. In *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS-94)*, 1994.
3. A. Cesta and A. Oddi. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*, 1996.

4. A. Cesta, A. Oddi, and S.F. Smith. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the 16th Int. Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.
5. A. Cesta, A. Oddi, and A. Susi. O-OSCAR: A Flexible Object-Oriented Architecture for Schedule Management in Space Applications. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-99)*, 1999.
6. C. Cheng and S.F. Smith. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on AI (AAAI-94)*, 1994.
7. B. De Reyck, E. Demeulemeester, and W. Herroelen. Algorithms for Scheduling Projects with Generalized Precedence Relations. In J. Weglarz, editor, *Handbook on Recent Advances in Project Scheduling*. Kluwer, 1998.
8. R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.
9. I.P. Gent, A.S. Grant, E. MacIntyre, P. Prosser, P. Shaw, B. Smith, and T. Walsh. How Not To Do It. Technical Report 97.27, University of Leeds, School of Computer Studies, 1997.
10. I.P. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random Constraint Satisfaction: Flaws and Structure. Technical Report 98.23, University of Leeds, School of Computer Studies, 1998.
11. J.N. Hooker. Needed: An Empirical Science of Algorithms. *Operations Research*, 42:201–212, 1994.
12. J.N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1:33–42, 1996.
13. R. Kolish, A. Sprecher, and A. Drexl. A Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science*, 41:1693–1703, 1995.
14. B. A. Myers, R. G. McDaniel, R. C. Miller, A. Ferrency, A. Faulring, B. D. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, 23:347–365, 1997.
15. B. Nebel. Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class. *Constraints*, 1(3):175–190, 1997.
16. W.P.M. Nuijten and C. Le Pape. Constraint-Based Job Shop Scheduling with ILOG-SCHEDULER. *Journal of Heuristics*, 3:271–286, 1998.
17. G. Sander. Graph Layout through the VCG Tool. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing, DIMACS International Workshop GD'94, Proceedings*, pages 194–205. Lecture Notes in Computer Science 894, Springer Verlag, 1995.
18. E. Taillard. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research*, 64:278–285, 1993.
19. P. van Beek and D. W. Manchak. The Design and Experimental Analysis of Algorithms for Temporal Reasoning. *Journal of Artificial Intelligence Research*, 4:1–18, 1996.