A System Prototype for Solving Multi-granularity Temporal CSP[⋆]

Claudio Bettini, Sergio Mascetti, and Vincenzo Pupillo

DICo – Università di Milano, Italy

Abstract. Time granularity constraint reasoning is likely to have a relevant role in emerging applications like GIS, time management in the Web and Personal Information Management applications for mobile systems. This paper reports recent advances in the development of a system for solving temporal constraint satisfaction problems where distance constraints are specified in terms of arbitrary time granularities.

1 Introduction

When variables in a constraint satisfaction problem (CSP) are used to represent event occurrences and constraints to represent their temporal relations, a CSP is called *temporal* CSP or TCSP. Scheduling, planning, diagnosis, natural language understanding, and even temporal databases are examples of areas where temporal CSP's have been applied.

In some cases, a temporal CSP can be formulated in terms of qualitative temporal relations between events, like "event1 must occur *before* event2" or "event1 must occur *immediately after* event2", while in other cases quantitative temporal relations are necessary, like "event2 must occur *at least 1 time unit* and *at most 5 time units after* event1". The many formalisms and algorithms proposed in the literature for TCSP have essentially ignored the subtleties involved in the presence of multiple time units (granularities) in the temporal constraints. Examples of simple constraints specified in terms of a time granularity are the following: "package shipment must occur *the next business day* after check clearance" and "package delivery should occur *during working hours*". There are several emerging applications like GIS, time management in the Web, and Personal Information Management for mobile systems that could greatly benefit of algorithms and systems for time granularity constraint reasoning.

In the last years we have been investigating the concept of time granularity and multi-granularity TCSP, focusing on GSTP, the extension of STP [7] to multiple and arbitrary granularities. Technically, in a GSTP, binary constraints have the form $Y - X \in [m, n]\, G$, where m and n are the minimum and maximum values of the distance between X and Y in terms of time granularity G. Variables take values in the positive integers, and unary constraints can be applied on their domains.

[⋆] This work has been partially supported by Italian MIUR (FIRB "Web-Minds" project N. RBNE01WEJT_005).

A first issue in the representation and processing of these constraints is the need for a clear semantics for time granularities. *Business days*, for example, may really have different meanings in different countries or even in different companies. In this respect GSTP adopts a formalism, first introduced in [3], which can model arbitrary user-defined time granularities and has a clear set-theoretic semantics. In order to guarantee a finite representation, granularities in GSTP are limited to those that can be defined in terms of periodic sets. Hours, days, weeks, business days, business weeks, fiscal years, and academic semesters are common examples.

A second issue is related to the difficulty to reduce a network of constraints given in terms of different granularities into an equivalent one with all constraints in terms of the same granularity, so that some of the standard algorithms for CSP could be successfully applied. Indeed, any conversion necessarily introduces an approximation; For example, a constraint imposing delivery to start the next business day may be translated in terms of hours with a minimum of 1 hour and a maximum of 95 hours[1]. However, if the check is cleared on Monday, the constraint in hours would allow a shipment on Thursday which is clearly a violation of the original constraint. Approximate conversion algorithms are extensively discussed in [3, 4]. We have shown that any consistency algorithm adopting these conversions as the only tool to reduce the problem to a standard CSP is inevitably incomplete, and have proposed a different algorithm, called AC-G, which has been proved to be complete [5].

A prototype system, named GSTP, has been developed at the University of Milan with the objective of providing universal access to the implementation of a set of algorithms for multi-granularity temporal constraint satisfaction. GSTP, in addition to implementing the reasoning algorithms, assists the user in the definition of constraint networks, in their submission to a remote processing service and in the analysis of the output. A rich pre-defined set of time granularities is available, but new user-defined granularities can be added, and they will be handled by the constraint solving algorithms. The GSTP system has been publicly shown at the Intelligent Systems Demonstration venue at IJCAI 2003 [2]. While the prototype is mainly based on algorithms presented in [5], several enhancements, both theoretical and at the implementation level have been studied and applied. This paper illustrates the overall architecture and essential features of the system, reports recent enhancements and preliminary experimental results.

An extensive literature exists on CSP and TCSP problems. The most popular techniques to deal with CSP are arc- and path-consistency. Several versions of these algorithms have been proposed [8, 1]. These algorithms are not specific for temporal constraints and usually assume a finite domain; extensions to deal with infinite domains and TCSP have also been studied [7], but they do not deal with periodic sets. Recent work has been done on identifying tractable classes of periodic CSP [6], but we do not see any immediate applicability of those results to the problems addressed by GSTP. We are also not aware of any CSP

[1] The number 95 takes into account a check clearance at the beginning of a Friday and a shipment at the end of next Monday according to the constraint.

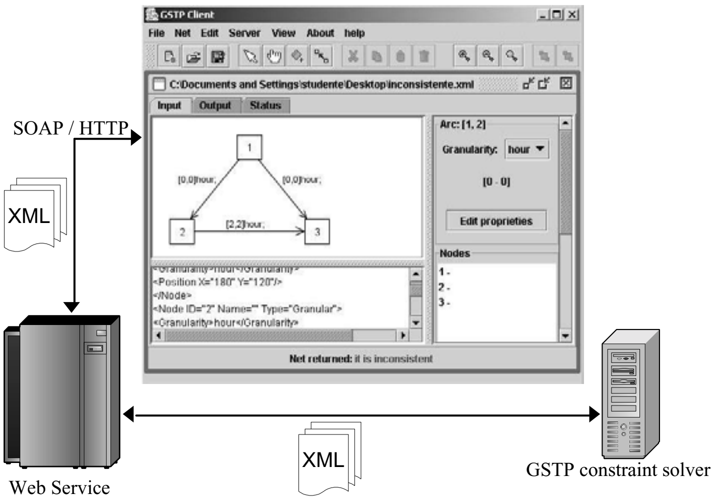
system that can directly handle a GSTP problem. Related ideas can be found in [9], where authors note that AC-3 can be generalized to deal with intensionally described domains and constraints, provided that the *domain restriction operation*, usually contained in the REVISE procedure of AC-3, can be performed on the intensional descriptions. Indeed, the AC-G algorithm proposed in our paper can be considered an extension of the AC-3 algorithm to deal with possibly infinite (but periodic) domains and with binary temporal constraints in terms of multiple periodic granularities.

The rest of the paper is organized as follows: In the next section we briefly present the architecture of the GSTP system, and describe two of the three components, the Web Service and a client graphical interface. In Section 3 we present the third component, the constraint solver, illustrating the main algorithm and some recent enhancements. In Section 4 we briefly discuss implementation issues and experimental results, and in Section 5 we conclude the paper.

2 The GSTP Architecture

Figure 1 shows the general architecture of the GSTP system. There are three main modules: the constraint solver, the web service, which enables external access to the solver, and a web service client user interface that can be used locally or remotely to design and analyze constraint networks. All data, including time granularity definitions, constraint network specification, algorithm parameters and processing requests are encoded in XML following specific XML schemas.

The GSTP Constraint Solver is clearly the most complex and innovative component, and the one which required the most implementation efforts; It is described separately in Section 3. Here we give a brief introduction to the main functionalities of the Web Service and of the Client Interface.



```
<Position X="180" Y="120"/>
</Node>
<Node ID="2" Name="" Type="Granular">
<Granularity>hour</Granularity>
```

Net returned: it is inconsistent

Fig. 1. The GSTP Architecture.

2.1 The GSTP Web Service

The Web Service defines, through a WSDL specification, the parameters that can be passed to the constraint solver, including the XML schema for the constraint network specification. The service is exposed to the public web, and despite we provide a specific client application, it can be invoked by different clients or web applications. Therefore, in principle, our service can be easily integrated in any third party software which requires GSTP processing.

The web service application performs three tasks: first of all it validates the parameters by checking if the XML is valid with respect to the XML schema and if the names of the granularities used are already defined. Then it invokes the solver and finally it passes back the results in XML format.

A single service is currently supported even if a number of parameters can be used to specify different versions of the constraint solver algorithms. It is possible, for example, to give up completeness by selecting a variant of the main algorithm in order to have much lower response time, or to use the complete version and possibly set time-out values different from the default ones.

2.2 The GSTP Client

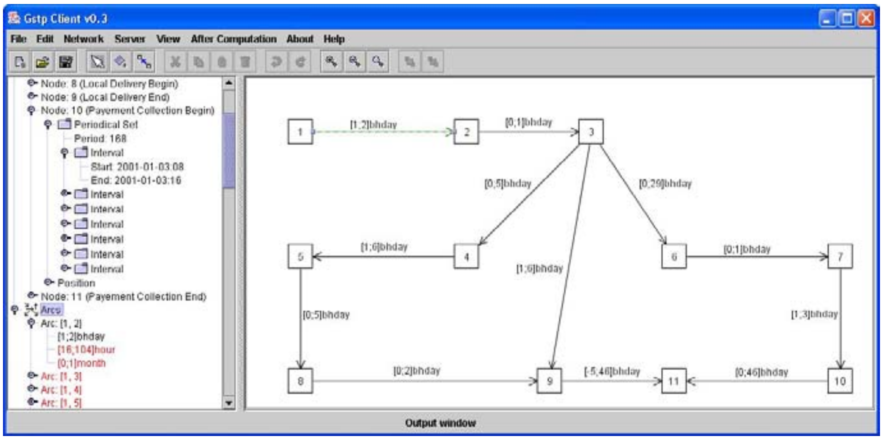
The main goal of the client interface, in addition to remotely interact with the constraint solver through the web service, is to facilitate two tasks: i) the specification and editing of input networks, and ii) the analysis of processed networks.

For the former task, the GSTP Client supports the user by providing standard functionalities like adding, editing and removing nodes or edges. Networks can also be saved and browsed in XML format.

For the latter, more specialized tools have been developed. In fact the result of the GSTP Constraint Solver is a fully connected network having each arc possibly labeled by one constraint for each of the granularities appearing in the input network. It is clear that is practically infeasible to graphically show all this information in a single screen-shot in a way that is still useful to the user. Therefore some functionalities have been introduced: first of all zooming and scrolling features allow to examine large networks, while nodes can be automatically disposed in order not to overlap with each other or to preserve the position they had in the input network. Moreover it is possible to selectively hide and show information from the network: in particular it is possible to have views of the network in terms of specific single or set of time granularities.

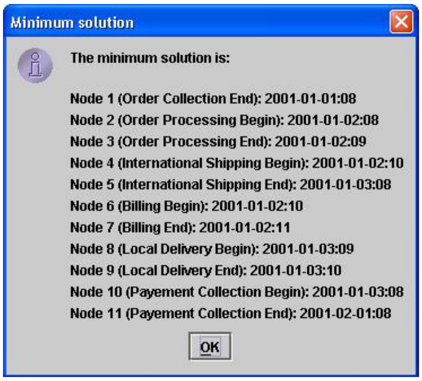
Figure 2 shows the GSTP Client interface showing the result of a GSTP constraint solver computation. The nodes are disposed as they were in the input network, and the only edges that are shown are the ones that were explicit in the input network. All the constraints are hidden except those in terms of the granularity `bhday` (the business hour day, i.e., the working hours during the business days).

A specific functionality has been introduced in order to show a network solution if the network is found to be consistent (see Figure 3).



```
Node: 8 (Local Delivery Begin)
Node: 9 (Local Delivery End)
Node: 10 (Payement Collection Begin)
  Periodical Set
    Period: 168
    Interval
      Start: 2001-01-03:08
      End: 2001-01-03:16
    Interval
    Interval
    Interval
    Interval
    Interval
    Interval
  Position
Node: 11 (Payement Collection End)
Arcs
  Arc: [1, 2]
    [1;2]bhday
    [16,104]hour
    [0;1]month
  Arc: [1, 3]
  Arc: [1, 4]
  Arc: [1, 5]
```

Fig. 2. The view of a processed network in terms of a specific time granularity.



Minimum solution

The minimum solution is:

Node 1 (Order Collection End): 2001-01-01:08
Node 2 (Order Processing Begin): 2001-01-02:08
Node 3 (Order Processing End): 2001-01-02:09
Node 4 (International Shipping Begin): 2001-01-02:10
Node 5 (International Shipping End): 2001-01-03:08
Node 6 (Billing Begin): 2001-01-02:10
Node 7 (Billing End): 2001-01-02:11
Node 8 (Local Delivery Begin): 2001-01-03:09
Node 9 (Local Delivery End): 2001-01-03:10
Node 10 (Payement Collection Begin): 2001-01-03:08
Node 11 (Payement Collection End): 2001-02-01:08

Fig. 3. The minimum solution found by the solver is displayed by the GSTP client interface.

3 The Constraint Solver

The algorithmic task of the constraint solver is to decide the consistency of a set of granularity constraints, to find a solution if one exists, and to restrict the constraints as much as possible while preserving the same set of solutions. Standard algorithms to solve TCSP cannot be easily adapted to GSTP. In order to understand the exact semantics of these networks we first report a few basic notions and then describe our strategy for constraint solving.

3.1 Basic Notions

For lack of space we define informally time granularities referring the interested reader to [3] for formal definitions. A granularity is intuitively seen as a particular grouping of instants from a time domain. Each group is called a *granule*. In most application domains we can find a *bottom* granularity, with the property of being sufficiently fine grained so that no further refinement is needed to represent data in the application domain, and that all other granularities can be represented as groupings of granules of this one. For example, if `day` is the bottom granularity, `week` can be defined by grouping 7 days. More complex groupings are needed to represent `month`, `academic-semester`, or `business-week`, but they can all be represented by a periodic expression whose primitive elements are the granules of the bottom granularity. For practical reasons, we index granules with positive integers, so that periodic expressions are in terms of positive integers and algebraic operations can be quite easily defined on granularities. As we will seen in the following, the domains of variables in our constraint networks are indeed indexes of granules of the bottom granularity, and hence, a network solution is the assignment of a specific granule to each variable.

In the following, the function $\lceil t_1 \rceil^G$ denotes the index of the granule of granularity G containing the granule of the bottom granularity indexed by t_1. For example, $\lceil 32 \rceil^{\texttt{month}} = 2$, i.e., the day indexed by 32 falls in the month indexed by 2, if we assume that the bottom granularity is `day`, the first `day` is January 1st and the 1st month is January. Similarly, the function $\lfloor j \rfloor^G$ denotes the set of indexes of granules of the bottom granularity forming the j-th granule of G.

Definition 1. *Let $m, n \in \mathbb{Z} \cup \{-\infty, +\infty\}$ with $m \leq n$ and G a granularity. Then $[m, n]\, G$, called a* temporal constraint with granularity *(TCG), is the binary relation on positive integers defined as follows: For positive integers t_1 and t_2, $(t_1, t_2) \models [m, n]\, G$ ((t_1, t_2)* satisfies *$[m, n]\, G$) if and only if (1) $\lceil t_1 \rceil^G$ and $\lceil t_2 \rceil^G$ are both defined, and (2) $m \leq (\lceil t_2 \rceil^G - \lceil t_1 \rceil^G) \leq n$.*

Intuitively, for instants t_1 and t_2 of granules of the bottom granularity, if t_1' and t_2' are the indexes of the granules of G containing t_1 and t_2, respectively (i.e., $t_1' = \lceil t_1 \rceil^G$ and $t_2' = \lceil t_2 \rceil^G$), t_1 and t_2 satisfy $[m, n]\, G$ if the difference of t_2' and t_1' is between m and n (inclusively). That is, the instants t_1 and t_2 are first translated in terms of G, and then the difference is taken. If the difference is at least m and at most n, then the pair of instants is said to satisfy the constraint. For example, if `day` is the bottom granularity and t_1 and t_2 denote two specific days, the pair (t_1, t_2) satisfies $[0, 0]$ `week` if the days denoted by t_1 and t_2 are within the same week. Similarly, (t_1, t_2) satisfies $[-1, 1]$ `month` if those two days are at most one month apart (and the order of them is immaterial). Finally, (t_1, t_2) satisfies $[1, 1]$ `year` if the day denoted by t_2 is in the next year with respect to the one denoted by t_1.

Definition 2. *A* constraint network (with granularities) *is a directed graph (W, A, Γ, Dom), where W is a finite set of variables, $A \subseteq W \times W$ a set of arcs, Γ is a mapping from A to the finite sets of temporal constraints with granularities, and Dom is a mapping from W to possibly bounded periodical subsets of the positive integers (indexes of the bottom granularity).*

Note that in these networks multiple constraints (in terms of different time granularities) can be associated with the same arc. From the results in [3], where temporal constraints with granularities were first defined, it follows that, even if constraints on the domains are excluded, and a single TCG is associated with each arc, the consistency problem is NP-hard when arbitrary periodic granularities are allowed, while the single-granularity problem is in PTIME [7].

3.2 A Strategy to Solve GSTP

Constraint satisfaction in GSTP is based on the implementation and optimization of algorithms presented in recent papers. An extension to standard path-consistency based on approximate conversions among constraints with granularities has been proposed in [3]. However, the algorithm was shown to be incomplete with respect to consistency. A sound and complete consistency algorithm, called AC-G (Arc-Consistency with Granularities), has been recently proposed in [5]. On the other side, this algorithm does not directly help in the refinement of constraints and it can greatly benefit from using the previous algorithm as a preprocessing step.

Hence, the solution adopted for the GSTP constraint solver is shown in Fig. 4. In step 1, the original network is decomposed in as many networks as are the granularities appearing in the constraints; each network has the explicit constraints given in terms of one granularity as well as constraints in the same granularity obtained by conversion from others on the same arc, but in terms of different granularities. Then, standard path consistency is applied to each network; networks are re-merged in a single one and if any refinement occurred a new conversion step followed by path consistency is performed, until a fix-point is reached. The resulting network most likely has refined constraints with respect to the original one. Any inconsistency captured by this processing has the effect of terminating the constraint solver reporting the inconsistency status. However, if this is not the case, the network may still be inconsistent and it will go through AC-G (step 2) which is guaranteed to detect an inconsistency if one exists and was not detected by the previous step. From the node domains returned by AC-G, it is possible to further refine some of the constraints (the function doing this job in step 3 is called *RefineArcsFromNodes()*). The steps are repeated, since path consistency applied to the refined constraints may lead to some changes. Correctness and termination have been proved.

Conversion is not a trivial task. A simple example has been given in the introduction considering the conversion of the constraint $[1,1]$`b-day` in terms of `hours`. Since, in general, it is not possible to replace a TCG with an equivalent one in terms of a different granularity, our goal is to find a TCG which is the

```
Repeat
        1. Repeat Conversion+PC Until no change is observed
        2. AC-G
        3. RefineArcsFromNodes()
Until no change is observed
Return Inconsistent or NewNetwork+solution
```

Fig. 4. The main loop of the constraint solver.

```
INPUT: a network $\mathcal{N} = \langle W, A, \Gamma, Dom \rangle$.
OUTPUT: a network $\mathcal{N}' = \langle W, A, \Gamma, Dom' \rangle$ equivalent to $\mathcal{N}$ and
having one of the domains empty if inconsistent.
METHOD:
$Q := \{(X_i, X_j) \mid (X_i, X_j) \in A\}$
while $Q \neq \emptyset$ do
  1. select and delete an arc $(X_l, X_k)$ from $Q$
  2. if $Dom(X_l) \neq^{MAX} Dom(X_l) \cap (Dom(X_k) \uplus \Gamma(X_k, X_l))$ then
     2.1. $Q := Q \cup \{(X_i, X_l) \mid (X_i, X_l) \in A,\ i \neq k\}$
     2.2. $Dom(X_l) := Dom(X_l) \cap (Dom(X_k) \uplus \Gamma(X_k, X_l))$
  3. if $Dom(X_l) =^{MAX} \emptyset$ then $Q := \emptyset$; $Dom(X_l) := \emptyset$
end while
```

Fig. 5. The AC-G algorithm.

tightest among those in terms of the new granularity that are implied by the network for the same arc[2]. With respect to previously published algorithms for granularity constraint conversion [3], in the current implementation of GSTP, we use a new algorithm described in [4] that has been proved to derive the tightest converted constraints, and indeed leads to a more effective preprocessing for the constraint solver.

3.3 The AC-G Algorithm

The most challenging part of the system is perhaps the implementation of is algorithm, called AC-G. It is based on arc-consistency, and it is essentially an extension of the AC-3 algorithm [8] to deal with possibly infinite (but periodic) domains and with constraints in terms of multiple periodic granularities. This extension is not trivial since it involves the algebraic manipulation of the mathematical characterization of granularities. AC-G also derives the *minimal* solution for the constraint network.

AC-G is in general exponential in the number of granularities involved in the network, but can be considered to take polynomial time when the time granularities in the constraints are known by the system on which the algorithm is run (i.e., the description of granularities is not given as part of the CSP). Note that most practical applications can satisfy this condition.

A sketch of the AC-G algorithm is reported in Figure 5. Without loss of generality, we assume that for each TCG $[m, n]\, G$ on arc (X_l, X_k), the TCG $[-n, -m]\, G$ exists on arc (X_k, X_l). Basically, the algorithm non-deterministically selects and deletes an arc (X_l, X_k) from a queue (Q) that initially consists of all the arcs, and uses the domain for X_k and the constraints between X_k and X_l to restrict the domain of X_l. If it is restricted, the queue is updated so that any arcs that could lead to further restrictions are re-inserted. Eventually, a fix-point will be reached and the queue will become empty. Except for the presence

[2] By logically implied we intuitively mean that when the TCG is associated with the same arc as the source TCG, the solutions for the original network are still solutions.

of granularity constraints, this is a classical arc-consistency algorithm, in the version known as AC-3 [8]. The central issue in the algorithm is how domains can be restricted considering the granularity constraints associated with the arcs. This is achieved by the operation $Dom(X_k) \uplus \Gamma(X_k, X_l)$ that is defined as returning the set $\{t_l \mid \exists t_k \in Dom(X_k) \wedge (t_k, t_l) \models \Gamma(X_k, X_l)\}$. This ensures that for each value t_l in the domain of X_l, there is a value t_k in the domain of X_k such that (t_k, t_l) satisfies all the constraints on arc (X_k, X_l). The current domain of X_l is then intersected with the set derived by the $\uplus$ operation, since any other values for X_l cannot be part of a network solution. A second issue which distinguishes this algorithm from known arc-consistency algorithms is that we are dealing with possibly infinite periodical domains. This not only requires the implementation of algebraic operations over these sets, but may also pose questions about the termination of the algorithm. Essentially, termination is guaranteed by the fact that the equality and inequality tests in the algorithms are limited by the finite constant MAX.[3] The proper identification of this constant has been a relevant technical issue, since it greatly affects the complexity of the algorithm and it is also essential to guarantee its completeness.

While the AC-G algorithm is exactly the one presented in [5], in Figure 6 we present a variant of the procedure to compute $\uplus$ that has lead to significant performance improvements in the implementation of the constraint solver[4].

Very briefly, in Step 1 the indexes of the starting node which denote granules not included in granules of the constraint granularity are excluded (they could not contribute to the result). In Step 2 we determine for each interval of indexes of the starting node the corresponding interval in the ending node by shifting the interval extremes of the minimum and maximum distances specified in the constraint. The optimization we introduce with respect to [5] is that as soon as one of these resulting intervals covers a span of granules greater than the period we stop the procedure (except for computing global bounds in Step 4). Indeed, since we know that the result is periodic with period P, an interval greater than P implies that the result is G itself, possibly limited by new bounds. Step 3 is executed only when none of the intervals resulting from Step 2 is larger than P. In this case the period characterization of the resulting set is obtained by dropping some granules of G accordingly to the intervals derived in Step 2.

The correctness proof can be trivially reduced to the one provided in [5].

4 Implementation and Experimental Results

In this section we provide some information about the technologies used to implement the GSTP service and report some experimental results.

[3] $S_1 \neq^{MAX} S_2$ ($S_1 =^{MAX} S_2$, resp.) means that S_1 and S_2 are different (equal, resp.) if only numbers no greater than MAX are considered.

[4] The version of $\uplus$ reported here assumes a single TCG is associated with each arc. The same procedure can be used in the general case by first transforming the input network into an equivalent one that meets this requirement. Alternatively, a slightly different procedure can be used to operate directly on multi-TCG arcs.

```
INPUT: a finite or periodical set $S$ and a TCG $[m,n]\,G$.
OUTPUT: the finite or periodical set $S \uplus \{[m,n]\,G\}$
METHOD:

Step 1: Replace $S$ with its intersection with $G$. If empty, return the empty set.
Step 2: (Any bound on $S$ and $G$ is ignored here and in Step 3)
    For each interval $[a_i, b_i]$ in the resulting representation of $S$
    Do:
    – compute $[Lo_i, Up_i]$ with $Lo_i = min\lfloor\lceil a_i\rceil^G + m\rfloor^G$, and $Up_i = max\lfloor\lceil b_i\rceil^G + n\rfloor^G$.
    – If $|Up_i - Lo_i| \geq P - 1$ then Goto Step 4 ($G$ is the output set, with bounds as computed in Step 4). Endif.
    – If $Lo_{i-1} \leq Lo_i$ and $\lceil Lo_i\rceil^G \leq \lceil Up_{i-1}\rceil^G + 1$ Do
        – substitute interval $[Lo_{i-1}, Up_{i-1}]$ in the list with $[Lo_{i-1}, Up_i]$.
        – If $|Up_i - Lo_{i-1}| \geq P - 1$ then Goto Step 4 ($G$ is the output set,
        with bounds as computed in Step 4). EndDo
    Otherwise insert $[Lo_i, Up_i]$ in the list. Endif.
    EndDo
Step 3: The period representation of the output set is derived from the one of $G$
    by excluding each granule $G(j)$ such that there is no $K \in \mathbb{Z}$ and no $i$ for which
    we have $j = j' + K * R$, $\lceil Lo_1\rceil^G \leq j' < \lceil Lo_1 + P\rceil^G$, and $\lceil Lo_i\rceil^G \leq j' \leq \lceil Up_i\rceil^G$
    where $R$ is the number of granules of $G$ in each period, $Lo_1$ is the first value
    derived in Step 2 and $[Lo_i, Up_i]$ is the $i$-th pair of bounds in the list computed
    in Step 2.
Step 4: The global bounds of the output set are:
    $Lo = max(min\lfloor\lceil t_{first}\rceil^G + m\rfloor^G, min\lfloor l\rfloor^G)$, and
    $Up = min(max\lfloor\lceil t_{last}\rceil^G + n\rfloor^G, max\lfloor u\rfloor^G)$, where $t_{first}$, $t_{last}$ are the first and
    last values in the set $S$ from Step 1, and $l$, $u$ are the indexes of the first and
    last granule of $G$.
```

Fig. 6. The optimized procedure for $\uplus$.

4.1 Technologies and Optimizations

The Web Service has been implemented in Java using the "Apache Axis" development framework, and Tomcat as the application server. SOAP over HTTP is used to transmit and receive requests. The service is currently active on a publicly accessible server. Java is also the choice for the web service client interface, mostly because of platform independence. It has been tested on recent versions of the Linux and Windows operating systems. On the contrary, the constraint solver has been entirely implemented in ANSI C 99 using the GCC compiler and the libxml2 libraries for parsing XML input. The choice was clearly dictated by the efficiency of the resulting code with respect to other choices.

Regarding optimizations, we worked in particular on the efficient use of memory, devising appropriate data structures and a cache mechanism for granularity internal representation. The cache is initialized after parsing the input constraint network. It is implemented through a hash table having as key the granularity name. Another optimization worth mentioning is a particular ordering of the list

Q of arcs to be considered by the AC-G algorithm. The ordering is based on the computation of the least common multiple among the period of the granularity in the constraint and the periods of the periodical sets for the domains of starting and ending nodes. Arcs with lower values are processed first. This strategy has significantly reduced the processing time.

4.2 Experimental Results

The constraint solver code has been totally rewritten from its first implementation and extensive testing has been performed. In the following we first illustrate how we generated our benchmarks, and then report experimental results both regarding performance and observed upper bounds on the number of executions of the main loops of the algorithm.

Generating Benchmarks. A nontrivial problem is the generation of a significant set of consistent constraint networks. Indeed, the intuitive approach of generating a network by connecting nodes with randomly created constraints leads to a very high rate of inconsistent networks. We addressed this problem by building networks iteratively, ensuring the consistency of the networks obtained at each iteration. The generation process starts from a small consistent network and a node is added at each iteration. This node is connected to a randomly chosen node by labeling the arc with a single-granularity randomly generated constraint C. The GSTP solver is then executed to verify the consistency of the resulting network. If it is not consistent, C is replaced with a new randomly chosen constraint, and this process is repeated until a consistent network is found Then, additional arcs are added between the new node and other randomly chosen nodes, using as labels a relaxed version of the constraints obtained by the GSTP solver that was executed for consistency check[5]. Since the constraints returned by GSTP preserve consistency, their relaxation preserve it as well. Note that, if no relaxation was performed, the generated network would contain a large number of constraints which would not be restricted by path consistency, and this is not realistic for real networks.

Several parameters affect the generation illustrated above, the most relevant being the set of possible granularities and the range of values for the minimum and maximum distance for the randomly generated constraints, and the range of values for the number of arcs added at each iteration. In particular, the last parameter determines the density of arcs in the generated network: we say that a network has a density of $d\%$ if each node connects with about $d\%$ of the other nodes in the network.

The benchmark tests we performed have been obtained generating a great number of networks (tens of thousands). The specific benchmark test considered in this paper includes networks with up to 50 nodes, a density of arcs of 5%, and the granularity of each constraint randomly selected among a set of 10

[5] Note that path-consistency is a step of the algorithm, and returns a set of constraints (each one in a different granularity) for each pair of nodes in the network. We currently take one randomly chosen constraint from each set.

different granularities. Granularities in this set include standard ones with their exceptions (e.g., years modeling leap years), granularities with non-contiguous granules (like business days), and granularities whose granules have internal gaps (e.g., business months).

A different range of values for minimum and maximum distances for each randomly generated constraint is specified depending on the granularity of the constraint. Ranges are chosen trying to simulate realistic problem specifications, and taking into account that small distance values in terms of granularities with large granules are converted during constraint processing into large distance values in terms of granularities with small granules. For example, a maximum distance of +100 in terms of years becomes +885383 in terms of hours. In order to understand the impact of this parameter, we performed a specific test with two sets of networks with the second one generated using ranges which are larger by a order of magnitude with respect to the ones used for the first set.

Performance Results. All the performance experimental results reported here are obtained by averaging the processing time over several networks generated using the same parameters. As expected, the processing time is significantly affected by the number of nodes in the network. However, experimental results have shown that a very relevant parameter is the subset of granularities appearing in the constraints. When the least common multiple of their periods is large, the processing time sharply increases, and the size of the network in terms of number of nodes becomes less relevant.

Figure 7 shows the difference in performance between networks involving a set of granularities having a high common period, with respect to networks with a low common period.

The chart presents the processing time for four classes of networks differing in the granularities used in their constraints. The first class has all constraints in terms of the granularity `hour` and the common period is clearly 1. The second class has constraints in terms of granularities `hour` and `day`, with their common

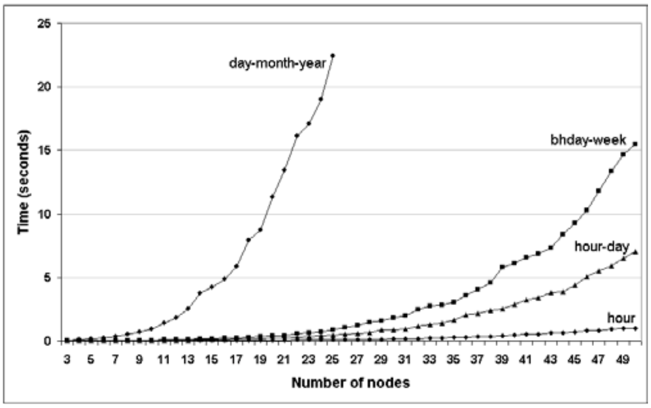


Fig. 7. Computation time for multi-granularity networks.

period being 24. The granularities appearing in the third class of networks are `bhday` and `week`, and their common period is 168. Finally, the fourth class of networks involve `day`, `month`, and `year` with a common period[6] of 35064. The chart shows that the processing time grows significantly with the value of the period. In particular, for the fourth class of networks, the processing time with 25 nodes is greater than the processing time of a 50 nodes network of the third class.

Figure 8 shows how the constraints distance range affects processing time. The two classes of networks considered in Figure 8 have constraints in terms of the same set of granularities (`bhday` and `week`), but the second class has been generated using distance ranges that are larger than the ones used for the first class by one order of magnitude. It is clearly visible how this parameter affects processing time. Accordingly to our experiments, the processing time growth is similar for classes of networks with a single granularity and in classes of networks using multiple granularities.

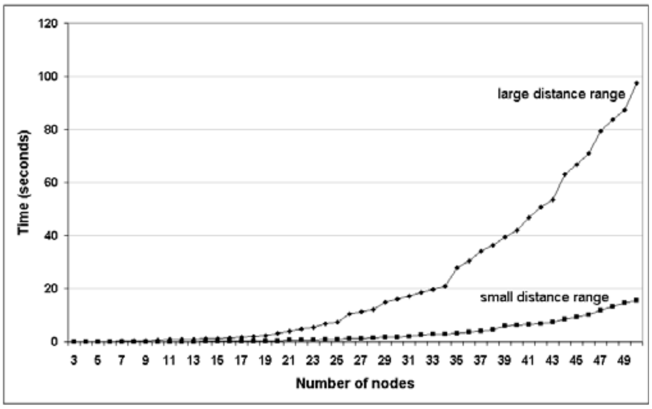


Fig. 8. Impact of the distance range on processing time.

We should also mention that first evaluations on the impact of arc density show very moderate variations of the processing time with changes to this parameter. While we are working on several optimizations that will definitely reduce the execution times measured for this benchmark, the optimizations will not affect the qualitative observations reported above on the main parameters affecting GSTP performance.

Experimental Results on the Upper Bound for Loop Cycles. The extensive testing performed with our benchmarking suite has shown that the number of iterations of Conversion and Path Consistency, i.e., the loop at Step 1 of the algorithm in Figure 4, is very low. Indeed, this number has never been more than 5 in our experiments.

[6] We consider leap years, but ignoring exceptions to the 4 years periodicity.

We have also observed that the number of iterations of the external loop of the algorithm, i.e., the three main steps in Figure 4, is very low. Actually, it has never been more than 2 in our experiments. Note that any network inconsistency is always detected at the first iteration, either by Conversion+PC or by the first run of AC-G.

These results are very significant to us since the current theoretical bounds on the number of iterations depend on the range of values in the constraints. They indicate that, despite our efforts in this direction, a much better theoretical characterization of the bounds on iterations can probably be obtained.

5 Conclusions

In this paper we presented the GSTP system, a web service to solve TCSP expressed in terms of multiple time granularities. It is to our knowledge the only implementation of a complete algorithm for consistency checking of these networks. We are currently working in two directions. On one side we are still working at the optimization of reasoning algorithms, while on the other side we are enhancing the web service XML-based architecture to provide easy access to the algorithms by external applications, and to facilitate the specification of user-defined time granularities to be used in GSTP.

Acknowledgments

Many people contributed to the implementation of the GSTP system. In particular we would like to thank Carlo Cestana for his work on the GSTP web service, Simone Ruffini for his work on granularity constraint conversions, and Sean Wang for his valuable suggestions on implementation issues. This work has been partially supported by Italian MIUR (FIRB "Web-Minds" project N.RBNE01WEJT_005).

References

1. C. Bessière. Arc-Consistency and Arc-Consistency Again. *Artificial Intelligence* 65(1):179–190, 1994.
2. C. Bettini, S. Mascetti, V. Pupillo. GSTP: A Temporal Reasoning System Supporting Multi-Granularity Temporal Constraints, in Proc. of Int. Joint Conference on Artificial Intelligence (IJCAI), (Intelligent Systems Demonstrations), pp. 1633-1634, Morgan Kaufmann, 2003.
3. C. Bettini, X. Wang, S. Jajodia. A General Framework for Time Granularity and its Application to Temporal Reasoning. *Annals of Mathematics and Artificial Intelligence* 22(1,2):29–58, 1998.
4. C. Bettini, S. Ruffini, Granularity Conversion of quantitative temporal constraints DICo – Università di Milano, Technical Report N. 276-02, `http://homes.dico.unimi.it/~bettini/papers/tr276-02.pdf`. Under journal review.

5. C. Bettini, X. Wang, S. Jajodia, Solving Multi-Granularity constraint networks, *Artificial Intelligence*, 140(1-2):107–152, 2002.
6. Hubie Chen, Periodic Constraint Satisfaction Problems: Polynomial-Time Algorithms. In Proc. of Int. Conf. on Principles and Practice of Constraint Programming, LNCS 2833, pp. 199–213, Springer, 2003.
7. R. Dechter, I. Meiri, J. Pearl. Temporal constraint networks. *Artificial Intelligence* 49:61–95, 1991.
8. A. Mackworth, E. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* 25:65–74, 1985.
9. A.K. Mackworth, J. A. Mulder, W. S. Havens. Hierarchical Arc Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems. *Computational Intelligence*, 1:118–126, 1985.