Reasoning about a Temporal Scenario in Natural Language

Benjamin Han
IBM Watson Research Center
1101 Kitchawan Road, Yorktown Heights, NY 10598, U.S.A.
dbhan@us.ibm.com

Abstract

Linguistically the temporal information of an event is often introduced in an incremental and incomplete fashion, and understanding a complete temporal scenario requires both a flexible event-level representation and a global model capable of capturing the interactions among them. In this paper we describe a method of constructing a Dependency Simple Temporal Problem with Mixed Granularities (DGSTP) from a set of event-level representations. The constraint network can then be solved to obtain a set of possible times for the events and to discover implicit temporal relationships among them.

Introduction

The capability to deduce the temporal location of an event described linguistically can benefit many real-world applications such as question answering, text summarization and intelligence analysis. Like many phenomena in natural language, however, temporal information about events are usually given throughout a discourse in a piecemeal fashion, often incomplete. For example, consider the following sentence in a news story published on Aug 16, 2006:

> *Karr admitted to being involved in the death of the **6-year-old** beauty pageant winner.*

Lacking any additional information, one might assume that the death had occurred in the same year as the publication and be tempted to conclude that the victim was born in year 2000. But if the reader is given another sentence from the same story:

> *Authorities are examining John Mark Karr's writings for clues that might link him to the death of JonBenet Ramsey **10 years ago**.*

It is then possible to conclude that the victim was born in 1990, assuming that the two *death* events described are identical. These observations can be summarized using the following fomulae:

$$t_1 := \{t_0 + |6_{\text{year}}|\} \quad (1)$$

$$t_2 := \{\{2006_{\text{year}}, \text{aug}, 16_{\text{day}}\} - |10_{\text{year}}|, = t_1\} \quad (2)$$

Each formula encodes the *local* temporal information of a death event: (1) defines temporal variable t_1 to be a time point 6 years after the birth of the victim (t_0), and (2) defines variable t_2 to be a point 10 years before the publication date. The assumption that the two events are identical is then explicated as a conjunctive constraint in the second half of (2) ($= t_1$) - this assumption can come from an automatic event coreference system, or simply come from a human analyst performing a what-if experiment.

In the above we have captured the *global* information of a temporal scenario in a set of time formulae, and all there is left is a way to systematically solve for the variables. In this paper we will describe a method for solving such formulae by constructing a temporal constraint problem called *Dependency Simple Temporal Problem with Mixed Granularities* (DGSTP), where temporal relations among events are encoded as temporal constraints among the temporal variables. The resulting constraint network can then be solved to determine its consistency, to obtain a set of possible times for the variables, and to discover other valid temporal relationships among the variables.

In the next section we will first make a brief introduction on our event-level temporal representation called TCNL. We will then describe in the following three sections a progression of three classes of temporal constraint problems: Temporal Constraint Satisfaction Problems (and in particular its subclass STP), GSTP as an extension to STP with mixed granularities, and the further extension DGSTP. In particular the core solution procedures are described in the section *Modeling Temporal Scenarios*, and the methods for translating TCNL formulae into a DGSTP, our tool for modeling a temporal scenario, are discussed in the section *From Formulae to Dependency GSTP*. Finally we conclude the paper and suggest future work in the final section.

Event-Level Representation

Temporal information of an event can be conveyed linguistically via verbal tenses, temporal expressions ("*10 years ago*"), prepositional words ("*before*" and "*during*") and aspectual relations ("*admitted to being involved*" where the admission happened after the involvement). We encode this information using an arithmetic-like formalism called *Time Calculus for Natural Language* (TCNL), where temporal information is viewed as constraints to the possible times an

event can take place. In the recent years TCNL has been successfully applied to the task of normalizing temporal expressions found in emails (Han, Gates, and Levin 2006b), newswire and web texts (Florian et al. 2007). In this section we will provide a concise review of TCNL, but readers are recommended to refer to (Han, Gates, and Levin 2006a) and (Han 2008) for a more detailed description.

Calendar Models

The foundation of TCNL is a constraint-based calendar model providing a repertoire of temporal concepts for writing time formulae. There are two kinds of concepts: temporal *units* (e.g., `month`) and temporal *values* (e.g., `feb`), and each unit can take on a set of fully ordered values. The entire calendar model is therefore a constraint satisfaction problem (CSP) (Dechter 2003), with each temporal unit acting as a variable, and the modeling task involves designing constraints among a set of units (e.g., February in a non-leap year cannot have 29 days).

Temporal units in the calendar model are ordered by two relations: the *measurement* relation and the *periodicity* relation. Unit u_i *is measured by* u_j, written as $u_j \dot{\leq} u_i$, if every value of u_i can be mapped to a set of consecutive values of u_j on a time line; e.g., `month` is measured by `day`. A unit u_j *is periodic in* u_i, written as $u_j \rightarrowtail u_i$, if u_j is measured by u_i and iterating through the values of u_j does not advance the value of u_i;[1] e.g., `day` (days of a month) is periodic in `month` but is not periodic in `week`, because iterating through all possible values of days of month surely advances the time from one week to another. These two relations play a crucial role in defining the concepts of granularity and the anchoring status of a time entity.

TCNL Formulae

Built on top of the constraint-based calendar model is a way of representing temporal semantics via *formulae*. Every TCNL formula is of one of the three possible types: *coordinates* ($\mathbb{C}$), *quantities* ($\mathbb{Q}$) and *enumerations* ($\mathbb{E}$).[2] A coordinate represents a time point and is essentially a set of assignments to the temporal units of a calendar model; e.g., $\{\texttt{fri}, 13_{\texttt{day}}\}$ represents the under-specified expression "*Friday the 13th*".[3] A quantity represents a certain number of temporal units or coordinates; e.g., $|2_{\texttt{day}}|$ means "*2 days*" and $|2_{\{\texttt{fri}\}}|$ represents "*two Fridays*". Finally an enumeration represents a set of coordinates such as intervals ([{`wed`}: {`fri`}] for "*Wednesday to Friday*") and discontiguous sets ([{`wed`}, {`fri`}] for "*Wednesday and Friday*"). The basic idea behind a TCNL formula is to translate whatever is said in an expression into a constraint satisfaction problem in the hope of inferring more information.

[1] This is a simplified definition – the complete definition involves concepts of *periods* and the *immediate* measurement relation (Han 2008)

[2] We use $\{\cdot\}$, $|\cdot|$ and $[\cdot]$ respectively to mark the formulae of these types.

[3] Subscript `dow` (day-of-week) is dropped for `fri` since there is no ambiguity.

Associated with a formula f is its *granularity*: it is the set of minimal units (under the measurement relation) appearing in f:

$$\mathrm{g}(f) = \min(\{u | u \in f\}) \tag{3}$$

For examples $\mathrm{g}(\{2006_{\texttt{year}}, \texttt{aug}\}) = \{\texttt{month}\}$ (because $\texttt{month} \leq \texttt{year}$) and $\mathrm{g}(|2_{\texttt{day}}|) = \{\texttt{day}\}$. We also say $\mathrm{g}(f_1) \leq \mathrm{g}(f_2)$ if for every unit $u_i \in \mathrm{g}(f_1)$ we can find $u_j \in \mathrm{g}(f_2)$ such that $u_i \dot{\leq} u_j$. Granularity of a coordinate can also be used to check the "anchoring" status of a coordinate. Intuitively $\{2007_{\texttt{year}}, \texttt{may}\}$ ("*May 2007*") is *anchored* in the sense that it can be identified as a unique interval on a timeline, but $\{\texttt{may}\}$ is not. This distinction is defined as follows: a coordinate c is anchored if for every $u_1 \in \mathrm{g}(c)$ there exists a path $\langle u_n, \ldots, u_1 \rangle$ such that $\pi_{u_i}(c)$ is defined, where $u_i \rightarrowtail u_{i+1}$ for $i = 1 \ldots (n-1)$ and u_n is a maximal unit under the measurement relation. E.g., $c = \{2007_{\texttt{year}}, \texttt{may}\}$ is anchored because $\texttt{month} \in \mathrm{g}(c)$, $\pi_{\texttt{month}}(c) = \texttt{may}$ and $\pi_{\texttt{year}}(c) = 2007$, $\texttt{month} \rightarrowtail \texttt{year}$ and `year` is a maximal unit under the measurement relation.

Operators and Relations

The representational power of TCNL mostly comes from its set of infix operators and relations (see Table 1 and 2). Each of them has a set of type requirements stipulating the types of its operands; e.g. in $\{\_ + |1_{\texttt{day}}|\}$ ("*the next day*") the left operand '_' (a temporal variable representing the temporal focus) must be of type $\mathbb{C}$ or $\mathbb{E}$, the right operand $|1_{\texttt{day}}|$ must be of type $\mathbb{Q}$, and the entire term is of type $\mathbb{C}$. Each operator also ensures the result is at the granularity of one of its operands. The operators '+'/'−' implement a granularity-sensitive arithmetic: the granularity of the left operand (op_l) will first be converted to that of the right operand (op_r) before the addition/subtraction, therefore the result is at the granularity $\mathrm{g}(op_r)$; e.g., $\{\{2006_{\texttt{year}}, \texttt{feb}, 1_{\texttt{day}}\} + |2_{\texttt{month}}|\}$ is evaluated to $\{2006_{\texttt{year}}, \texttt{apr}\}$, with the information at `day` granularity eliminated. The selection operator '@' picks time points from op_r based on the constraints given by op_l, therefore the result granularity is $\mathrm{g}(op_l)$. Finally the merging operator '&' merges the non-conflicting constraints from op_l to op_r (result granularity is $\mathrm{g}(op_l)$) and the proximity operator picks the nearest time point around op_l that satisfies the constraints given in op_r (result granularity is $\mathrm{g}(op_r)$).

Temporal Variables

Temporal variables in TCNL serve two purposes: they are used to represent contextual information and to encode interactions among formulae. There are two kinds of variables in TCNL: the pre-defined variables speech time '`now`' (of type $\mathbb{C}$) and temporal focus '_' (of type $\mathbb{C}$ or $\mathbb{E}$), and user variables (can be of type $\mathbb{C}$ or $\mathbb{E}$). Formulae making references to only the pre-defined variables are always easy to evaluate – we just need to substitute the variables with their denotations and evaluate away[4]. On the other hand, formulae using user variables are not always straightforward; e.g.,

[4] The denotation of '_' needs to be determined by an external module.

Operator	Semantics	Type requirements	Result granularity	Example
+ and -	Arithmetic	$(\mathbb{C} \mid \mathbb{E}) \times \mathbb{Q} \to \mathbb{C}$	g(op_r)	{_ + $\lvert 1_{\texttt{day}} \rvert$} (“*the next day*”)
@	Selection	$\mathbb{Q} \times (\mathbb{C} \mid \mathbb{E}) \to \mathbb{C}$ $(\mathbb{C} \mid \mathbb{E}) \times (\mathbb{C} \mid \mathbb{E}) \to \mathbb{E}$	g(op_l)	{$\lvert 2_{\{\texttt{sun}\}} \rvert$ @ {now + $\lvert 0_{\{\texttt{may}\}} \rvert$}} (“*the 2nd Sunday this May*”) [{$9_{\texttt{hour}}$} @ [{now + $\lvert 0_{\{\texttt{wed}\}} \rvert$}: {fri}]] (“*9am on this Wednesday, Thursday, and Friday*”)
&	Merging	$\mathbb{C} \times \mathbb{C} \to \mathbb{C}$	g(op_l)	{now & {now + $1_{\texttt{year}}$}} (“*this time next year*”)
∧	Proximity	$(\mathbb{C} \mid \mathbb{E}) \times \mathbb{C} \to \mathbb{C}$	g(op_r)	{{$\lvert 1_{\{\texttt{mon}\}} \rvert$ @ {now + $\lvert 0_{\texttt{sep}} \rvert$}} ∧ {weekend}} (“*this Labor Day weekend*”)

Table 1: Operators in TCNL; op_l/op_r is the left/right operand.

Relations	Semantics	Type requirements	Examples
<, <=, =, >=, >	before, before or equal-to, equal-to, after or equal-to, and after	$\mathbb{C} \times \mathbb{C}$	{< {$2006_{\texttt{year}}$, may}} (“*sometime before May 2006*”)
b , **s** , **d** , **de** , **f** , **di**	before, starting, during, during/equal, finishing, and after; **de** :=(**s** or **d** or **f**)	$\mathbb{C} \times \mathbb{E}$	{**d** [{now + $\lvert 0_{\texttt{day}} \rvert$}: $\lvert 3_{\texttt{day}} \rvert$]} (“*sometime during the 3-day period starting from today*”)
b , **s** , **f** , **bi**	before, starting at, finishing at, and after	$\mathbb{E} \times \mathbb{C}$	[**s** now] (“*from now on*”)
b , **m** , **o** , **s** , **d** , **f** , **=** , **fi** , **di** , **si** , **oi** , **mi** , **bi**	See (Allen 1984).	$\mathbb{E} \times \mathbb{E}$	

Table 2: Selected relations in TCNL.

{t_0+$\lvert 6_{\texttt{year}} \rvert$} is *resolvable* only when t_0 is defined with a resolvable formula $t_0 \coloneqq \{1990_{\texttt{year}}\}$ (or we say when t_0 is resolvable). For a unresolvable formula the process of *variabilization* kicks in to automatically introduce a variable representing the formula (e.g., $t_1 \coloneqq \{t_0 + \lvert 6_{\texttt{year}} \rvert\}$), and the constraints between this variable and the others in the formula can then be extracted for constraint solving (described later).

Temporal Constraint Satisfaction

As motivated in *Introduction*, to fully understand a temporal scenario it is often insufficient to consider events individually. Instead we will capture the temporal relations among events by way of constructing a variation of Temporal Constraint Satisfaction Problems (TCSP). This section introduces the basic concepts behind TCSP and its subclass STP.

A TCSP is a particular kind of constraint satisfaction problems (Dechter, Meiri, and Pearl 1991): it contains a set of temporal variables $\{t_1, \ldots, t_n\}$ with continuous domains and a set of unary/binary constraints. A binary constraint between variable t_i and t_j must be formulated as a disjunction of allowed *time differences* between the variables: $(a_1 \le t_j - t_i \le b_1) \vee \ldots \vee (a_k \le t_j - t_i \le b_k)$ (also written as a set of disjunctive intervals $\{[a_1, b_1], \ldots [a_k, b_k]\}$ and is said to have a *scope* $\{t_i, t_j\}$), and a unary constraint on t_i is encoded as a binary constraint between t_i and t_0, which is an artificially introduced variable with a singleton domain {0}. A tuple $(a_1, \ldots a_n)$ is a solution to a TCSP if the assignment $(t_1 = a_1, \ldots t_n = a_n)$ violates no constraint, and a TCSP is consistent if there exists at least one solution to the problem.

Solving a TCSP is a NP-hard problem. However if no disjunction is allowed in any constraint, solving the problem - a Simple Temporal Problem (STP) - takes only polynomial time. This is done by converting an STP to its corresponding “distance graph”: for a constraint $[a_k, b_k]$ from variable t_i to t_j, we add an edge (t_i, t_j) of distance b_k and an edge (t_j, t_i) of distance $-a_k$ to the graph. We can then run Floyd-Warshall’s all-pairs-shortest-paths algorithm on the distance graph to derive the minimal but equivalent STP (takes $O(n^3)$ time): a constraint from t_i to t_j is $[a'_k, b'_k]$ when the shortest distance from t_i to t_j (or t_j to t_i) is b'_k (or $-a'_k$). The STP is consistent if no negative cycle is detected, and a backtrack-free search can be used to assemble a solution.

Despite its no-disjunction-allowed restriction, STPs are attractive in its simplicity and efficiency. We shall therefore focus on STPs in the rest of the work.

Modeling Temporal Scenarios

STPs have many deficiencies for our purpose due to their disconnect from natural language. For one they do not accommodate temporal constraints expressed in mixed granularities (e.g., [10, 20]day and [5, 10]month). Another problem is their use of the artificial “origin of time” (t_0) to transform unary constraints into binary ones: this approach is not applicable to many unary constraints often encountered in natural language, such as “variable t_2 must be a *Tuesday*.”

In this section we describe GSTP as an extension to STP that allows mixed granularities. Our formulation is based on (Bettini, Wang, and Jajodia 2002) but specifically designed to work with our event-level representation TCNL.

Formulating the GSTP

In our version of GSTP each variable can have a set of unary constraints specified in the form of a (usually unanchored) coordinate, and the domain of the variable contains all possible *anchored* coordinates satisfying the constraints (e.g., the

domain of variable t_2 in Fig. 1 contains all possible Tuesdays). Each binary constraint tc in a GSTP is colored by a temporal unit u, and we overload the granularity function in (3) to give $\mathrm{g}(tc) = \{u\}$. The granularity of a variable t, on the other hand, is determined by its unary constraints uc (a coordinate) and TC, the set of binary constraints whose scopes include t:

$$\mathrm{g}(t) = \mathrm{glb}((\bigcup_{tc_i \in TC} \mathrm{g}(tc_i)) \cup \mathrm{g}(uc)) \tag{4}$$

$\mathrm{glb}(\cdot)$ is a function returning the greatest lower bound unit of the input set under the measurement relation. For example in Fig. 1 we have $\mathrm{g}(t_3) = \mathrm{glb}(\texttt{month}, \texttt{day}) = \{\texttt{day}\}$ (recall $\texttt{day} \dot{\leq} \texttt{month}$).

In an STP a *qualitative* constraint such as $t_i <= t_j$ can be represented by converting it into its *quantitative* counterpart $[0, \infty]$. In a GSTP this can be done similarly with special care taken to infer appropriate granularity for the resulting constraints: for a qualitative constraint tc whose scope is $\{t_i, t_j\}$, we add a quantitative constraint of granularity $\{u_k\}$ for every $u_k \in \min(\mathrm{g}(t_i) \cup \mathrm{g}(t_j))$. E.g., if $\mathrm{g}(t_i) = \{\texttt{day}\}$ and $\mathrm{g}(t_j) = \{\texttt{hour}\}$, we can convert $t_i <= t_j$ into $[0, \infty]\texttt{hour}$, or $t_i < t_j$ into $[1, \infty]\texttt{hour}$. It can be easily shown that this conversion will ultimately lower every variable weakly connected by qualitative constraints into a common granularity (because of (4)), but it will not alter the granularity of any of the other variables. This allows us to use the following one-pass procedure for inferring granularity in a GSTP with both quantitative and qualitative constraints:

Procedure 1 (Granularity inference)**.**

1. For every variable t compute $\mathrm{g}(t)$ according to (4) if possible; if not (because t has no unary constraint and participates no quantitative constraint), assign a default granularity to $\mathrm{g}(t)$.
2. For a set of variables T weakly connected by qualitative constraints, assign $\mathrm{g}(t_j) = \mathrm{glb}(\bigcup_{t_i \in T} \mathrm{g}(t_i))$ for all $t_j \in T$.

Constraint Propagation

Using a coordinate as the implicit domain of a variable has two consequences: the domain is no longer contiguous, and the propagation processes of the unary and the binary constraints are now separate. The first consequence also implies that disjunctions are back to the GSTP, thus breaking the *decomposability* that enables a backtrack-free search for solutions (Dechter, Meiri, and Pearl 1991). Assuming a GSTP with only quantitative constraints (i.e., all qualitative constraints are already quantified), we use an approximate method similar to the one described in (Bettini, Jajodia, and Wang 2000) to propagate the binary constraints (pictorially depicted in Fig. 1):

Procedure 2 (Approximate constraint propagation)**.**

1. We first decouple a GSTP into single-granularity STP_{u_i} where $\{u_i\}$ is a granularity.
2. We then run the all-pairs-shortest-paths algorithm on each STP_{u_i} to derive its minimized counterpart. If any negative cycle is detected, stop the algorithm and report the inconsistency. E.g., in Fig. 1 the constraint $[30, 60]\texttt{day}$ in $\mathrm{STP}_{\texttt{day}}$ is produced by this step.
3. For every STP_{u_i} we convert its constraints into granularity $\{u_j\}$ and add them into STP_{u_j}, meaning intersecting a preexisting constraint between the same pair of variables with the new one. If any constraint is refined as a result, we go back to the previous step. E.g., in Fig. 1 the constraint from t_2 to t_3 in $\mathrm{STP}_{\texttt{month}}$ was $[-\infty, \infty]$ before this step, and is refined to $[1, 2]\texttt{month}$ afterwards. We will come back to the granularity conversion of a constraint later.
4. Finally we conjoin the constraints of all possible granularities and produce a propagated GSTP.

Procedure 2 is an approximation because the granularity conversion done in the step 3, although guaranteeing no loss of solutions, could unduly enlarge the set of solutions. The advantage of this procedure is its polynomial time complexity: if we have m different granularities and n variables, the overall runtime is $O(mn^2I(m+n))$ where I is the number of the iterations. For GSTP with single granularity this reduces to $O(n^3)$, otherwise the loop can run at maximum mn^2w iterations (w is the maximum width of any constraint after the first iteration). Although in practice the procedure seldom runs longer than a few iterations, when quantifying a qualitative constraint we replace ∞ with a large number to avoid this potentially infinite runtime.

Granularity Conversion

The step 3 of Procedure 2 converts a constraint $[a, b]u_1$ into $[a', b']u_2$ where u_1 and u_2 are two different temporal units and $a \leq b, b > 0$. The conversion must satisfy one criterion: if an assignment satisfies $[a, b]u_1$, the same assignment must also satisfy $[a', b']u_2$ (i.e., no loss of solutions). E.g., in Fig. 1 the constraint $[0, 1]\texttt{month}$ is a valid conversion of $[10, 20]\texttt{day}$ since all possible assignments of t_1 and t_2 with difference between 10 to 20 days must also fall within a 0- to 1-month window. It is therefore clear that we prefer a "tighter" conversion since the target constraint can be made arbitrarily lenient to let in more assignments.

In general this conversion task can be very difficult because temporal granules do not always have fixed sizes. As an example, we could argue that in Fig. 1 $[1, 2]\texttt{month}$ is not the tightest conversion possible for $[20, 40]\texttt{day}$ from variable t_2 to t_3, since the difference between March 1, 2006 and April 9, 2006 is clearly less than two months. Our approach is only a result of compromise: it is a constant-time operation and it requires much simpler engineering in terms of calendar modeling.

We first define two utility functions $\mathrm{minsize}(u_i, u_j)$ and $\mathrm{maxsize}(u_i, u_j)$ where u_i and u_j are two temporal untis and $u_j \dot{\leq} u_i$: the functions return the min/max number of consecutive granules of u_j that can overlap on a timeline with a granule of u_i. E.g., $\mathrm{minsize}(\texttt{month}, \texttt{day}) = 28$ and $\mathrm{maxsize}(\texttt{month}, \texttt{day}) = 31$. The conversion is then given as follows ($u = \mathrm{glb}(u_1, u_2)$):

$$b' = \left\lfloor \frac{(b+1) \cdot \mathrm{maxsize}(u_1, u) - 1}{\mathrm{minsize}(u_2, u)} \right\rfloor$$

$$a' = \left\lfloor \frac{(a-1) \cdot \mathrm{minsize}(u_1, u) + 1}{\mathrm{maxsize}(u_2, u)} \right\rfloor \qquad (\text{if } a > 0)$$

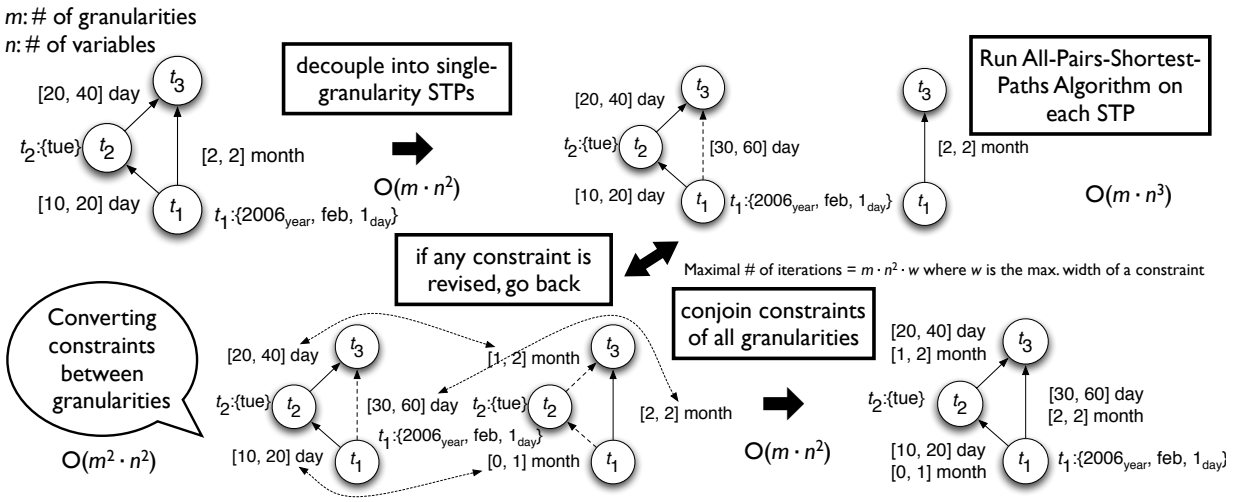

Figure 1: Approximate method for constraint propagation on a GSTP.

Note when $a \leq 0$ we can simply swap the two bounds and treat $-a$ as an upper bound.

Assembling Solutions

After constraint propagation is done on a GSTP and no inconsistency is reported, we still need to check if any solution exists to confirm its consistency. For certain applications it is also desirable to find some or all solutions of a GSTP. For these purposes we use a simple backtracking method for assembling solutions of a propagated GSTP:

Procedure 3 (Backtracking search for solutions of a propagated GSTP)**.**

1. For each variable t_i compute its initial domain.
2. From a list of unassigned variables pick t_i.
3. Pick the next possible anchored coordinate from the domain of t_i at its inferred granularity; if it is not possible, backtrack to the previously assigned variable as the new t_i and re-run this step; if no previously assigned variable is available, stop.
4. For each possible granularity and variable $t_j \neq t_i$, update the domain of t_j based on the constraint from t_i to t_j in that granularity. If the domain of t_j should become empty, return to 3. Note that when updating domains we use the TCNL operator '+' and invoke granularity conversion on coordinates if necessary. Continuing the example given in Fig. 1, assuming t_1 is already assigned with $\{2006_{\texttt{year}}, \texttt{feb}, 1_{\texttt{day}}\}$ and we want to update t_3 using the constraint $[2,2]\texttt{month}$, we first compute $\{t_1 + |2_{\texttt{month}}|\} = \{2006_{\texttt{year}}, \texttt{apr}\}$. We then convert the granularity of the result to $g(t_3) = \{\texttt{day}\}$ and derive the new bounds $\{2006_{\texttt{year}}, \texttt{apr}, 1_{\texttt{day}}\}$ and $\{2006_{\texttt{year}}, \texttt{apr}, 30_{\texttt{day}}\}$.
5. t_i is now assigned; if there is no unassigned variable left, report all possible assignments as solutions, then backtrack to the previously assigned variable as the new t_i and return to 3. Otherwise return to 2.

Note that the ordering of the unassigned variables in step 2 can greatly affect the performance of the procedure. One useful ordering is to pick a more constrained variable with a smaller domain earlier in the search process. For example, in Fig. 1 we use the ordering $t_1 \rightarrow t_2 \rightarrow t_3$: after assigning the only possible coordinate $\{2006_{\texttt{year}}, \texttt{feb}, 1_{\texttt{day}}\}$ to t_1 we update the domain of t_2 to contain only $\{2006_{\texttt{year}}, \texttt{feb}, 14_{\texttt{day}}\}$ and $\{2006_{\texttt{year}}, \texttt{feb}, 21_{\texttt{day}}\}$ and the domain of t_3 to contain only $\{2006_{\texttt{year}}, \texttt{apr}, 1_{\texttt{day}}\}$ and $\{2006_{\texttt{year}}, \texttt{apr}, 2_{\texttt{day}}\}$. Later

iterations will eliminate $\{\texttt{2006}_{\texttt{year}}, \texttt{feb}, \texttt{14}_{\texttt{day}}\}$ from the domain of t_2 and give us two solutions in total.

From Formulae to Dependency GSTP

We are now left with the final task of translating a set of TCNL formulae into a GSTP. Naturally this translation needs to deal with the various syntactic and semantic devices provided by TCNL. An immediate complication is that several of the TCNL operators – such as the proximity operator '@' – have semantics not expressible in the form of a time-difference constraint. We will propose an extension *Dependency GSTP* (DGSTP) to address this problem.

At a higher level, since our GSTP extension only allows variables with coordinate domains to be present, while TCNL allows variables to be of type $\mathbb{C}$ or $\mathbb{E}$, we need to re-interpret a constraint to eliminate any possible variable of type $\mathbb{E}$ in its scope. A corollary is that we need to infer variable types first - this is our next topic below.

Variables and Their Types

Variables in a TCNL formula can be of type $\mathbb{C}$ or $\mathbb{E}$. If a variable is defined explicitly, it must have the same type as its definition (e.g., in $t \coloneqq \{t_0 + |\texttt{6}_{\texttt{year}}|\}$ we have $type(t) = \mathbb{C}$). Otherwise its type can be inferred from the contexts as follows. For each context the variable is in, if it is used as an operand to an operator/relation, the context allows the types compatible with the requirements of the operator/relation (see Table 1 and 2), otherwise we assume the context allows both types. After considering all of the contexts, the variable is assigned the type that is compatible to all: if both $\mathbb{C}$ and $\mathbb{E}$ are compatible, $\mathbb{C}$ is picked, but if no compatible type can be found, a type mismatch is detected and no more processing is attempted. E.g., in $\{\{\texttt{15}_{\texttt{hour}}\}\ @\ t\}$ we have $type(t) = \mathbb{C}$, but in formulae $\{t, \texttt{15}_{\texttt{hour}}\}$ and $[\mathbf{d}\ t]$ we assign $type(t) = \mathbb{E}$.

A slight complication for determining $type(t)$ arises when variable t is involved in relation '=' (both a $\mathbb{C} \times \mathbb{C}$ and an $\mathbb{E} \times \mathbb{E}$ relation): if we have $t = t'$ and $type(t) = \mathbb{E}$, we will assign $type(t') = \mathbb{E}$ as well. This "type propagation" can be easily done over the closure of the '=' relation.

Having decided types for variables, for every variable t of type $\mathbb{E}$ we then create two *bound variables* t^-/t^+ (of type $\mathbb{C}$) to represent its lower/upper bound, and we also add a constraint $t^- \leq t^+$ to relate the two. Our goal later in the section *Constraint Re-interpretation* will be replacing all occurrences of t (of type $\mathbb{E}$) in constraints with its bound variables and re-interpreting the constraints.

Translating Coordinates

A coordinate formula in TCNL can pack a lot of information. Among the terms that can appear inside a coordinate are temporal values (e.g., $\{\texttt{feb}\}$), embedded coordinates (e.g., $\{\{\texttt{feb}\}, \texttt{1}_{\texttt{day}}\}$), terms with operators (e.g, $\{\texttt{now} + |\texttt{6}_{\texttt{year}}|\}$) and relations (e.g., $\{< \{\texttt{2006}_{\texttt{year}}, \texttt{may}\}\}$). A unresolved variable can appear almost at any place where a coordinate/enumeration is expected. Below we will discuss each of the possibilities.

Translating a term with operator + or − is straightforward: if a term t_j+q (or t_j-q) appears in the formula represented by t_i and $\mathrm{g}(q) = \{u\}$, we add a constraint $[q^-, q^+]u$ from variable t_j to t_i (or from t_i to t_j, respectively), where q^-/q^+ is the lower/upper bound of q. E.g., for $t_1 \coloneqq \{t_0 + |<= |\texttt{6}_{\texttt{year}}||\}$ we add a constraint $[0, 6]\texttt{year}$ from t_0 to t_1.

We call the terms containing the other operators (@, & and ∧) *dependency terms* (or d-terms for short) since there are "dependencies" among the involved variables that cannot be made explicit by a time-difference constraint. As a simplification TCNL only allows resolvable formulae to be used as the left operand for the operators '@' and '&' and the right operand for the operator '∧'. From these terms we can still add useful time-difference constraints based on the semantics of the operators, thus making it possible for Procedure 2 to narrow down the domains of the involved variables. We can also *inversely* infer the possible values of a unassigned variable if some of the other variables in these terms are assigned. Consider the formula $t \coloneqq \{|\texttt{2}_{\texttt{day}}|\ @\ [t_1 : t_2]\}$ (the second day between t_1 and t_2, inclusive): obviously the constraints $t_1 \leq t \leq t_2$ and $t_1 \leq t_2$ must be true, and if we know t_2 is May 2, 2006 then t_1 must be on or before May 1, 2006 and t must be on or before May 2, 2006. We will defer the discussion on the inverse inference to the section *Solviong DGSTP*.

Fig. 2 shows the three allowed d-terms and their accompanied time-difference constraints: q and c are a quantity and coordinate constant respectively, and each d-term is represented by a new variable d. For all of the operators the d-terms are obviously equal to t at the respective result granularity. For the merging operator '&', we constrain t' to be equal to the result of the operator at the granularity $\mathrm{lub}(\mathrm{g}(c) \cup \mathrm{g}(t'))$ ($\mathrm{lub}(\cdot)$ returns the least upper bound unit of the input set under the measurement relation) based on the intended use case of the operator; e.g., in $t \coloneqq \{\{\texttt{now}+|\texttt{0}_{\texttt{day}}|\}\&t'\}$ for "*this day in that week*", we constrain the d-term with t' via constraint $[0, 0]\texttt{week}$ ($\mathrm{lub}(\{\texttt{day}\} \cup \{\texttt{week}\})$). For the proximity operator ∧ we specify a "search window" by introducing the constraint $[-w, w]\texttt{u}_{\texttt{w}}$ between the d-term and t', because the output of such a term cannot be constrained otherwise; e.g., $\{t' \wedge \{\texttt{fri}, \texttt{13}_{\texttt{day}}\}\}$ (the closest Friday the 13th relative to t') can be earlier or later than t' depending on what t' is resolved to.

Translating a term with a $\mathbb{C} \times \mathbb{C}$ relation (Table 2) is straightforward since we can always quantify qualitative constraints using Procedure 1. Translating terms with $\mathbb{C} \times \mathbb{E}$ relations is also easy since we can reduce them into a conjunctive set of $\mathbb{C} \times \mathbb{C}$ relations; e.g., $t \coloneqq \{\mathbf{d}\ [t_1 : t_2]\}$ is equivalent to $t \coloneqq \{> t_1, < t_2\}$.

For terms of sole variables such as $t \coloneqq \{t', \ldots\}$, they have different semantics compared to $t \coloneqq \{= t', \ldots\}$: the former uses t' to build up t while the latter declares both variables to be equivalent. They also contribute different constraints: the former gives a quantitative constraint $[0, 0]u_i$ for every $u_i \in \mathrm{g}(t')$, but the latter gives a qualitative constraint $t = t'$, which will result in a granularity propagation that brings both $\mathrm{g}(t)$ and $\mathrm{g}(t')$ to a common granularity according to Procedure 1.

Translating Enumerations

Given an enumeration formula $[\ldots, \texttt{t}', \ldots]$ where t' is a unresolved term, from above we know a variable t of type $\mathbb{E}$

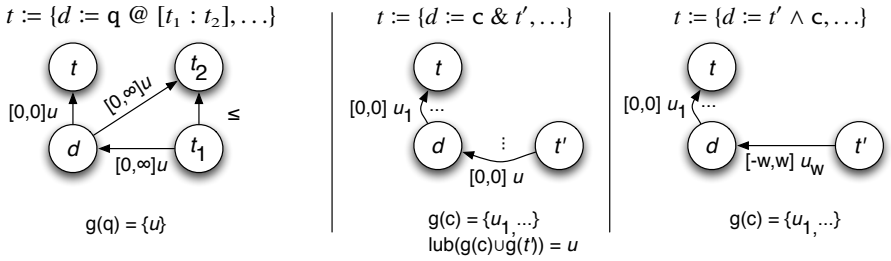

Figure 2: Converting dependency terms into constraints; lub(·) returns the least upper bound unit of the input set under the measurement relation.

will be introduced via the variabilization process together with two bound variables t^-/t^+. If $type(t') = \mathbb{C}$, we can account for the term by introducing constraints $t^- \leq t'$ and $t' \leq t^+$; otherwise we produce constraints $t^- \leq t'^-$ and $t'^+ \leq t^+$ if $type(t') = \mathbb{E}$.

For terms that use the selection operator @, we will introduce constraints relating the bound variables of the host formula and those of the right operand of the operator, i.e., for $t := [\mathsf{c} \ @ \ t', \ldots]$ where c is a coordinate constant and $type(t') = \mathbb{E}$, constraints $t^- \leq t'^-$ and $t'^+ \leq t^+$ will be introduced.

Finally, terms using any relation involving type $\mathbb{E}$ in Table 2 can easily be made to use a conjunctive set of $\mathbb{C} \times \mathbb{C}$ relations; e.g., $t_1 := [\mathbf{b} \ \mathsf{t}_2]$ (t_2 is of type $\mathbb{E}$) is equivalent to $t_1^+ := \{< t_2^-\}$.

Constraint Re-interpretation

When following the instructions given above to translate TCNL formulae into constraints, we need to make sure every variable appearing in the scope of a constraint is of type $\mathbb{C}$, since the final DGSTP only allows variables of type $\mathbb{C}$. E.g., we should translate $t := \{< t'\}$ into a constraint $t^+ < t'^-$ (instead of $t < t'$) if both variables are of type $\mathbb{E}$. In general, when adding a constraint with scope $\{t, t'\}$, we should *re-interpret* the constraint based on $type(t)$ and $type(t')$ so no enumeration variable can slip into the scope. We list below the re-interpretations needed when $type(t) = type(t') = \mathbb{E}$ (a and b are integers and u is a temporal unit):

$$t < t' \rightarrow t^+ < t'^- \tag{5}$$

$$t \leq t' \rightarrow t^- \leq t'^- \textbf{ and } t^+ \leq t'^+ \tag{6}$$

$$t = t' \rightarrow t^- = t'^- \textbf{ and } t^+ = t'^+ \tag{7}$$

$$t - t' \in [a, b]u \rightarrow t^- - t'^+ \in [a, b]u \tag{8}$$

Note that (5) is essentially the $\mathbb{E} \times \mathbb{E}$ relation **b** , and (6) is equivalent to the disjunction (**b** ; **m** ; **o** ; **s** ; **f** ; =). Also, (8) in effect disallows overlapping between t and t', and with (7) they will force $t^- = t^+ = t'^- = t'^+$ when $a = b = 0$. This might seem strange, but note that (8) is added only when a quantitative constraint is introduced between t and t', which intuitively asserts that the two enumerations should never overlap.

If $type(t) = \mathbb{E}$ but $type(t') = \mathbb{C}$, and t' is not a bound variable, the re-interpretations are

$$t < t' \rightarrow t^+ < t'$$
$$t \leq t' \rightarrow t^+ \leq t'$$
$$t > t' \rightarrow t' < t^-$$
$$t \geq t' \rightarrow t' \leq t^-$$
$$t - t' \in [a, b]u \rightarrow t^- - t' \in [a, b]u$$

Note that $t = t'$ can never occur because of the type propagation described earlier. If the types of both variables are the same but t' *is* a bound variable, the re-interpretations include the above plus

$$t = t' \rightarrow t^- = t' \qquad \text{(if } t' \text{ is a starting bound)} \tag{9}$$

$$t = t' \rightarrow t^+ = t' \qquad \text{(if } t' \text{ is an ending bound)} \tag{10}$$

Note that (9) is equivalent to (t' **s** t) and (10) is equivalent to (t' **f** t).

Solving DGSTP

Solving a DGSTP is almost identical to solving a GSTP: we first run Procedure 2 to narrow down the domains of the variables, we then run a revised backtracking search based on Procedure 3 to assemble the solutions. This new search method uses both propagated constraints and *d-terms* to update variable domains: if a variable is assigned in a d-term, we can inversely infer the possible values for the other. Here we will only describe how the inverse inference procedure works for the major cases in the d-terms of operator @ and ∧.

Consider the d-term $d := \{|\mathsf{n}_\mathsf{x}|@[t_1 : t_2]\}$ where n is an integer constant and x is a unit or a coordinate. If only d is assigned, we can infer that $\{-|(\mathsf{n}+1)_\mathsf{x}|@\{<= d\}\} < t_1 \leq \{-|\mathsf{n}_\mathsf{x}|@\{<= d\}\}$ and $t_2 \geq d$; e.g., if d is Sunday, Jan 21, 2007 in $d := \{|2_{\{\mathsf{sun}\}}|@[t_1 : t_2]\}$, we should have Jan 7, 2007 $< t_1 \leq$ Jan 14, 2007 and $t_2 \geq$ Jan 21, 2007. If only t_1 is known, we can easily compute $d = \{|\mathsf{n}_\mathsf{x}|@\{>= t_1\}\}$ and $t_2 \geq d$. If only t_2 is known, than we should have $d = \{-|1_\mathsf{x}|@\{<= t_2\}\}$ and $\{-|(\mathsf{n}+1)_\mathsf{x}|@\{<= d\}\} < t_1 \leq \{-|\mathsf{n}_\mathsf{x}|@\{<= d\}\}$.

Consider the d-term $d := \{t \wedge \mathsf{c}\}$ where c is a coordinate constant. If only d is known, we can find $c_1 = \{-|2_\mathsf{c}|@\{<= d\}\}$ and $c_2 = \{|2_\mathsf{c}|@\{>= d\}\}$, and compute δ_1 and δ_2 as the distance between c_1 and d and between d and c_2, respectively. We can than infer that $\{d - |\lfloor \delta_1/2 \rfloor_\mathsf{u}|\} \leq t \leq \{d + |\lfloor \delta_2/2 \rfloor_\mathsf{u}|\}$ where u is the greatest lower bound of $\mathrm{g}(c)$.

E.g., if $d := \{t \wedge \{\texttt{sun}\}\}$ and d is Jan 21, 2007, we should have Jan 18, 2007 $\leq t \leq$ Jan 24, 2007.

We now present the revised search procedure:

Procedure 4 (Backtracking search for DGSTP)**.** Replace the step 4 in Procedure 3 with the following:

4′ For each possible granularity and variable $t_j \neq t_i$, update the domain of t_j based on the constraint from t_i to t_j in that granularity. Additionally, if t_i is a variable appearing in a d-term, update the domains of the other variables in the d-term using the inverse inference procedure. If any of the updated domains should become empty, return to 3.

Conclusion and Future Work

In this paper we have described a method for modeling a temporal scenario via Simple Temporal Problems with Mixed Granularities (GSTP). More specifically, we capture event-level semantics using a compact representation Time Calculus for Natural Language (TCNL) and construct a Dependency GSTP (DGSTP) from a set of TCNL formulae by: (1) creating variables for unresolved formulae and inferring the variable types; (2) translating the formulae into constraints whose scopes contain only coordinate variables; and (3) inferring variable granularities and quantifying qualitative constraints accordingly (Procedure 1). We can then propagate the constraints of the resulting DGSTP to narrow down the variable domains (Procedure 2), and search for the solutions by using a backtracking search method (Procedure 4). Finally, qualitative relations can be discovered by inspecting the propagated domains of the relevant variables.

There are at least three parameters in our method that are open for tuning. In the granularity inference procedure (Procedure 1) a default granularity is assigned to a variable if its granularity cannot be inferred from its context. In such cases if we know the typical durations of the events associated to the variable (such information is learned in (Feng, Mulkar, and Hobbs 2006)), we could assign a more sensible granularity to it. A second parameter is the large number we use to replace ∞ when quantifying a qualitative constraint (to avoid the theoretical infinite run-time of Procedure 2). Again we might be able to set this number based on the granularities or the other contextual information of the involved variables. Similarly, contextual information might also be useful in setting the width of the search window imposed in the d-term of the proximity operator $\wedge$ (Fig. 2). In summary, these questions can only be answered from an empirical study using real-world data.

References

Allen, J. F. 1984. Towards a General Theory of Action and Time. *Artificial Intelligence* 23:123–154.

Bettini, C.; Jajodia, S.; and Wang, S. X. 2000. *Time Granularities in Database, Data Mining, and Temporal Reasoning*. Berlin: Springer-Verlag.

Bettini, C.; Wang, X.; and Jajodia, S. 2002. Solving multi-granularity temporal constraint networks. *Artificial Intelligence* 140:107–152.

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

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Feng, P.; Mulkar, R.; and Hobbs, J. R. 2006. Learning Event Durations from Event Descriptions. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL)*, 393–400.

Florian, R.; Han, B.; Luo, X.; Kambhatla, N.; and Zitouni, I. 2007. IBM ACE'07 System Description. In *Proceedings of NIST 2007 Automatic Content Extraction Evaluation*.

Han, B.; Gates, D.; and Levin, L. 2006a. From Language to Time: A Temporal Expression Anchorer. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME 2006)*.

Han, B.; Gates, D.; and Levin, L. 2006b. Understanding Temporal Expressions in Emails. In *Proceedings of Human Language Technology conference - North American chapter of the Association for Computational Linguistics annual meeting (HLT-NAACL 2006)*.

Han, B. 2008. A constraint-based modeling of calendars. *Proceedings of AAAI Workshop on Spatial and Temporal Reasoning*.