Representing time intervals in the semantic network

Marecki J. Academy of Computer Science and Management
jmarecki@wsi.edu.pl

Abstract

A specific graph representation of the set of time intervals is presented in this paper. Time intervals arranged in a time graph can be be quickly accessed or referred to given the nature of the graph structure. Pseudo-code algorithms ment to derive the data from the time graph as well as adding new time intervals to it are explained. The possibility of attaching the family of time intervals to the semantic network structure is also discussed.

1. Introduction

Given the nature of each semantic network there are two important elements it consists of: objects and relations. Since each object / relation can be either obsolete, present of occurring in the future there is a need to extend the information saved in the semantic network knowledge base. Since both obsolete and future objects / relations may have the same name, each object / relation should be connected to its corresponding time interval (see fig. 1).

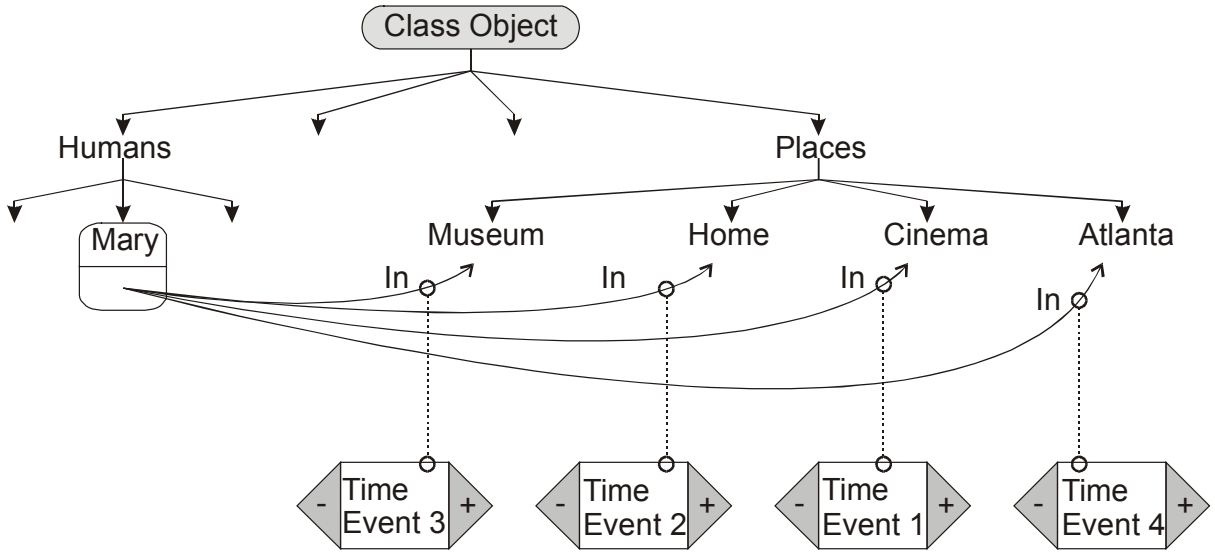

Figure 1 Connections between time events and the Semantic Network

The above figure shows the situation where Mary was (or is) at 4 different locations. If Museum, Home or Cinema can be found in Atlanta, it means then Mary can be at more than one location simultaneously.

Each time event is an object which consists of two values:

"–" the starting time of the event

"+" the ending time of the event.

Starting time should be always a constant time value, whereas ending time can be either a constant time value (if the event's ending time is known), or label "**Now**" (if the event

has started, and has not yet finished). Given the "Actual Time" integrated with the system, it can distinguish three categories for time events:

- **Obsolete Events** – have both "–" and "+" before the "Actual Time"
- **Actual Events** – have "–" before the "Actual Time", and "+" either after the "Actual Time" or marked as "Now"
- **Future Events** – have both "–"and "+" after the "Actual Time".

It is quite obvious, that for obsolete events the starting/ending times are constant time values. For the Actual Events, we may know their ending times or not. In the latter case, their "+" value is marked as "Now". For some future events, their "+" or "–" values may not be known. Future event becomes the Actual Event if Future Event's "–" is known to be before the "Actual Time".

The proposed notation for relations and time events is the following:

Given a relation A(x,y,z), we may directly influence its time values by writing $A(x,y,z).T_{event}^{-} = 13^{\underline{28}}$, $A(x,y,z).T_{event}^{+}$ = Now. Such a notation allows us to quickly check the time boundaries for each relation / object in the semantic network.

2. Time events representation

The first approach to find a data structure for time events is to use a basic set of independent elements. In most AI tools like for example prolog, time events (or situations – for situation calculus) used by time predicates are treated as any other object. The unification algorithm explores the whole knowledge base searching for objects which not only represent time events, but also have specific time properties. More advanced knowledge based systems use hashing tables for different types of objects and limit their search only to time associated events. However, the set of independent time events can be so huge that big common sense knowledge systems equipped with time reasoning mechanism would need enormous hardware resources to perform real time computations.

For time events e_1 and e_2, there is an important set of time relations:

$\forall e_1, e_2$ $\quad Meet(e_1, e_2) \Leftrightarrow e_1^{+} = e_2^{-}$

$\forall e_1, e_2$ $\quad Before(e_1, e_2) \Leftrightarrow e_1^{+} < e_2^{-}$

$\forall e_1, e_2$ $\quad After(e_1, e_2) \Leftrightarrow Before(e_2, e_1)$

$\forall e_1, e_2$ $\quad During(e_1, e_2) \Leftrightarrow e_1^{-} > e_2^{-} \wedge e_1^{+} < e_2^{+}$

$\forall e_1, e_2$ $\quad Overlap(e_1, e_2) \Leftrightarrow \exists e_3\ During(e_3, e_2) \wedge During(e_3, e_1)$

If we asked the first order logic system which of the above relations is satisfied for two known events e_1 and e_2, we would obtain the answer quickly. The situation would complicate dramatically if the system had to find for us two time events which satisfy one of the above relations. In the worst case scenario it would have to examine all time events to come up with the answer.

The answer to this problem may be to store some facts about the time in the lookup table and therefore speed up the searching of specific time events. Unfortunately, if we just wanted to store the relation *After* then for a set of *n* well ordered elements we would need *n!* entries in the lookup table. Since the other relations would also need a lot of space in the lookup table, this solution is good only for small knowledge bases.

The second approach to find an appropriate data structure for time events would suggest the implementation of a resizable array of events. This data structure would surely carry the information about the relations *Meet*, *Before* and *After*, but it would say nothing about the two others: *During* and *Overlap*.

The Semantic Network's approach towards the time representation problem is a **Time Graph** depicted in figure 2. Time graph connects different time events using two types of edges:

Next ↔ Previous	During ↔ When
If the event e_1 has a link *Next* to the event e_2, then $e_1^- < e_2^-$. *Next* and *Previous* are opposite: whenever there is the edge $e_1 \xrightarrow{Next} e_2$ there also must be the edge $e_2 \xrightarrow{Previous} e_1$ etc. Both relations keep track of the sequence of events.	If the event e_1 has a link *During* to the event e_2, it means that e_2 happens during e_1. In other words: $e_1^- < e_2^- \wedge e_2^+ < e_1^+$. *During* and *When* are opposite: whenever there is the edge $e_1 \xrightarrow{During} e_2$ there also must be the edge $e_2 \xrightarrow{When} e_1$ etc. Both relations create the hierarchical structure of the time graph.

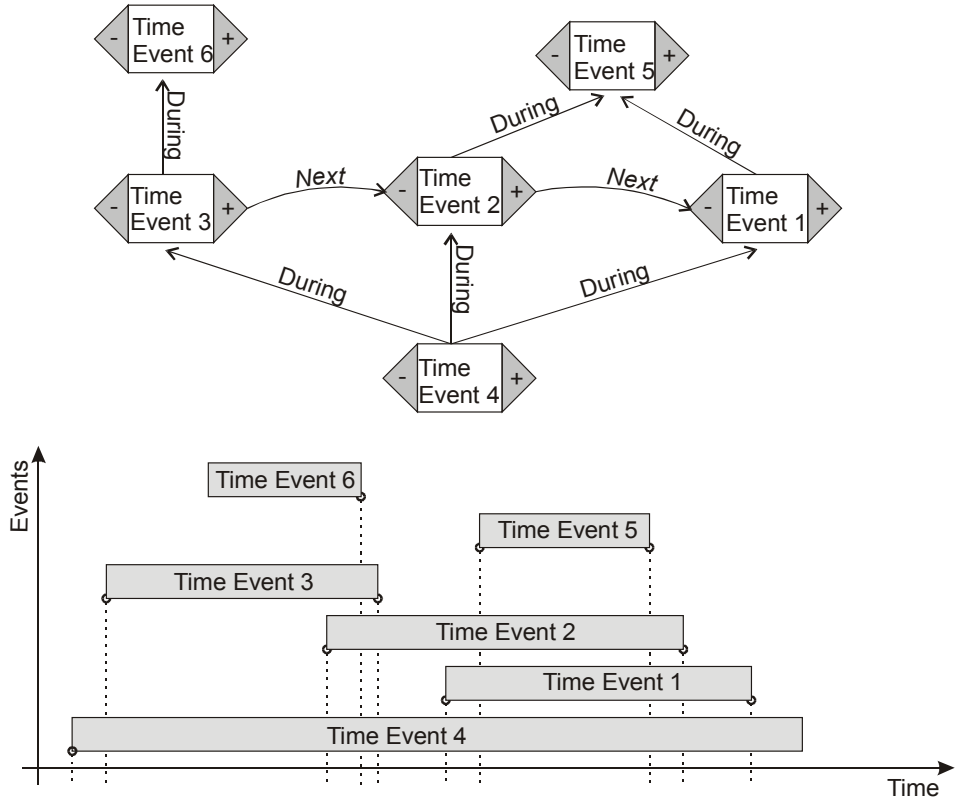

Figure 2 Time Graph and time axis

Relations Next/Previous can be called horizontal, and During/When – vertical.

The time graph can be understood as a pyramid. Its bottom is the longest, most general time event: Time. It is also the root of the time graph; all other events happen during the time root, and no event happens after or before it. The higher the pyramid we go, the shorter time events are found. If an event is at the top of the pyramid (there are many tops, confusing?) it means, that nothing happened during it. The general schema of a time event and its links is depicted in figure 3.

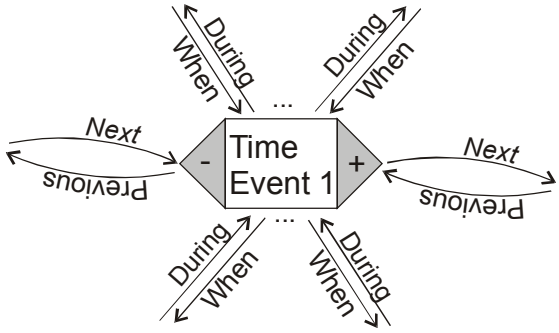

Figure 3 Time event as a Time Graph node

Relations *During* and *When* are transitive, therefore we can infer that all time events happen during root time event. Relations *Next* and *Previous* are also transitive, but in this case their transitivity is widened through the *During* relation: given the time events E_1, e_1, E_2, e_2, if $E_1 \xrightarrow{During} e_1$ and $E_2 \xrightarrow{During} e_2$ and $E_1 \xrightarrow{Next} E_2$ and $e_1 \neq e_2$, then it is obvious that $e_1 \xrightarrow{Next} e_2$.

Before we show the implementation of the Time Graph, a brief study of its advantages and disadvantages should be made. On the side of Time Graph's advantages the following points should be mentioned:

- Time Graph is a human readable representation of time events. Event big number of events can be represented in a transparent way.
- Since vertical relations apply to the Time Graph the hierarchical structure, the number of horizontal relations can be seriously reduced.
- For questions like: "When e_1 happened?" or "What happened during e_2" the answer is immediate. Moreover, because time events become more and more general when they approach the time root, the answers for the above questions are more and more general. Furthermore, if there is an event e_1 which happens during e_2 and e_3, then the fact that e_2 and e_3 overlap can be immediately inferred.

The biggest problem for the Time Graph is that it forces the hierarchical structure of time events, and therefore when a new event enters the graph it must be placed in the appropriate place updating some horizontal and vertical relations. In this section the event inserting algorithm will be depicted, along with a simple example of its execution.

The Time Graph carries some important properties:

1. **The first property of horizontal relations**. If there are time events e_1 and e_2 such that: $e_1 \xrightarrow{Next} e_2$, then not only $e_1^- < e_2^-$, but also $e_1^+ < e_2^+$. This fact is obvious, because if finishing time of e_1 was greater then finishing time of e_2, then since $e_1^- < e_2^-$ we would have $e_1 \xrightarrow{During} e_2$ which of course is false when $e_1 \xrightarrow{Next} e_2$.

2. **The first property of vertical relations**. Let us suppose there is a continuous set of sequential events: $\{e_i, e_{i+1}, \ldots, e_{j-1}, e_j\}$ and $e_i \xrightarrow{Next} e_{i+1} \ldots e_{j-1} \xrightarrow{Next} e_j$. If an event e_k is during e_i and e_j, then it must be also during $e_{i+1} \ldots e_{j-1}$. The proof of this property is indirect. Let us suppose that e_k is not during e_n ($i<n<j$). Since e_k is during e_i and e_j, then it is during the period $p = (e_j^-, e_i^+)$. Moreover, if e_n is after e_i and before e_j (property 1) then $e_n^- < e_j^-$ and $e_n^+ > e_i^+$, and therefore p is during e_n. Since e_k is during p and p is during e_n, we infer that e_k is during e_n – contradiction. Figure 4 shows the Time Graph which represents the first property of vertical relations.

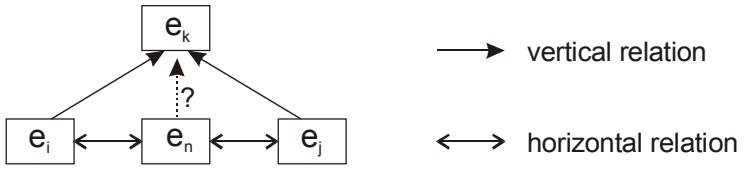

Figure 4 Improper Time Graph

3. **The second property of vertical relations**. If there are two sets of sequential events: $E_{ab}=\{e_a, e_{a+1}, \ldots, e_{b-1}, e_b\}$ and $E_{cd}=\{e_c, e_{c+1}, \ldots, e_{d-1}, e_d\}$, then for two events e_1 and e_2, if $e_1 \xrightarrow{Next} e_2$ and $E_{ab} \xrightarrow{During} e_1$ and $E_{cd} \xrightarrow{During} e_2$, then $a \le c$ and $b \le d$. Proving this property is the same as proving that e_2 is during e_d, for the graph depicted in figure 5.

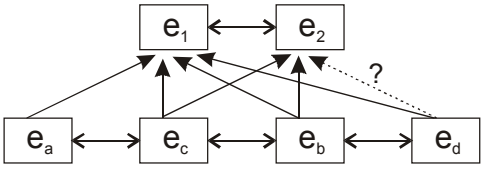

Figure 5 Second property of Time Graph vertical relations

In order to prove that e_2 is during e_d, we need to have: I) $e_2^- > e_d^-$ and II) $e_2^+ < e_d^+$.
Prove for I): If $e_1 \xrightarrow{Next} e_2$, then $e_1^- < e_2^-$. Moreover, if $e_d \xrightarrow{During} e_1$, then $e_d^- < e_1^-$. As a result we obtain $e_d^- < e_1^- < e_2^-$.
Prove for II): If $e_b \xrightarrow{During} e_2$, then $e_2^+ < e_b^+$. Moreover, if $e_b \xrightarrow{Next} e_d$, then $e_b^+ < e_d^+$. As a result we obtain $e_2^+ < e_b^+ < e_d^+$.

4. **The third property of vertical relations**. If there are time events e_1, e_2, e_3, e_4 such as: $e_1 \xrightarrow{Next} e_2$, and $e_1 \xrightarrow{During} e_3$ and $e_2 \xrightarrow{During} e_4$, but e_4 is not during e_1 and e_3 is not during e_2, then for set E_3 of all time events that happened during e_3, and for set E_4 of all events that happened during e_4 we have:

   $E_3 \xrightarrow{Next} E_4$ and
   $E_3 \cap E_4 = \varnothing$

This property is depicted in figure 6:

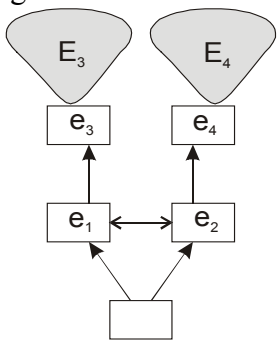

Figure 6 Third property of Time Graph vertical relations

5. **The second property of horizontal relations; graph levels connection**. One graph level consists of the nodes connected by the Next / Previous relations. The power of the Time Graph lies in the connections between time events belonging to different

graph levels. If a new time event happens during e_x and e_y, where e_x and e_y are located at different graph levels, then this new event inherits all time properties for e_x and e_y. Consequently this new time event splices two different graph levels into a single level connected by Next / Previous relations. It should be remarked, that time events inherit the properties of the events they belong to, not the graph level properties.

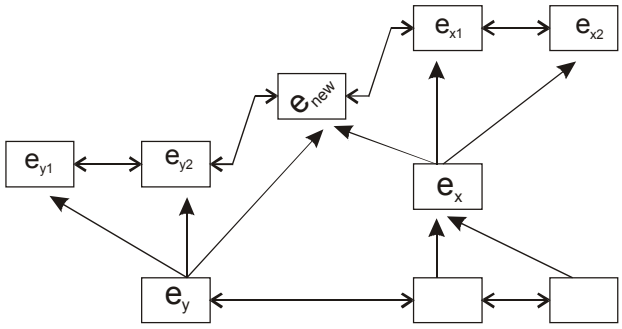

Figure 7 Time Graph level connections

Figure 7 shows an example graph where two different levels have been joined. The new, common level which consists of nodes e_{y1}, e_{y2}, e_{new}, e_{x1}, e_{x2} fulfills the properties 1, 2 and 3.

All time graph properties presented above help to understand the basis of the Time Graph functionality. Time Graph is hierarchical structure of graph levels. Whenever a new time event enters the graph, it is either inserted into a graph level, inserted between two different graph levels creating a common level, or put into a new graph level. In all cases, respective vertical relations link the new event with the events during which it happens.

Balanced Time Graph

The structure of the Time Graph along with all its properties provides one major advantage for the time representation: even with a huge number of time events, the system maintains its readability and simplicity. Instead of accumulating time relations in one node, and making the structure dense and complicated, the time graph expands nicely in horizontal and vertical directions. As it is shown soon, the Time Graph functions use only local parts of this graph, keeping all the time their high efficiency.

3. Inserting a new time event

The biggest problem for the proposed Time Graph is to create an algorithm which inserts a new event into it, and updates some graph relations if necessary. The algorithm **Insert(e_{new})** depicted below consists of four stages:

1. Given a new event e_{new}, find the set *P* of parent nodes (set of events during which e_{new} occurs). Due to the fact that relation *During* is transitive, only the shortest events during which e_{new} occurs will be in P.
2. Attach e_{new} to all the nodes from the set P using the relations *During* and *When*.
3. Create or update an existing graph level formed from the events which occur during the events from the set P, and insert e_{new} into that level.
4. Add vertical relations linking e_{new} with all events which happen during this event.

Stage 1

This stage should determine the elements of set P along the way of e_{new} from the Time Root to its appropriate location. The algorithm uses a temporary set T consisting of actual events during which e_{new} occurs. Initially, the set T has only one element: TimeRoot. Then the algorithm picks in a loop each element e from T, and explores the set E of all events which occur during e. For each of those events e', it checks whether e_{new} occurs during e'. If that is the case, e' is put into T. If there is an event e' during which e_{new} occurs, e is deleted from T, in the other case e rests intact. After deleting possible duplicates from T, the algorithm checks if T has been modified in the current loop. If yes, the algorithm continues its loop, if no it terminates.

The stage one algorithm is written below:

Function **DetermineP** returns P
    T' := {TimeRoot}; T := ∅
    While T≠T' do
        T := T'
        For each e∈T do
            E:= *Events during* e
            For each e'∈E do
                If e' $\xrightarrow{During}$ e_{new} then
                    T':=T' ∪ {e'} \ {e}
    Return T

Stage 2

This stage modifies the Time Graph by adding the following links:

$e \xrightarrow{During} e_{new}$ $\quad \forall e \in P$

$e_{new} \xrightarrow{When} e$ $\quad \forall e \in P$

Stage 3

At this stage, e_{new} will be put into an appropriate graph level. First, the set H of all events which happened during the events from P will be created. Then two events e_1 and e_2 closest to e_{new} from its both sides will be selected (if of course they exist). Finally appropriate horizontal relations will be added to the Time Graph.

Function **AddHorizontalLinks**
    E:= ∅; $e_1^- := -\infty$; $e_2^- := +\infty$
    For each e∈P do
        H := *Events during* e
        E := E ∪ H
    For each e ∈ E do
        If $e^- < e_{new}^-$ and $e^- > e_1^-$ then $e_1 := e$
        If $e^- > e_{new}^-$ and $e^- < e_1^-$ then $e_2 := e$
    If $e_1^- > -\infty$ then Add link $e_1 \xrightarrow{Next} e_{new}$ and $e_{new} \xrightarrow{Previous} e_1$
    If $e_2^- < +\infty$ then Add link $e_{new} \xrightarrow{Next} e_2$ and $e_2 \xrightarrow{Previous} e_{new}$

Stage 4

At this final stage, vertical relations linking e_{new} with all events which happen during it will be added. In order to achieve it, we explore via the *During* relation all events located on the graph level of e_{new}. If we find an event e which happens during e_{new}, the relation

$e_{new} \xrightarrow{During} e$ will added. To cut the branching factor of such a search, we would deal with events "on the left" and "on the right" of e_{new} separately.

Events "on the left" of e_{new} are located on the graph level of e_{new}, and start earlier than e_{new}. Suppose e_{left} is an event on the left of e_{new}. If there is an event e such that $e_{left} \xrightarrow{During} e$, then if that event starts after e_{new}, it must also be during e_{new}. If e starts and ends before e_{new}, then e and its descendants via the *During* relation cannot be during e_{new}. If e starts before e_{new}, but finishes after e_{new}, then although e cannot be during e_{new}, some descendants of e via the *During* relation can. The reasoning is analogous for the event "on the right" of e_{new}. The algorithm for stage 4 is presented below:

Function **AddVerticalLinks**

E:= ∅;
For each e∈P do
H := *Events during* e
E := E ∪ H
For each e∈E do
If $e^- < e_{new}^-$ then ExploreLeft(e)
If $e^- > e_{new}^-$ then ExploreRight(e)

Function **ExploreLeft**(e)

H := Events during e
For each h∈H do
If $h^- > e_{new}^-$ then
Add link $e_{new} \xrightarrow{During} h$
Add link $h \xrightarrow{When} e_{new}$
If $h^- < e_{new}^-$ and $h^+ > e_{new}^-$ then
ExploreLeft(h)

Function **ExploreRight**(e)

H := Events during e
For each h∈H do
If $h^+ < e_{new}^+$ then
Add link $e_{new} \xrightarrow{During} h$
Add link $h \xrightarrow{When} e_{new}$
If $h^+ > e_{new}^+$ and $h^- < e_{new}^+$ then
ExploreLeft(h)

4. Time Graph functions

Finally, we can demonstrate the efficiency of the Time Graph for the following questions: "What happened during the event E?", "When did E happen?", "What happened after E?" and "What happened before E?".

To make the demonstration more readable, we assume the presence of two functions: $BFS_{During}(E)$ and $BFS_{When}(E)$. BFS is a shortcut for **Breadth First Search** – the graph search algorithm, which explores the graph starting from the node N and returns a set of visited nodes. $BFS_{During}(E)$ explores the Time Graph via the *During* relation, and $BFS_{When}(E)$ explores the Time Graph via the *When* relation.

The answer for the first two questions is already done:

During(E) = $BFS_{During}(E)$

When(E) = $BFS_{When}(E)$

It can be remarked that the Time Graph structure along with Breadth First Search algorithm will return the more and more general answers for the first two questions.

Before we show how to answer the following two questions, we will introduce two auxiliary functions: LatestWhen(E) – which returns the latest event linked with E via the *When* relation, and EarliestWhen(E) - which returns the earliest event linked with E via the *When* relation. Functions A(E) and B(E) may return a set containing events which are during E, which must be deleted from the final return set.

Function **After**(E) returns P Return A(E) \ During(E)	Function **Before**(E) returns P Return B(E) \ During(E)
Function **A(E)** returns P E_1 := E ; P:= ∅ E_2 := Next(E) While exist E_2 do P:= P ∪ BFS(E_2) E_1 := E_2 E_2 := Next(E_1) Return P ∪ After(LatestWhen(E_1))	Function **B(E)** returns P E_1 := E; P:= ∅ E_2 := Previous(E) While exist E_2 do P:= P ∪ BFS(E_2) E_1 := E_2 E_2 := Previous(E_1) Return P ∪ After(EarliestWhen(E_1))

Example

We can demonstrate how the event insertion algorithm puts a new event to the Time Graph. Let us suppose we deal with a graph from Figure 2, and the new event e_{new} happens during e_6, e_3, e_2, e_4. After step 1, the set P will contain the elements e_2 and e_6. The new event will be therefore linked with *During / When* relations with e_2 and e_4. Consequently the graph level of e_{new} will contain only two sequential events: $e_{new} \xrightarrow{Next} e_5$. Since there is no event during e_{new}, the 4th step of the insertion algorithms will immediately terminate.

Now, if we asked the system a question "What happened after Time Event 3?" the *After* algorithm would go to e_2, explore it reaching e_{new} and e_5, before finally visiting e_1. Since there is no event after e_4, the algorithm would return a following set:

$P = \{e_2, e_{new}, e_5, e_1\} \setminus \{e_6, e_{new}\} = \{e_2, e_5, e_1\}$.

5. Bibliography

1. Cichosz, P. (2000) Systemy uczące się, Wydawnicta Naukowo-Techniczne, Warsaw.
2. Cleal, D. M., Heaton, N. O. (1988) Knowledge Based Systems: Implications for Human-Computer Interfaces, Ellis Horwood Limited
3. Cormen, H. T., Leiserson, C. E., Rivest, R. L., (1994) Introduction to Algorithms, Massachusetts Institute of Technology.
4. Figwer, J. (2001), Aplikacje Metod Sztucznej Inteligencji, scientific editor, Academy of Computer Science and Managemenet, Bielsko-Biała.
5. Flasiński, M. (1997) "Every man in his notions" or alchemists' discussion on artificial intelligence. In *Foundations of Science*, pages 107-121. Kluwer Academic Publishers, Dordrecht / London / Boston
6. Frąckiewicz Z., Marecki J.: Modelowanie matematyczne obsługi statków w porcie, Workshop on :" Management and Artificial Intelligence", Natioal Academy of Sciences of Ukraine, Institute of Information Infrastructure, Lviv, 2002, pp. 74-86.

7. Hendrix, G. G. (1975). Expanding the utility of semantic networks through partitioning. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 115-121, Tbilisi, Georgia.
8. Kayser, D. (1998) Representation de connaissances, Hermes, Paris
9. Kowalowski, H. (2000) Inżynieria Wiedzy, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.
10. Marecki, J. (2000) Struktury Danych, Wydawnictgwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.
11. Marecki, J. (2001) Metody Sztucznej Inteligencji, Wydawnictgwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.
12. Marecki, J. (2002) Grafy i Rekurencje, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.
13. Marecki J.: Modele grafów, Beskidzki Festiwal Nauki, Akademia Techniczno-Humanistyczna, Bielsko-Biała, 2001, tom : Badania Operacyjne, ss. 35-62.
14. Marecki J.: Algorytmy numeracji i grafów, Beskidzki Festiwal Nauki, Akademia Techniczno-Humanistyczna, Bielsko-Biała, 2001, tom : Badania Operacyjne, ss. 63-92.
15. Marecki J.: Teoria funkcji informatycznych, Beskidzki Festiwal Nauki, Akademia Techniczno-Humanistyczna, Bielsko-Biała, 2001, tom : Podstawy informatyki i Sieci Komputerowe, ss. 5 - 28.
16. Marecki J.: Metody sortowania, Beskidzki Festiwal Nauki, Akademia Techniczno-Humanistyczna, Bielsko-Biała, 2001, tom : Podstawy Informatyki i Sieci Komputerowe, ss. 29 – 54.
17. Marecki J.: Modelling of Decision Trees, International Conference on : „Inductive Modelling", National Academy of Sciences of Ukraine, Lviv, 2002, v.5, pp. 239-244.
18. Pilch-Kowalczyk, G. (2001) Internetowe Systemy Zarządzania, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.
19. Pilch-Kowalczyk, G. (2003) Komputerowe Systemy Gospodarki Elektronicznej. W *Internet w Społeczeństwie Informacyjnym*, Wyższa Szkoła Biznesu w Dąbrowie Górniczej.
20. Russell, S. J, Norvig, P. (1995) Artificial Intelligence, a modern Approach Prentice – Hall.
21. Shapiro, S. C. (1979). The SNePS semantic network processing system. In Findler, N. V., editor, Associative Networks: *Representation and Use of Knowledge by Computers*, pages 179-203. Academic Press, New York.