Research of an Event Process Designed for Dynamic & Voluminous Graphs to Compute Customized Itinerary

Kassara Omar, Cherrat Loubna, Essaaidi Mohammad
Abdelmalek Essaâdi University
LaSIT, Faculty of Sciences
Tetuan, Morocco
{okassara, ezziyyani}@gmail.com, cherrat81@yahoo.fr, essaaidi@ieee.com

ABSTRACT: *Processes of itinerary calculation in graphs have been, and still are, an interesting research topic. Despite the great interest that this topic has drawn in the research community, the proposed algorithms in the literature reveal certain limits to compute the itinerary in the graphs which parameters (vertices, edges) could be varying and could have a big graph order. The purpose main of this paper is to present the solution that resolve the issue in question and final results will be operated for the benefit of humans (included people with visual, auditory and mobile disabilities) to guide them in the presence of the unexpected of navigation that cause the variability of graph parameters. The presented solution can also be operated in a wide range of applications fields.*

Keywords: Routing Algorithms, Dynamic Graphs, Navigation, Graphs

Received: 11 April 2012, Revised 18 June 2012, Accepted 29 June 2012

1. Introduction

Nowadays, current navigation systems focus to offer to the users new complementary services. On one hand, the detection of the unexpected of navigation (closed paths, flooded areas, repairs…) is a new focal area of scientific research, and on the other hand, the generation of itinerary within a brief time of calculation, the both, are very stimulating challenges to achieve in the framework of this paper. Indeed, generation of itinerary in the presence of navigation contingencies requires establishing a process of route calculation being able to process graphs with variable parameters because the principle of classical algorithms isn't appropriated for such graphs.

2. Short state of art about classical routing algorithms

The majority of routing algorithms perform almost the same principle of calculation, except that the major difference resides in the ability of each one to process a dense graph. More specifically, some algorithms only compute shortest paths between two vertices (one-to-one), the both known in advance. This kind of algorithm is characterized by its speed of calculation (like A* algorithm). Some others algorithms like Dijkstra, find shortest paths between the input vertex and all remaining vertices (one-to-all), even, there are some generalized algorithms which are able to compute shortest paths between all input vertices and all output vertices (all-to-all). Certainly, the last kind of algorithms could process denser graphs, but it goes at the detriment of the processing speed. [1]

2.1 A* (star) algorithm

A* algorithm (one-to-one) provides the path between two vertices (input & output) the both known in advance. Due to its simplicity, it is often exhibited as a typical example of scheduling algorithms. It is famed in applications such as video games to favour time of calculation at the accuracy of results. [2]

Practically, A* is characterized among all others algorithm types by its natural high execution speed, because processing made in the graph doesn't exceed searching a shortest path between two vertices. On the contrary, A-star is often known by its instability to generate optimal results. [3]

2.2 Dijkstra

Dijkstra algorithm is categorized among one-to-all type and compute shortest paths between one vertex and all remaining vertices provided that the graph in question has edge's values greater than zero.

2.3 Bellemen-Ford

Bellemen-Ford algorithm (one-to-all) replace Dijkstra algorithm for graphs whose edges are negative values. The principle is broadly the same as Dijkstra except some slight modifications in its code.

2.4 Floyd

Floyd algorithm (all-to-all) can be considered as generalization of Dijkstra algorithm and can compute the shortest paths between all input vertices and all output vertices. With this algorithm, the graph is entirely processed, thus, it requires considerable processing time.

3. Limitation of Classical Algorithm

Classical algorithms go more & more slowly in terms of processing time while increasing the graph order. The present section examines & evaluates the maximum limit of the graph order for the three kinds of algorithm which permit an acceptable processing time at the user terminal. The determination of this limit will allow knowing later the sufficient number of parts to partition the graph. Thus, the process of itinerary calculation will be operated separately on the parts of the partition.

The followed figure represents the processing time depending on the graph order for the three kinds of algorithms. Tests were

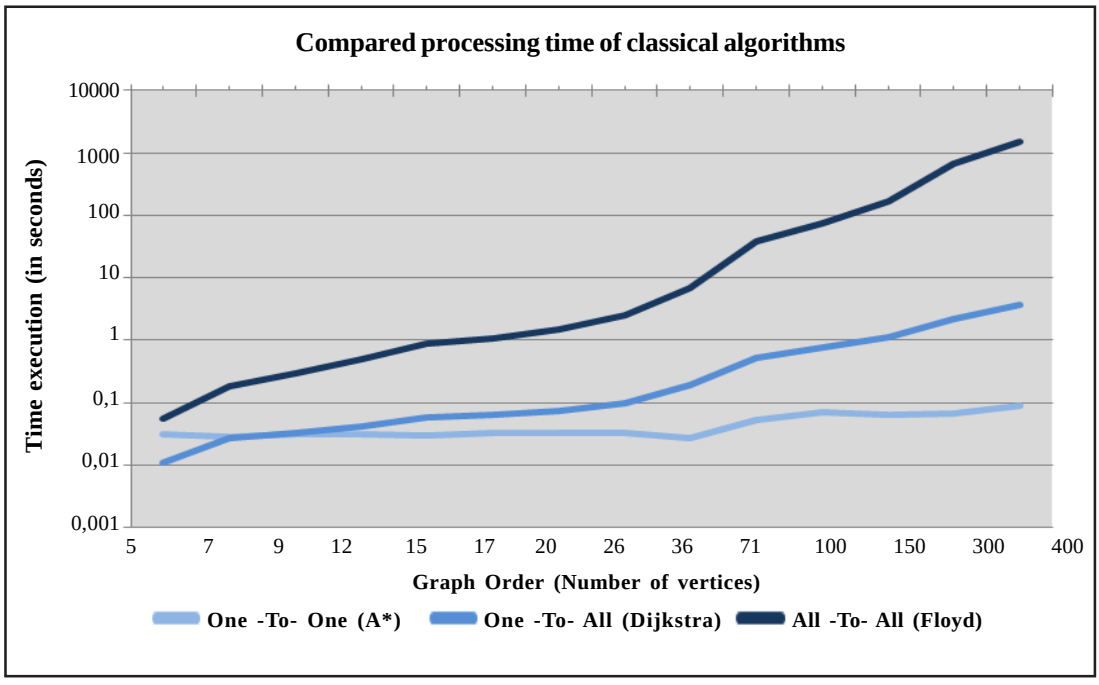

Figure 1. Speed of classical algorithms per graph ordrer

performed on a machine with frequency processor is like 1.6GHz. The pseudo-codes of algorithm were coded in "*Java Runtime Environment* ".

For A* algorithm, the graphic reveal constant evolution of its speed independently of graph order, this is explained by its non-influence by graphs with a big order the fact that it process only one shortest path in the graph. However, it presents some contradictions as case of the value recorded when the graph order like 36, that has been noted the minimum value of all the tests made on A*. One of its others contradictions consist in the time taken to compute shortest paths for graphs with small order whose it has been slower than Dijkstra even if the last one performs more & more treatments.

Regarding Dijkstra algorithm, apart from its remarkable advantage on A* for graphs with small order, it is characterized for graphs with big order, by a slight difference of processing time by comparing it at A*. That is an advantage because its capacity of processing (one-to-all) is much broader than A*.

Concerning Floyd algorithm as a reminder is a generalization of Dijkstra algorithm, goes more and more slower in its execution when the graph order increases, that is normal because it entirely processes the graph.

Concerning the issue in question, we have remarked that e.g. Dijkstra algorithm is going to cost 1 second to compute itinerary when the graph order tend to 150 vertices. After this limit, that become more slowly in terms of processing time. To remedy this, we have set 150 vertices as a maximum threshold & as a maximal size of each part of graph generated during the partitioning.

4. Graph partioning

Taking in consideration the last deduction, the waiting time to generate itinerary at the user is estimated to be less than 1 second for the graphs which order could reach 150 vertices. Concerning the graphs with order higher than 150 vertices, we consider dividing the graph into multiple connected subgraphs which the order of each one must be less than 150 vertices.

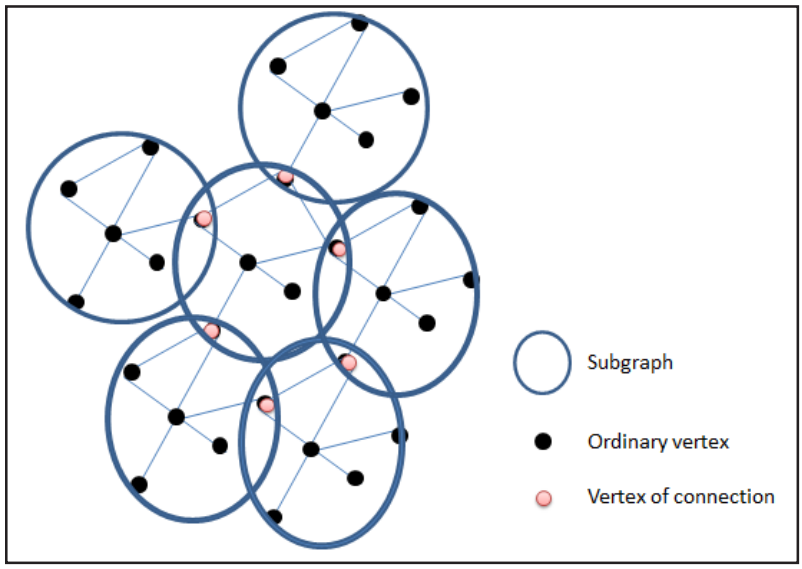

Figure 2. Partitionning of the graph into subgraphs

As techniques of partitioning, the both presented below, we think use the greedy graph growing algorithm as bisection technique, and the recursive bisection technique to obtain the desired k-partitioning.

5. Process of Itnerary Calculation for Graphs with Dynamic Parameters

For static graphs, the values of edges and the number of vertices are usually the same, thus, the calculation of the different itineraries can only be done once through Floyd algorithm (all-to-all) regardless the graph order. In this case, results can be

Greedy graph growing Algorithm

Procedure GGP (G = (S; A))
Take randomly one vertex $S_o \in S$
E $\rightarrow \{ s_o \}$
F$\rightarrow \{ s \in R \text{ such as } (s, s_o) \in A \}$
while weight (E) $< \frac{1}{2}$ weight (S) **do**
for all $s_i \in F$ **do**
F $\rightarrow \{s \in R \text{ such as } (s, s_i) \in A\}$
move s_i from F into E
end for
end while
end procedure

stored and the response time of system only requires the time of loading stored results. Thus, it will allow better use of system resources.

Taking in consideration the unexpected of navigation in graphs leads to consider modifications on the structure of the graph at its edges, therefore, the computation of itinerary have to be remade. Indeed, that looks difficult in the case when the graph has a big order. For example, the figure 1 reveals that remaking calculation when the order of graph like 71will cost an almost 50 seconds, thereby, this will dissatisfy users cause of the enormous waiting time. To remedy to this problem, the event process of route calculation designed for the graphs with dynamic parameters is presented in this paper, it consists in the realization of the next cycle:

Before initial state (inexistence of the unexpected of navigation in the ways of graph), the graph will be entirely processed by an all-to-all algorithm to generate all possible shortest paths could exist in the graph. Those results will be stored in the memory and

Recursive bisection technique
Procedure BissectionRec (G = (*S*; A), k)
P = {S_1,.., S_k} % Partition of G in k parts
Procedure Iterate(*S*', *k*', num)
If *k*' > 1 **then**
$k_1 \leftarrow [\frac{k'}{2}]$ % the most integer less than $\frac{k'}{2}$
$k_1, k' \leftarrow k_1$
$S_1', S_2' \leftarrow$ bisection (S', $\frac{k_1'}{k'}$, $\frac{k_2'}{k'}$)
Iterate (S_1', k_1', num)
Iterate (S_2', k_2', num + k_1')
else % no bisection to make, the part *S*' is put in P_k
$S_{num} \leftarrow$ S'
End if
End procedure
Iterate (*S*, *k*,1)
Return P_k
End procedure

Figure 3. Pseudo-code of partitioning algorithms

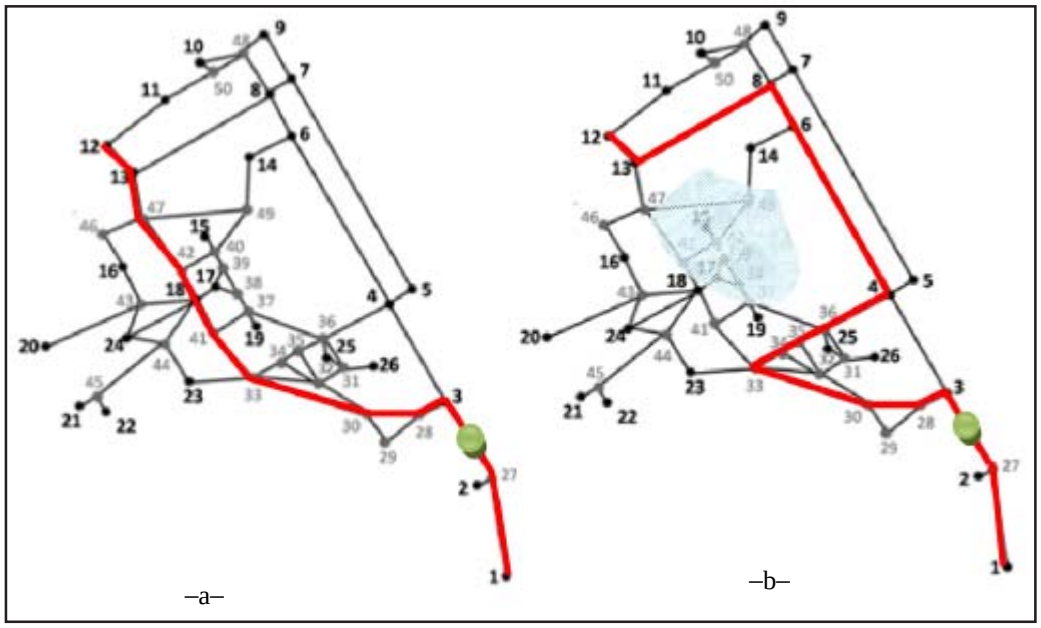

Figure 4. Exemple of desired result (-b-) when the unexpected of navigation is a flooded area

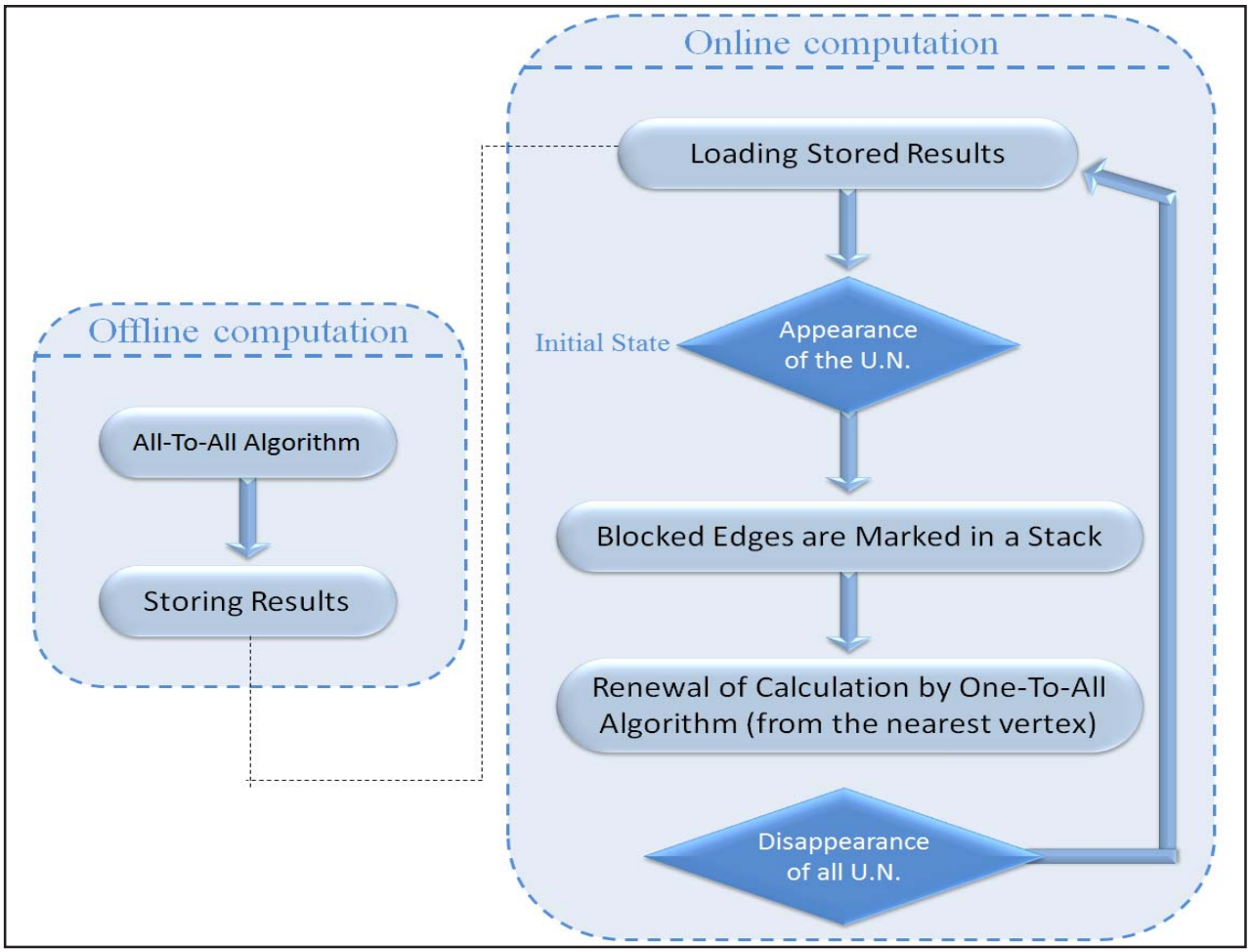

Figure 5. Diagram representing the process of itinerary calculation for graphs with dynamic parameters

Figure 6. The processing time of itinerary calculation after applying the event process

will be loaded in each use providing that this must occur only in the case where there are no changes in the parameters of the graph. The initial state is triggered by the apparition of an unexpected that it will marked in a stack. The latter serves as indicator of the apparition/disappearance of the navigation contingencies. When the stack is loaded at least by one element, the blocked edges are ignored in the graph and this time, the computation of itineraries will be performed through a one-to-all algorithm between the nearest vertex to the user and all remaining vertices. The complete disappearance of the unexpected of navigation indicates once again the load of stored results in the memory at the beginning of the cycle.

This technique will resolve the problematic that concern processing time performed by the process for graphs with big order. The following diagram describes the entire process of itinerary calculation for graphs with dynamic parameters.

6. Results & discussion

The figure 6 shows the results of comparing the processing time before and after applying the event process for several sizes of graph.

When there is no contingency (offline computation of the event process), the system resources are conserved & the system have only to load stored results of an all-to-all algorithm. The apparition of one contingency at least triggers the real time computation. As we can remark, following the different sizes of graphs, the processing time is practically the same or slightly less than classical one-to-all algorithm until we reach the threshold (150 vertices). From this limit, the processing time of the event process goes more & more constant approximately at a one second. The last value is the tolerable processing time we have achieved for any size of graph. Thus, the event process will allow better use of system resources and will generate quick results for any size of graph by using the partitioning techniques. In addition, thanks to the event process, the processing time is not influenced anymore by the apparition of the unexpected of navigation that causes generally the renewal of itinerary computation.

7. Conclusion & perspectives

We have presented in this paper the solution of computing optimal paths in graphs which parameters are variable. In addition, the technique of dividing a graph into subgraphs has permitted to reduce the processing time of routing algorithms in graphs which have a big order. Also, we have presented a process that computes itinerary in graphs which components could be varying.

In terms of perspectives, it will be studied how to generate itinerary in accordance with the user's profiles, we think establish a mathematical & parametrical model in order to attempt that aim.

References

[1] Bang-Jensen, J., Gutin, G. (2000). Digraphs: Theory, Algorithms and Applications. Springer.

[2] Maquin, D. 'Elements of graph theory and linear programmation', Éléments de théorie des graphes et programmation linéaire, Institut Nationale Polytechnique de Lorraine, École Nationale Supérieure d'Électricité et de Mécanique.

[3] Hart, P. E. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics* SSC4, 4 (2) 100–107.

[4] Balakrishnan, V. K. (1997). Graph Theory (1st ed.). McGraw-Hill.

[5] Berge, Claude. (1958) (in French). Graph theory and its applications, Théorie des graphes et ses applications. Dunod, Paris: Collection Universitaire de Mathématiques, II. pp. viii+277. Translation: . Dover, New York: Wiley.

[6] Biggs, Norman. (1993). Algebraic Graph Theory (2nd ed.). Cambridge University Press.

[7] Bollobás, Béla. (2002-08-12). Modern Graph Theory (1st ed.). Springer.

[8] Diestel, Reinhard. (2005). Graph Theory (3rd ed.). Berlin, New York: Springer-Verlag.

[9] Gosling, J., Mc Gilton, H. The Java language environment :A white paper. SunMicrosystems. Mountain View, CA.

[10] Peters, L. D., Saidin, H., (2000). IT and the Mass Customization of Services: the Challenge of Implementation, *International Journal of Information Management*, 20 (2) 103-119.

[11] Yagang Zhang, New frontiers in graph theory.

[12] Chein, M., Mugnier, Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs, Springer Verlag London, London, UK.

[13] Graph-based Knowledge Representation, Computational Foundations of Conceptual Graphs, Springer Verlag London, London, UK.

[14] Biggs, N., Lloyd, Wilson, R. (1976). Graph Theory 1736-1936, Clarendon Press Oxford.

[15] Steen editor, L. (1994). For All Practical Purpose, Introduction to Contemporary Mathematics 3ed. W. H. Freeman and Company, New York. QA7.F68 1994)

[16] Chartrand, Gary (1985). Introductory Graph Theory, Dover.

[17] Gibbons, Alan (1985). Algorithmic Graph Theory, Cambridge University Press.

[18] Mahadev, N.V.R., Peled, Uri, N. (1995). Threshold Graphs and Related Topics, North-Holland.

[19] Golumbic, Martin (1980). Algorithmic Graph Theory and Perfect Graphs, Academic Press.

[20] Harary, Frank, Palmer, Edgar, M. (1973). Graphical Enumeration, New York, NY: Academic Press. Mahadev, N. V. R.; Peled, Uri N. (1995), Threshold Graphs and Related Topics, North-Holland.