Exact and heuristic path planning methods for a virtual environment

Petr Brož[1,2]

Department of Computer Science and Engineering
University of West Bohemia
Pilsen, Czech Republic

Abstract

Path planning belongs to the best-known and well explored problems in computer science. However, in today's real-time and dynamic applications, such as virtual reality, existing algorithms for static environment are considerably insufficient and, surprisingly, almost no attention is given to techniques for dynamic graphs. This paper introduces two methods to plan a path in an undirected graph with evaluated nodes whose value can vary in time: a simple modification of Dijkstra's algorithm to find an optimal path while processing considerably less nodes than the standard Dijkstra's algorithm and a heuristic method to find a suboptimal path while processing even smaller amount of nodes. The heuristic was developed for a virtual reality path planning application but its use is more general.

Keywords: Path planning, Dijkstra, Virtual reality, Computer graphics

1 Introduction

Path planning belongs to the basic problems not only in the computer science. For that reason, there were developed many methods for determining a path that satisfies one or more optimality criteria according to a utilizing application. However, most of these path planning methods assume that the input structure, in most cases an evaluated graph, does not change its topology or rating. Nowadays, in time of the virtual and augmented reality, many dynamic applications arise and surprisingly, almost no attention was given to algorithms for the dynamic path planning, namely for fast, suboptimal solutions.

To follow requirements and context of our currently developed path planning system for the virtual reality, we focus on a slightly different definition of a dynamic graph. The overall system uses a 3D raster and an adaptive spatial structure (Figure 1) with rated nodes to enable finding a suboptimal path with the maximal clearance among all obstacles and moving threats. Therefore, we understand a dynamic graph as a graph with varying evaluation of the nodes. As for our virtual reality application the speed is more critical than optimality, we concentrate on suboptimal approach.

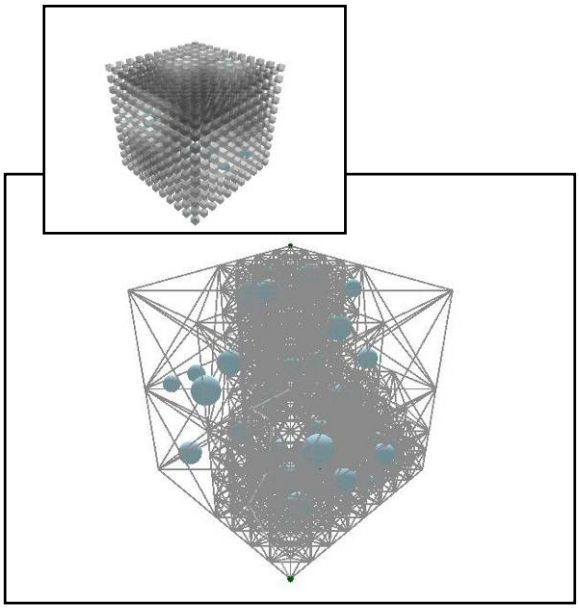

Figure 1: A visualization of data structures in our related VR project

In this paper, we present a modified Dijkstra's algorithm to plan an optimal path and a heuristic method to plan a suboptimal path in a dynamic graph without using any particular data structure. Our simple heuristic uses path information from the last graph traversal and provides suboptimal results in a notably shorter time than repeated optimal path computation. We compare this heuristic with the optimum on a dynamic graph with different ways of behavior.

[1] pebro@students.zcu.cz
[2] This project is supported by the Ministry of Education of the Czech Republic – project No. LC 06008

Section 2 describes the best known techniques for the static path planning and for the graphs with the possibility to insert or remove an edge. Section 3 presents the proposed algorithms and section 4 outlines experimentally gained characteristics and results. Section 5 then compares the presented algorithms with other techniques.

2 State of the art

Path planning represents a general task of finding an optimal path between two given spots in an abstract environment representation, in most cases in an undirected graph with weighted nodes or edges. Depending on a utilizing application, an objective of this task can be, e.g., the shortest path, the fastest path or the cheapest path.

First, we shortly summarize the standard algorithms for planning an optimal path in a static graph. The best known algorithm, the so called breadth-first search [3], finds shortest paths in any graph with unit weight of all edges with the overall running time O(|V|+|E|) where |V| means number of the vertices and |E| represents number of the edges in the graph. Dijkstra's algorithm [4] finds optimal paths in a more general graph whose edge lengths are positive integers with the running time O((|V|+|E|).log|V|). A simplified version of this technique is shown in an Algorithm 1 – ***distance*** property defines an integer distance between a current node and a starting node; ***previous*** property refers to a preceding node on the presently found path.

Finally, the shortest paths in a graph whose edges can be evaluated even with negative number can be found with the Bellman-Ford algorithm [3] with the time complexity O(|V|.|E|).

Next, we survey the algorithms for a graph with the possibility to remove or insert an edge. There are only few methods for planning an optimal path in such a graph [1, 4, 7, 9]. Most of these methods solve the so-called all pairs shortest path problem and are based on a special data structure. For a graph with unit edge costs, Ausiello and Italiano presented a data structure [1] which is able to find the shortest path between every pair in linear time O(|E|). However, maintaining the data structure requires the total time $O(|V|^3.\log|V|)$ in the worst case of insertion of at most $O(|V|^2)$ edges. Similar approaches [4, 5, 6, 11, 12] were presented to solve the all pairs shortest path problem, again by using a special data structure.

Input: the graph ***G(V, E)***,
the starting node ***S***,
the target node ***T***
Output: evaluation of nodes in ***G***
Auxiliary: ***P*** - set of complet. nodes
U,W – auxiliary nodes

- ***P*** ← empty set
- For each node in ***V***
  - Set ***distance*** to infinity
  - Set ***previous*** to null
- Set ***S.distance*** to 0
- Insert ***S*** to ***P***
- While not (***P*** contains all nodes)
  - From ***V***, find an edge between ***U*** ∈ ***P*** and ***W*** ∉ ***P*** such that ***U.distance+W.weight*** is minimal
  - Set value of the ***W.distance*** to ***U.distance+W.weight***
  - Set ***W.previous*** to ***U***
  - Insert ***W*** to ***P***
  - If ***W*** is the target node, break the cycle

Algorithm 1: A simplified graph processing using the Dijkstra's algorithm

3 The proposed methods

The presented methods come out from the standard path planning algorithms for static graphs. The heuristic is adapted to find a suboptimal path without using any special data structure in dynamic graphs with varying evaluation of their nodes. The input graph can have nodes of different degree with a positive integer evaluation which may vary in time. The evaluation changes may be randomly scattered over the whole graph but in the context of virtual reality applications, we expect the changes to be concentrated according to the movement of „threatening" avatars.

Our first technique was inspired by [10] and assumes that after the evaluation change, the existing path is affected mainly by the „near changes" of the graph evaluation. In addition, it assumes that with each iteration, the starting position – actually the momentary position of a moving avatar – gets nearer to the fixed destination point. Therefore, we use the standard Dijkstra's algorithm to find the path in the reverse direction – from the destination node to the current starting node. Let us denote this approach Backward Dijkstra. In comparison with the forward direction, fewer nodes are visited. Figure 2 shows the

algorithm planning a path from the starting node **S** to the destination node **T** at the beginning and after two iterations – the nodes visited during the graph traversal are highlighted with gray color.

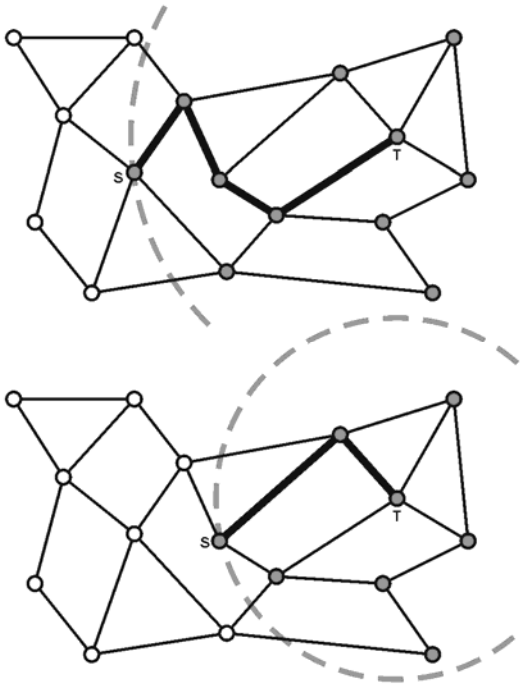

Figure 2: An example of path planning using the Backward Dijkstra's algorithm

Our second method uses the strategy that preserves as much of the original path as possible instead of finding the new optimal path after each change in the graph. It updates the last found path only in the nodes with changed evaluation. This resulting path obviously does not satisfy the optimality criteria. However, the mentioned virtual reality project and similar applications insist on the overall speed rather than on the path optimality. We call this approach Gaps filling method because of its corresponding behavior. In Algorithm 2, we show how the Gaps filling method refreshes the last found path. It can be seen that the technique stores a list of nodes on the presently found path together with their last known evaluations. In next iteration, it preserves the nodes with equal or better evaluation and spans the worse nodes with a new subpath. Figure 3 shows an original path between nodes **S** and **T** together with a new subpath through nodes **A**, **B** after the original path was disconnected in the middle node. In case that all nodes of the previous path have worse evaluation, a complete path from the starting node to the target node is found.

Input:	the graph ***G(V, E)***,
	the last found path ***W***,
Output:	the new suboptimal path ***W***
Auxiliary:	***N*** – set of new nodes

- Move the starting node from ***W*** to ***N***
- For each node ***X*** in ***N***
  - If ***X*** has equal or better evaluation or if it is the target point, find a new path from ***last(N)*** to ***X*** and insert the new nodes to ***N***

Algorithm 2: A graph processing using the gaps filling heuristic

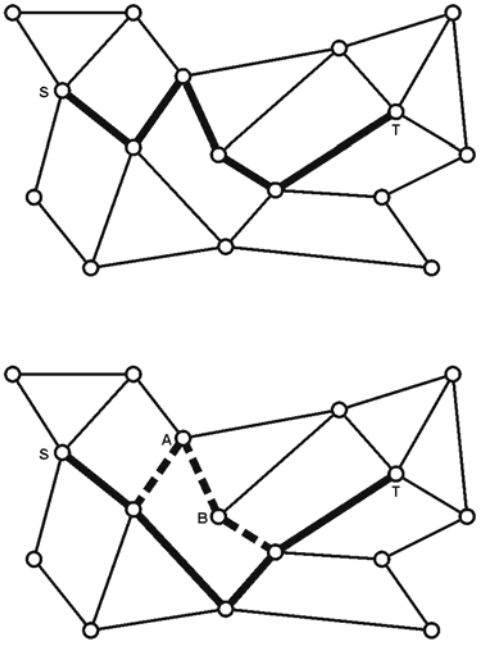

Figure 3: An example of path planning using the Gaps filling method

4 Experiments & results

For the purposes of comparison of the presented algorithms, a simple C# application was prepared to examine the behavior of the standard techniques and the proposed methods applied on an identical graph. The application was designed to enable easy extraction of the results and properties of each method. Figure 4 shows a screenshot of the application with paths generated by the tested algorithms and 3 additional windows showing the trends of the particular methods.

For now, the proposed techniques have not yet been used in the related VR project. However, we test these methods on similar datasets – we use graphs defined by an adaptive spatial structure similar to an octree with a weight value in each vertex of the smallest undivided areas.

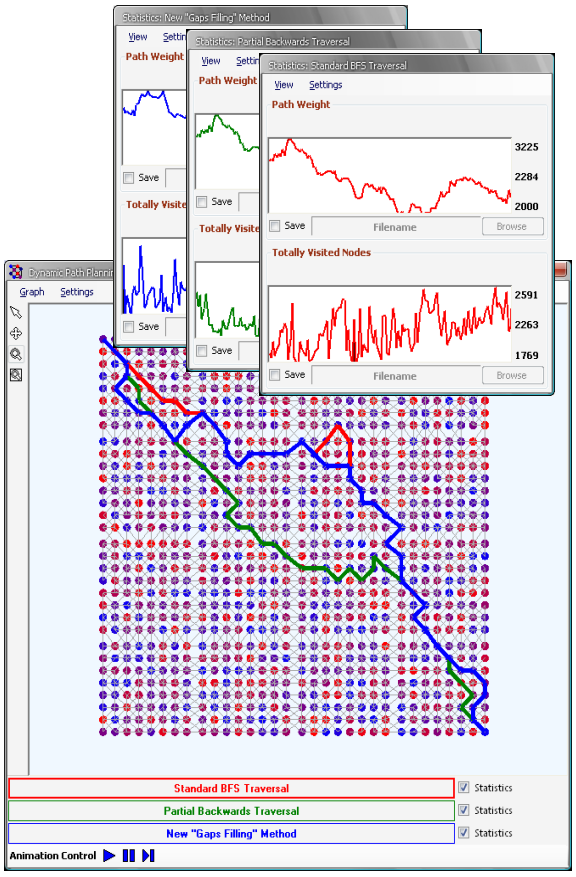

Figure 4: A screenshot of the testing application with 3 additional windows showing the trends of the particular methods

Again for the purposes of the virtual reality project, the testing data were a dense and nearly regular graph with an adjustable probability of the connection of adjacent nodes. Here, the 100% probability means that each node is connected with all neighbouring nodes, e.g., with 8 nodes within the scope of our testing planar graph. The dynamics of the graph is simulated by enabling to adjust the evaluation change probability for its nodes or to handle the evaluation changes directly through the user input. The compared path planning approaches were measured for the following cases:

- A graph with 32x32 nodes, connection probability 75% and random changes of the nodes evaluation with different probabilities (25%, 50% and 75%).
- A graph with 32x32 nodes, connection probability 75% and mouse driven changes of the nodes evaluation. Weight of each node was calculated according to its proximity to the mouse cursor.

The first case should provide a general idea about behavior of the methods, the second one simulates moving threats in a VR application.

As representative qualities of the examined techniques, the following properties were measured:

- An overall path weight – a sum of the weights of all nodes on the found path. An optimal path represents a path with the lowest overall weight.
- An amount of totally processed nodes – an amount of graph nodes visited by the particular graph traversal.

In order to concisely describe the suboptimal results of the proposed path planning approaches, an α-optimality term is used. The value $\alpha \geq 1$ stands for a ratio between suboptimal and optimal results. Following this definition, a 1,25-optimal path is 1,25 times longer or slower (according to the definition of the optimality) than the optimal path.

Random changes

For a constant probability of the evaluation change 50%, the proposed Gaps filling method finds a path that is 1,1-optimal on average (1,4-optimal in the worst case) and processes one third of nodes processed by the standard Dijkstra's algorithm. Figure 5 displays the overall weights of the paths found by the examined path planning methods. It can be seen that the results for the standard Dijkstra's algorithm and for the Backward Dijkstra are identical. Figure 6 then shows the amount of visited nodes.

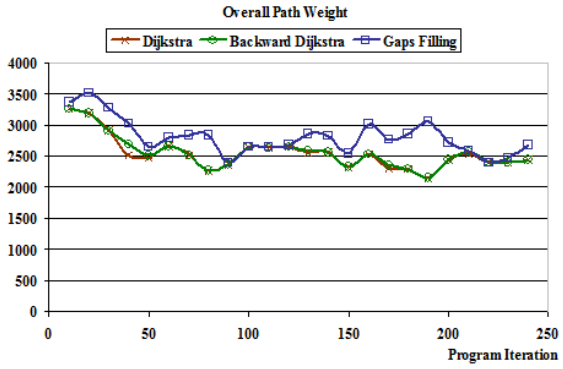

Figure 5: Overall path weight during the program run for each presented method

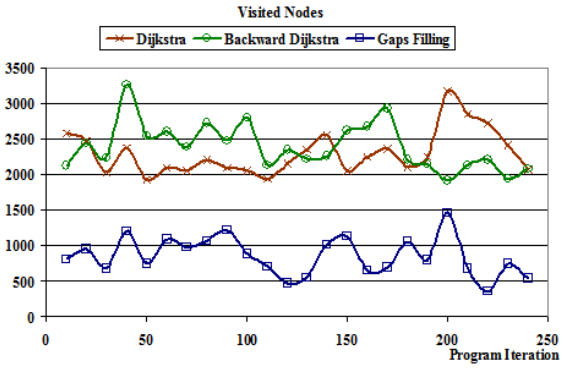

Figure 6: Total number of visited nodes during the program run for each presented method

It can be seen that Backward Dijkstra often visits more nodes than the standard Dijkstra's algorithm which was a bit surprising result for us at first but the explanation is simple – total number of visited nodes obviously depends on the initial node for the graph traversal. The probability of the evaluation change highly affects the results of the presented heuristic algorithm. For higher probabilities, more nodes can deteriorate their evaluation and the last found path has to be recomputed in more segments but the suboptimality is for that reason closer to the optimum.

Change probability [%]	Processed nodes [%]	Overall path weight [%]
25	13,70	126,26
50	37,75	115,76
75	85,67	108,67

Table 1: Different properties of the Gaps filling for the particular change probabilities

Another important factor for the quality of the Gaps filling heuristic is the amount of nodes in the graph. The higher is the amount of nodes the higher savings can be achieved by this method. On the other hand, there is a bigger chance of rising of a new optimal path that will not be registered by the presented Gaps filling method.

Driven changes

During the mouse driven changes in the graph evaluation, the gaps filling method finds a 1,3-optimal path on average and traverses one sixth of nodes processed by the Dijkstra's algorithm. Figure 7 shows an example of the testing application based on a user interaction to change the evaluation of graph nodes.

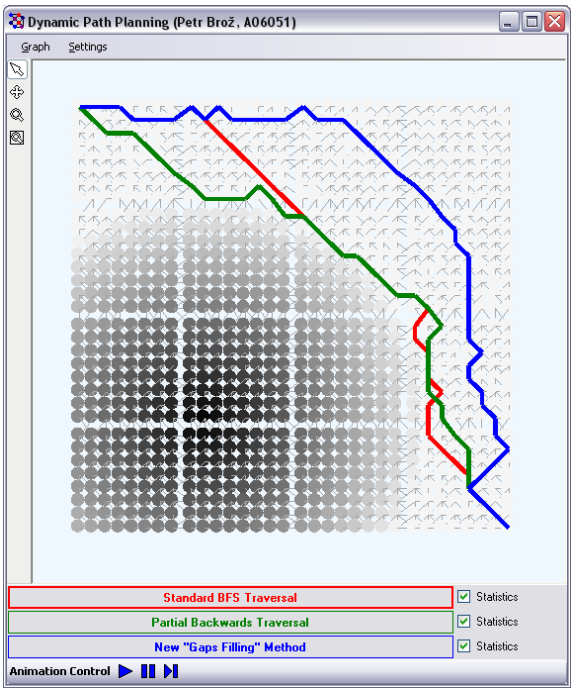

Figure 7: An example of the graph with mouse driven evaluation of its nodes

Figure 8 shows the overall path weights during the program run. Again, the results for the standard Dijkstra's algorithm and the Backward Dijkstra are identical. Figure 9 displays the totally visited nodes during the program run for each presented method. The results presented in figures 8 and 9 are measured for specific evaluation changes – all nodes in the graph are continuously evaluated according to their proximity to a potential point which is moving from the bottom-left corner to the upper-right corner of the area covered by the graph.

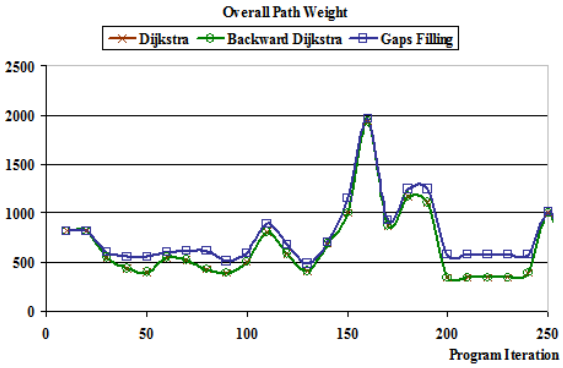

Figure 8: Overall path weight during the program run for each presented method

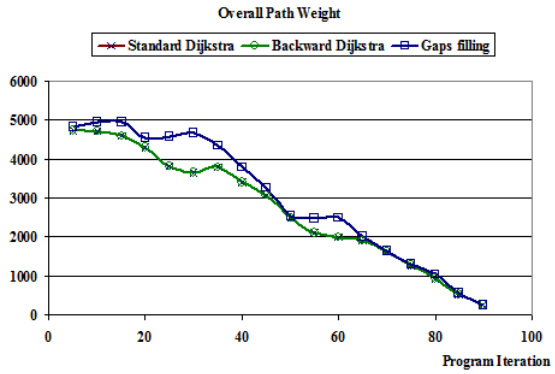

Figure 10: Overall path weight for the case of moving starting point

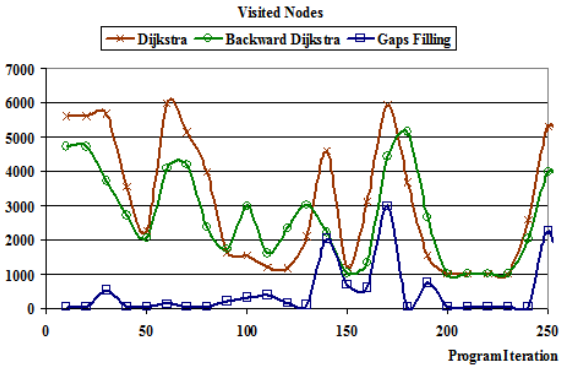

Figure 9: Totally visited nodes during the program run for each presented method

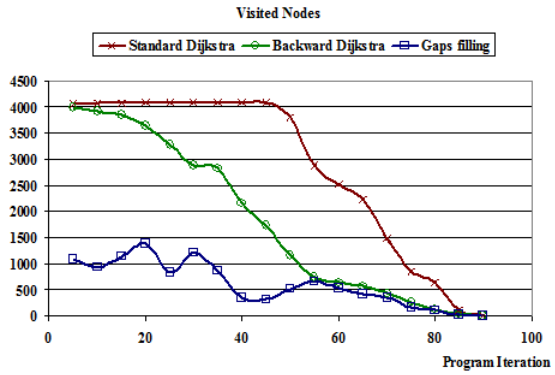

Figure 11: Visited nodes for the case of a moving starting point

To show the difference of the Backward Dijkstra's algorithm, we offer results of the measured techniques in case of a moving starting point. In each iteration, the starting node of the path moves to its successor – it continuously approaches the target node. Figure 10 again shows the overall path weight in about 90 iterations. It can be seen that the results of the standard Dijkstra's algorithm and the backward Dijkstra's algorithm are equal while the weight of a path found by the gaps filling approach stays a bit higher. Figure 11 then shows a number of processed nodes during the same measurement. There is a noticeable difference between the amount of nodes processed by the standard Dijkstra's algorithm and the backward Dijkstra's algorithm. In this example case, the backward approach finds an optimal path by visiting 55% of nodes processed by the standard technique and the gaps filling approach finds a 1,1-optimal path by processing 25% of nodes.

5 Conclusion & future work

The Backward Dijkstra's algorithm provides better results when used for planning an optimal path from a moving initial position. In these cases, it can be a suitable alternative to the standard Dijkstra's algorithm.

The proposed Gaps filling approach provides 1,1-optimal results on the average for the graphs with randomly changing evaluation and 1,3-optimal results on the average for the user driven changes in the graph. When applied on an adaptive mesh from the mentioned virtual reality project, the described approach found a 1,3-optimal path on the average by visiting 26% of the nodes processed by the standard Dijkstra's algorithm.

Due to the local updates of the last found path, the presented method does not perceive a possibly much better path, which may arise far from the last found path. A possible case may occur if there is an evaluation improvement around the last found path – the gaps filling method responds only to an aggravation of the nodes evaluation and it can miss the better path as well. However in the virtual reality, the behavior of the avatars tends to stay on

the previously found path rather than walk around in a completely new way.

In the future, we will incorporate the proposed approaches into the mentioned VR system [2] and we would also like to apply these methods in the project related to searching and visualisation of paths in chemicals (Masaryk University, Brno, Czech Republic). Figure 12 – a screenshot from this project [8] – shows a tunnel in protein molecules with a path found by the standard Dijkstra's algorithm.

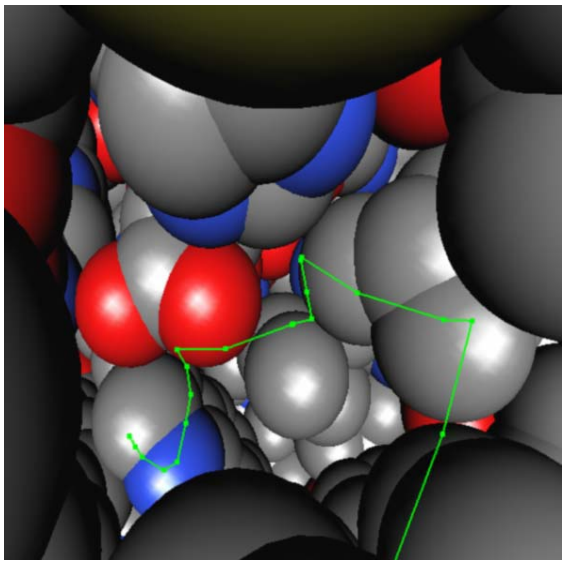

Figure 12: An example of a tunnel in protein molecules [8]

Acknowledgements

I would like to thank to Dr. I. Kolingerová from the University of West Bohemia, Pilsen, Czech Republic, for supervision and help with the paper preparation. I would also like to thank to Prof. Z. Ryjáček and Dr. F. Vávra from the same university for their consultations and valuable suggestions.

References

[1] G. Ausiello, G. F. Italiano. *Incremental algorithms for minimal length paths*. J. Algorithms, 1990.

[2] P. Brož. *Path planning in combined 3D grid and graph environment*. Proc. Central European Seminar on Computer Graphics, 2006.

[3] S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani. *Paths in graphs* (http://www.cse.ucsd.edu/users/dasgupta/mcgrawhill/chap4.pdf).

[4] S. Even, H. Gazit. *Updating distances in dynamic graphs*. Methods of Operations Research, 1985.

[5] G. F. Italiano. *Amortized efficiency of a path retrieval data structure*. Theor. Comput. Science, 1986.

[6] G. F. Italiano. *Finding paths and deleting edges in DAG*. Inform. Proc. Letter, 1988.

[7] P. N. Klein, S. Sairam. *Fully dynamic approximation schemes for shortest path problems in planar graphs*. Proc. 3rd Worksh. Algorithms and Data Structures, 1996.

[8] P. Medek, P. Beneš, J. Sochor. *Computation of tunnels in protein molecules using Delaunay triangulation*. Proc. Winter School of Computer Graphics, 2007.

[9] H. Rohnert. *A dynamization of the all pairs least cost path problem*. Proc. 2nd Annual Symp. on Theoretical Aspects of Comp. Science, 1985.

[10] Z. Ryjáček. Personal Communication, 2006.

[11] P. M. Spira, A. Pan. *On finding and updating spanning trees and shortest paths*. SIAM, J. Comput., 1975.

[12] R. E. Tarjan. *Depth-first search and linear graph algorithms*. SIAM, J. Comput., 1972.