Narrow Passage Problem Solution for Motion Planning

Jakub Szkandera[1(✉)], Ivana Kolingerová[1,2], and Martin Maňák[2]

[1] Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, 30614 Plzen, CZ, Czech Republic
{szkander,kolinger}@kiv.zcu.cz

[2] New Technologies for the Information Society, Univerzitni 8, 30614 Plzen, CZ, Czech Republic
manak@ntis.zcu.cz

Abstract. The paper introduces a new randomized sampling-based method of motion planning suitable for the problem of narrow passages. The proposed method was inspired by the method of exit points for cavities in protein models and is based on the Rapidly Exploring Random Tree (RRT). Unlike other methods, it can also provide locations of the exact positions of narrow passages. This information is extremely important as it helps to solve this part of space in more detail and even to decide whether a path through this bottleneck exists or not. For data with narrow passages, the proposed method finds more paths in a shorter time, for data without narrow passages, the proposed method is slower but still provides correct paths.

Keywords: Motion planning · Sample based algorithms · Rapidly exploring random tree · Narrow passage · Bottleneck

1 Introduction

A fast and reliable solution of the motion planning problem - to find a collision-free path for an agent (an abstraction of a moving object) between at least two spots in an environment filled with obstacles is needed in many areas (e.g. robotics, autonomous vehicle navigation, computational biology, etc.). For simple-shaped agents it is possible to use geometrical methods (e.g., Voronoi diagrams to compute centerlines). However, when the navigation of more complex or even a flexible agent is necessary, the geometric methods on themselves are not strong enough any more.

The concept of configuration space is used to interpret motion planning. The configuration space is a set of all existing configurations, where one configuration represents the specific position and rotation of the agent. These properties together form degrees of freedom. As the number of degrees of freedom increases, the dimension of the problem to be solved as well as its complexity increase. For example, the agent configuration in 3D space may be a six-dimensional vector

© Springer Nature Switzerland AG 2020
V. V. Krzhizhanovskaya et al. (Eds.): ICCS 2020, LNCS 12137, pp. 459–470, 2020.
https://doi.org/10.1007/978-3-030-50371-0_34

describing its position (3 vector components) and rotation (3 vector components) in the configuration space. A configuration space then contains a huge number of configurations that cannot be processed in a reasonable time, and, therefore, randomized sampling-based methods are used.

Randomized sampling-based methods [7,10] randomly select configurations to subsequently test for collisions. If the tested configuration is collision-free, it is added to a path-finding structure (roadmap). Otherwise, the configuration is rejected and the method creates a new random configuration. The roadmap approximates the free regions of the configuration space and enables to search a path with graph-based path planning methods. In many cases, this is a very effective way to find a passage through the environment in a reasonable time. However, the randomized sampling-based algorithms have problems with narrow passages; since the methods randomly sample space, it is very difficult to hit a sample inside a narrow passage.

This article proposes a solution to the narrow passage problem, based on the combination of Voronoi diagrams and randomization, using the idea of so-called exit areas [12]. Exit areas (exit regions) were originally proposed in the context of protein molecular models. They show the exits from deeply buried empty cavities. However, the same idea can be used in other motion planning applications as well. In general, exit areas capture the exact positions of the narrow passages, which greatly contributes to eliminating the biggest weakness of randomized sampling-based algorithms. Thanks to this knowledge, it is possible to sample the position of the narrow passage in detail and it is even possible to decide whether there is a passage through the narrow passage or not.

The paper has the following structure. Section 2 contains a description of existing motion planning methods that are useful for navigating an agent through the configuration space. Section 3 focuses on a detailed description of the proposed solution for motion planning in narrow passages. It also includes an algorithm description and improvement for the sample based algorithms to increase its acceptance of samples. Section 4 presents experiments and results on the real biomolecular data and artificially generated data. Section 5 concludes the paper.

2 Related Work

The widely used randomized sampling-based algorithms can be divided into two groups - algorithms based on Probabilistic Roadmaps (PRM) [7] and Rapidly Exploring Random Tree (RRT) [10]. The original PRM algorithm [8] builds a graph over the explored parts of the environment. This approach has two phases. First the random samples are generated and tested for collisions. The second phase tries to connect the close samples with an edge if possible. The possibilities of implementing these procedures are stated in [5] which also compares these procedures in detail. A sufficient input for the PRM algorithm is a set of obstacles. The knowledge of the start and the goal configuration is not required by the algorithm itself, but their knowledge can be used in some sampling heuristics.

There are two problems in the PRM algorithm. The first one is called boundary value problem, when it is necessary to solve whether the movement of the

agent from the first state to another is possible. It rises up when connecting the two given configurations. This problem can be difficult to solve under motion constraints, so PRM is primarily used in motion planning without motion constraints. The second problem is the already mentioned narrow passage problem. It can be solved (or at least approximated) by generating random samples close to obstacles or around medial axis of the environment [9] for low-dimensional configuration spaces.

sPRM [7] is a simplified version of the Probabilistic Roadmaps algorithm. Rather than for practical use it is used for the analysis of follow-up algorithms. On the other hand, unlike the previous method, the sPRM finds the path asymptotically optimal. PRM* [6] is another possible variant which uses a heuristic function to minimise roadmap lengths. This is an algorithm based on sPRM with the only difference that potential samples for interconnection are selected from the neighborhood with radii $r > 0$.

The Rapidly-exploring Random Tree (RRT) [10] belongs to the second mentioned group. RRT has been designed for use in models with a number of complex physical constraints. A tree is generated instead of the graph, which simplifies the path planning part. Next it incrementally grows towards unexplored regions of the configuration space. In addition, it also needs the start configuration. The main RRT computation is as follows. First of all, the tree structure t_{main} is initialized and then the algorithm repeats three main steps in cycle. The first step is to randomly generate a new sample in the configuration space. Steering the new sample close to the nearest tree list of the tree t_{main} is the second step. The last step is to check the collision of the new sample. If the sample is collision-free, it is added to the tree t_{main}, otherwise it is rejected. As RRT is the base of our proposed solution, it will be explained in more detail in the next section.

In the case of RRT algorithms, there are a number of modifications that solve motion planning in general or for specialized problems. RRT* [6] is an algorithm that uses a heuristic function to find the optimal solution. The extension for dynamic environment is solved by RRT^X [13]. There are plenty of other modifications but all of them suffer from the narrow passage problem like PRM algorithms. Guiding the tree along a precomputed path by geometry-based methods [14] is a possible way how to solve the narrow passage problem.

The motion planning methods can also be applied to other areas than to the navigation of mechanical objects. The motion planning in molecular simulations, where we have found inspiration for our proposed solution, is also a very important topic of research. The problem is, e.g., a navigation of the so-called ligand in a protein. Probabilistic Roadmaps can be used to sample the configuration space of the protein [1] in order to speed up molecular dynamics simulations but atoms bounds of the ligand lead to sampling in the high-dimensional configuration space.

The RRT algorithm is an appropriate planner also for a flexible ligand [3]. The ability to generate new configurations greatly affects the performance of the RRT. The high-dimensional space problem can be time consuming and the ML-RRT (Manhattan-like RRT) copes with this problem [4]. The method was

further extended for flexible ligands [3]. Moreover, the high-dimensional space roadmap can be projected back to 3D space [2].

Exit areas (exit regions) were originally developed for cavities in protein models [12]. Cavities and their exits are computed from a Voronoi diagram. The graph of Voronoi vertices and edges captures possible trajectories of collision avoiding spherical probes among spherical obstacles (the atoms of a protein model). When the probe is located in a cavity, it cannot get to the exterior space without a collision unless the probe radius is reduced. The exact value to which the probe radius must be reduced and the exact position where the probe will be located (the primary exit location) can be computed by analyzing the graph of Voronoi vertices and edges. The edge on which the probe could escape is then disabled and the process is repeated to discover remaining exits. Exit areas (exit regions) are then constructed as the groups of intersecting probes in the exit locations.

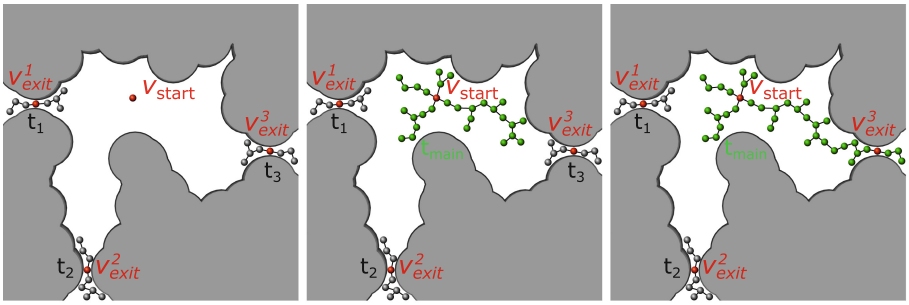
3 Proposed Solution

The proposed solution idea is based on incorporation of exit areas [11] into a randomized sampling-based algorithm where it helps to detect narrow passages in the data. The exit area in this context is the area that contains the nearest collision-free surrounding of the narrow passage. The position inside the exit area (more precisely in the middle of the narrowest passage) is called the exit point. Now let us first recall the original RRT algorithm in detail, see [10], and then the proposed modifications.

Each RRT algorithm contains three identical steps that have been used since the introduction of the original RRT algorithm [10], only the techniques used to solve these individual steps differ. After the tree structure t_{main} is initialized, three steps, which are repeated until the computation is finished (exceeding the maximum iteration, finding the path, etc.), are started. Generating the new random sample in the configuration space is the first step. The second step is steering the new sample close to the nearest tree list of the tree t_{main}. The third and final step is to check if the new sample is colliding with surroundings. If the sample is collision-free, it is added to the tree t_{main}, otherwise it is rejected.

Two modifications of the RRT algorithm are needed in the proposed solution. First, the exit points are computed before the main RRT cycle is started. Exit points tell us the exact position of the most problematic places (narrow passages) in the data, which can be then focused on (e.g., more detailed sampling of narrow passages can be done). In any case, the knowledge of the exact position of the narrow passage is absolutely crucial. The second change is the correction of the rejected samples. If the agent has a small collision with the surroundings and the sample would be rejected, we try to move the agent into the free space. This modification is included in the third step of the RRT algorithm (the collision test of the sample).

Let us illustrate the main idea of the proposed solution. First, the proposed solution finds exit points $v^i_{exit}, i = 1, 2, 3$, whose surrounding is then sampled



(a) (b) (c)

Fig. 1. (a) Sampling around exit points, (b) Sampling around the starting position v_{start}, (c) Merging exit point tree t_3 into the main tree t_{main}

by a randomized sampling method (Fig. 1a). When all narrow passages are processed, sampling from the starting position v_{start} is initiated (Fig. 1b). This sampling from the starting position v_{start} repeats until we find the pass through the sampled data. During this sampling, the tree t_{main} can reach some smaller tree t_i that has been created when the method was sampling the exit point surroundings. In this case, we connect the smaller tree t_i to the main tree t_{main} (Fig. 1c). This approach has two obvious advantages. The main tree t_{main} grows by already created samples to cover more space, and above all, it easily overcomes the narrow passage.

The whole process is shown in detail as Algorithm 1. Exit points are computed at (Algorithm 1, line 1). Each exit point v^i_{exit} is basically a new starting point for calculating the RRT algorithm, so we run it for each v^i_{exit} (Algorithm 1, line 5). It is important to note that we will let the RRT algorithm run only a limited number of steps c_{max}, because we only need to sample surroundings of a narrow passage to find if there is a possibility to get the agent through. Finally, the main RRT algorithm computation is run from the input starting point v_{start} (Algorithm 1, line 7), this run is not limited by the number of samples.

Now let us focus on the modification of the RRT algorithm itself (Algorithm 1, lines 9–19). For a given starting point v_{start}, which is the root of our main tree t_{main}, we will try to find a collision-free position (Algorithm 1, line 11). Then the main loop of the algorithm, which contains the above mentioned steps (sample, steer, connect), runs. The most important is the modification at the end of this loop where the merging of the existing trees t_{main} and t_i is done. If the new sample is collision-free and is added to the tree t_i, a check is done whether the new sample is also close to any other existing sampled tree $t_j, i \neq j$. If there is such a tree t_j, both trees are joined (i.e., the currently sampled tree t_i is extended).

We also modified the rejection of samples in collision with environment. An often case is that the sample is rejected, although only its small part is in collision with the surroundings. Therefore, in case of a collision, its 'size' is checked. If there is a 'big' collision (e.g., more than 20% of the agent is in an obstacle),

Algorithm 1: The proposed solution with modified RRT algorithm

Data: The flexible agent A with initial position v_{start}, set of obstacles O
Result: The main tree structure t_{main}

```
Algorithm proposed_solution
    exits ← computeExits (v_start)
    T ← ∅
    foreach e ∈ exits do
        t ← RRT (e, T, n, O)
        Add t into T
    t_main ← RRT (v_start, T, n_max, O)
    return t_main
Procedure RRT(Tree root v_root, set of trees T, number of iterations n, set of
  obstacles O)
    Create the tree t with the root v_root
    Collision-free rotation of agent in the root position v_root
    repeat n times
        Create a new sample s
        Steer s to the tree t
        if s is collision-free then
            Add s to the tree t
            if s is close to some tree t_j ∈ T then
                Merge the tree t_j into the tree t
    return t
```

the sample is rejected. Otherwise, we try to push the agent out of the obstacle to the free space following the shortest trajectory (Fig. 2a). There are multiple directions where to push the agent but we are using the direction with the shortest shift of the agent to the free space. Subsequently, it is necessary to check whether this shift was accessible or not. If the agent is collision-free (Fig. 2b), the sample is accepted and added into the tree structure. However, it may happen that the push-out of one obstacle results in a collision of the agent with another obstacle (Fig. 2c). In this case, the sample is rejected.

(a) (b) (c)

Fig. 2. (a) Small collision with obstacles, (b) Sample pushed correctly to the free space, (c) Sample pushed to another collision

4 Experiments and Results

All experiments of the proposed solution, which was implemented in C#, were performed on a computer with the CPU Intel® Core™ i7-7700K (4.2 GHz) and 64 GB 2400 MHz RAM. Three types of environments were used for the testing of the proposed solution – two artificial environment data sets and real biomolecule *dcp* (proteins are freely available from data bank). Moreover, each of the environments was tested with a different flexible agent.

In the first column of figures there are the first artificial data (Fig. 3a) that are compounded from two hollow cubes connected by tunnels. There is also a cross-section (Fig. 3d) of these data, the location of the exit point (green circle with red cross) and the starting position (green rhombus with red cross) are shown, too. Figure 3g contains an example of one state of a flexible agent. In total, we have 15 different states of this flexible agent with different positions and rotations of individual spheres towards each other. Similarly, the second column contains data, also artificially created, that resemble a shell (Fig. 3b). To be more specific, it represents a hollow sphere with a crack. The position of the start (green rhombus with red cross), which is exactly in the middle, and the exit point (green circle with red cross) is shown in Fig. 3e. Using this data, we navigated the agent shown in Fig. 3h, which also has 15 different states. The last column contains real biomolecular data (Fig. 3c). The cross-section of these data with location of some exit points (green circles with red cross) and starting position (green rhombus with red cross) are shown in Fig. 3f. Figure 3i contains an example of one state of a flexible agent (ligand). In total, we have 100 different states of this flexible agent with different positions and rotations of individual spheres towards each other.

Now let us look at the difference in the tree structure, which is an algorithm output, using the RRT algorithm with and without exit points. The biggest visual difference between the results of RRT and its modification is at the beginning of the computation. Differences remain also in the further course of the algorithms, but it is difficult to distinguish them visually, because the environment is filled with a large number of samples. Note that the starting position v_{start} is colored green and each exit point v^i_{exit} has red color (Fig. 4). Due to the good visibility of individual trees t_i, the rendering of obstacles is disabled. In the case of a modified RRT algorithm, we may notice that we have more than one tree (Fig. 4a). A small tree t_i is created around each exit point, and then the main sampling starts from the start point v_{start}, which creates the main tree t_{main}. The isolated vertices in Fig. 4a are those exit points which did not lead to a collision-free configuration, so were not subsequently used in the calculation. Figure 4b then shows the behaviour of the standard RRT algorithm, which will subsequently have a problem with narrow passages, as there is very little probability that the algorithm will hit the right place with the correct configuration.

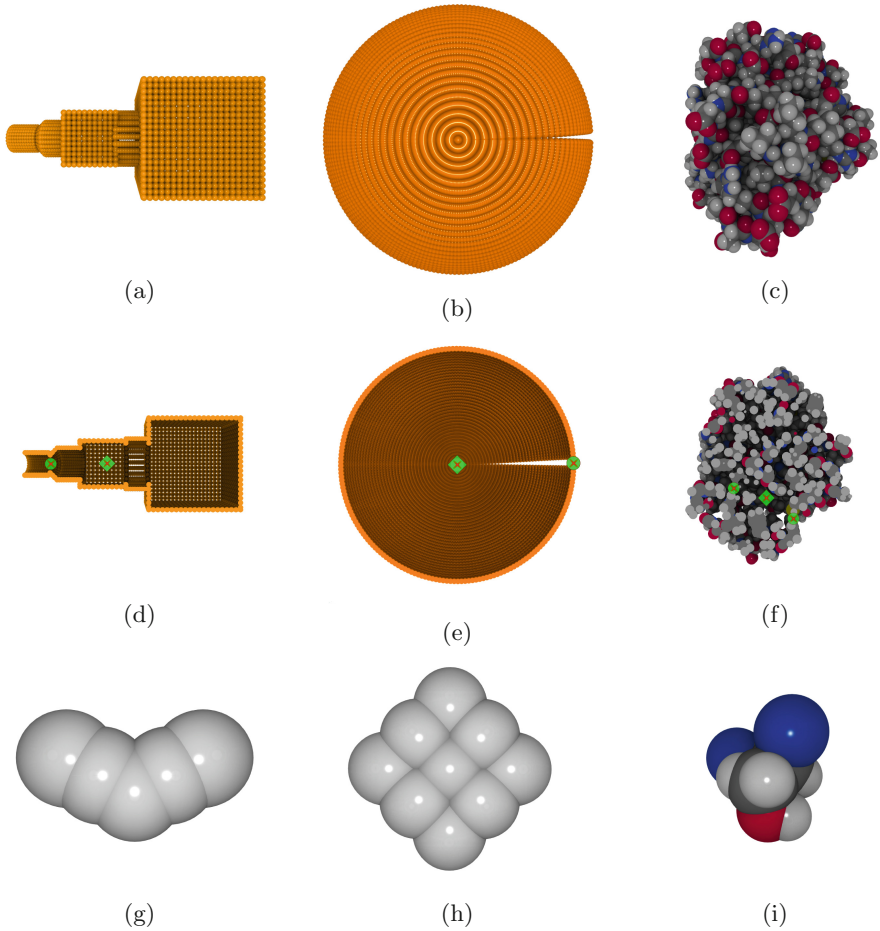
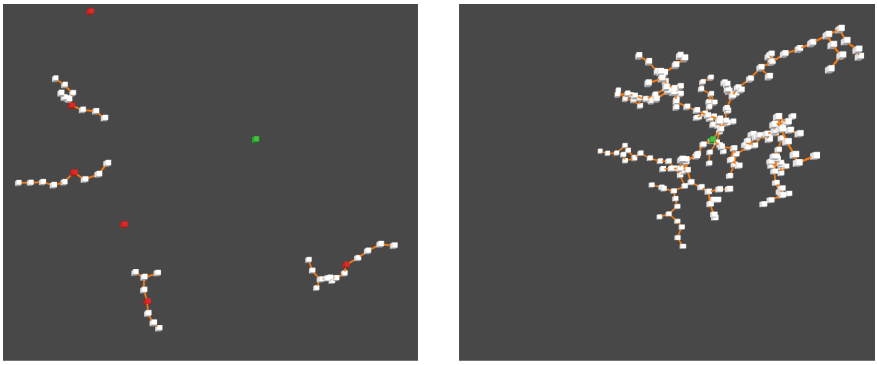


Fig. 3. (a–c) Tested data sets, (d–f) Cross-section through tested data sets, (g–h) Examples of navigated flexible agents (Color figure online)

Results of the standard RRT method and of our proposed solution (let us call it mRRT for short), applied on the first artificial data (Fig. 3a) are shown in Table 1. There is only one possible way how to get out through the obstacles and it was found in all cases by both tested algorithms with 100% success. However, mRRT has a clear superiority over the RRT algorithm as to the time of computation. As mentioned there is only one path through data but Table 1 contains 1000 found paths. This column means that we have ran the computation 1000 times with different seed of random generator.

This time difference was bigger on the other artificial data (Fig. 3b), where there is also one possible pass but the passage through the data now contains more possible space around the narrow passage, but at the same time a large amount of free space where the sampling algorithm may become congested.



(a) (b)

Fig. 4. Start of (a) The modified RRT algorithm with exit points, (b) Standard RRT algorithm

Table 1. Comparison of the RRT algorithm and our modified version (mRRT) - first artificial data

Algorithm	Number of found paths	Time [s]			
		Lowest	Average	Median	Biggest
RRT	1000	28.31509	281.595	259.9069	861.9697
mRRT	1000	0.145702	14.79853	0.23456	59.82809

In this case, the RRT algorithm did not find its passage through data at all (Table 2). The calculation even ended up with a low memory error, after having checked one million samples in approximately two hours of each run. Similarly to the previous table, the column with the number of paths means how many times was the path found from 5000 runs with different seeds. The mRRT found its way in every run in a very good time.

Table 3 compares the results of the real biomolecular data. The tested algorithms were run 500 times for the time of two minutes, and each run had a different seed of random sampling generator (but the same seed was used for both algorithms) Table 3 contains information on how many times the RRT algorithm was better in the evaluated property than mRRT and vice versa where the

Table 2. Comparison of the RRT algorithm and our modified version (mRRT) - second artificial data

Algorithm	Number of found paths	Time [s]			
		Lowest	Average	Median	Biggest
RRT	0	Out of memory	Out of memory	Out of memory	Out of memory
mRRT	5000	0.235554	11.2778	9.891314	53.59965

Table 3. Comparison of the RRT algorithm and our modified version (mRRT) - tested on the real biomolecular data

Algorithm	Number of paths	Time of the found path					Count of samples		
		First	Second	Third	Fourth	Fifth	Accepted	Rejected	Total
RRT	0	37	206	315	289	177	476	61	425
mRRT	500	463	294	185	211	323	24	439	75

evaluated properties were higher number of found paths, shorter path finding times, more samples accepted, fewer rejected samples, and more total samples, respectively. In all cases mRRT algorithm found a higher number of paths than the RRT algorithm. The result of the time comparison of these methods is interesting. The first two passes through the data were found faster by our method, the third and fourth by standard RRT algorithm, and since the fifth pass, mRRT began to lead again.

The last mentioned modification in the proposed solution - pushing the agent out of the obstacle if there is only a small collision with the environment - is in Table 4. There is a comparison of both RRT and mRRT algorithms with and without this modification on the real biomolecular data. Table 4 provides information on how many times the RRT algorithm was better without sample correction than with sample correction and vice versa (same for mRRT). Focusing on the RRT algorithm, we can notice that the first two passes through the data are found faster without correcting the samples. However, the other three passes are found faster when using this approach. At the same time, it is essential that using this approach, the number of samples received is higher and the number of rejected samples is lesser than the standard RRT algorithm has. The total number of samples is lower, but this is because the RRT algorithm without this approach has a much larger number of rejected samples. In the case of our proposed solution (mRRT), it is better to use the sample correction, because in almost all cases we get better results. Only in the total number of samples is the result worse, but the reason is the same as in the case of the standard RRT algorithm.

We should point out that results are highly dependent on the data type: if the data contain only narrow passages, mRRT is better. If the data contains large passes, mRRT is slower due to the calculation of exit points (useless for

Table 4. Comparison of the RRT algorithm and our modified version (mRRT) - push with and without

Algorithm	Time [s]					Samples		
	First	Second	Third	Fourth	Fifth	Accepted	Rejected	All
RRT without push	1931	1484	960	704	609	0	35	2187
RRT with push	333	780	1304	1560	1655	2264	2229	76
mRRT without push	813	727	710	696	732	0	28	1774
mRRT with push	1023	1109	1126	1140	1104	1836	1808	62

large passes). In the current data there are both types of passes (large passes and narrow passages) where our modified solution finds the narrow passages first and than the large passes. On the other hand, the standard RRT algorithm first finds the large passes and then the narrow passages (very low probability).

5 Conclusion

In this paper we introduced a modification of the RRT algorithm with improved ability to find a collision-free path for a flexible agent in an environment represented by configuration space. The proposed modification has been described for RRT algorithm but can be used in any sampling-based algorithm. Besides finding the path, the method provides knowledge where exactly the narrow passages are. This is extremely useful information as with proper tools it is possible to decide whether the narrow passage is passable or not. The RRT algorithm gives us two possible answers - there is a path through data or the algorithm cannot find any (the path may exist or not). The modified algorithm is able to give us also two answers - there is a path through the data or there is none. The proposed solution is most suitable for the data with narrow passages, where it is multiple times better than the original algorithm. For data without narrow passages the proposed method is slower than the standard sampling-based algorithm due to the extra computation of exit points and sampling their surrounding. However, a correct path will be found even for such unfavourable data.

Acknowledgement. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic, the project SGS-2019-016 Synthesis and Analysis of Geometric and Computing Models, and funded by Czech Science Foundation, the project No. 17-07690S Methods of Identification and Visualization of Tunnels for Flexible Ligands in Dynamic Proteins.

References

1. Amato, N.M., Dill, K.A., Song, G.: Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. J. Comput. Biol. **10**(3–4), 239–255 (2003)
2. Cortés, J., Barbe, S., Erard, M., Siméon, T.: Encoding molecular motions in voxel maps. IEEE/ACM Trans. Comput. Biol. Bioinform. (TCBB) **8**(2), 557–563 (2011)
3. Cortés, J., Le, D.T., Iehl, R., Siméon, T.: Simulating ligand-induced conformational changes in proteins using a mechanical disassembly method. Phys. Chem. Chem. Phys. **12**(29), 8268–8276 (2010)
4. Ferré, E., Laumond, J.-P.: An iterative diffusion algorithm for part disassembly. In: Proceedings of the 2004 IEEE International Conference on Robotics and Automation, vol. 3, pp. 3149–3154. IEEE (2004)
5. Geraerts, R., Overmars, M.H.: A comparative study of probabilistic roadmap planners. In: Boissonnat, J.-D., Burdick, J., Goldberg, K., Hutchinson, S. (eds.) Algorithmic Foundations of Robotics V. STAR, vol. 7, pp. 43–57. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-45058-0_4

6. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
7. Kavraki, L.E., Kolountzakis, M.N., Latombe, J.-C.: Analysis of probabilistic roadmaps for path planning. IEEE Trans. Robot. Autom. **14**(1), 166–171 (1998)
8. Kavraki, L.E., Svestka, P., Latombe, J.-C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Autom. **12**(4), 566–580 (1996)
9. Kurniawati, H., Hsu, D.: Workspace-based connectivity oracle: an adaptive sampling strategy for PRM planning. In: Akella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) Algorithmic Foundation of Robotics VII. Springer Tracts in Advanced Robotics, vol. 47, pp. 35–51. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68405-3_3
10. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
11. Manak, M.: Voronoi-based detection of pockets in proteins defined by large and small probes. J. Comput. Chem. **40**(19), 1758–1771 (2019)
12. Manak, M., Anikeenko, A., Kolingerova, I.: Exit regions of cavities in proteins. In: 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering, BIBE, pp. 1–6. IEEE Computer Society (2019)
13. Otte, M., Frazzoli, E.: RRTX: asymptotically optimal single-query sampling-based motion planning with quick replanning. Int. J. Robot. Res. **35**(7), 797–822 (2016)
14. Vonásek, V., Faigl, J., Krajník, T., Přeučil, L.: A sampling schema for rapidly exploring random trees using a guiding path. In: Proceedings of the 5th European Conference on Mobile Robots, vol. 1, pp. 201–206 (2011)