Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 12(2): 116-121, Jun. 2014 Regular paper

Heuristics for Motion Planning Based on Learning in Similar Environments

Dmitriy Ogay[1] and Eun-Gyung Kim[2*], ***Members, KIICE***

[1]Department of Computer Science & Engineering, Graduate School, Korea University of Technology and Education, Cheonan 330-708, Korea
[2]School of Computer Science & Engineering, Korea University of Technology and Education, Cheonan 330-708, Korea

Abstract

This paper discusses computer-generated heuristics for motion planning. Planning with many degrees of freedom is a challenging task, because the complexity of most planning algorithms grows exponentially with the number of dimensions of the problem. A well-designed heuristic may greatly improve the performance of a planning algorithm in terms of the computation time. However, in recent years, with increasingly challenging high-dimensional planning problems, the design of good heuristics has itself become a complicated task. In this paper, we present an approach to algorithmically develop a heuristic for motion planning, which increases the efficiency of a planner in similar environments. To implement the idea, we generalize modern motion planning algorithms to an extent, where a heuristic is represented as a set of random variables. Distributions of the variables are then analyzed with computer learning methods. The analysis results are then utilized to generate a heuristic. During the experiments, the proposed approach is applied to several planning tasks with different algorithms and is shown to improve performance.

Index Terms: Heuristics, Learning from experience, Machine learning, Motion planning, Path finding

I. INTRODUCTION

Motion planning has been an essential part of robotics for a long time. Many algorithms have been developed, which develop a plan for a robot to move from the initial configuration to the target configuration. Nevertheless, algorithm complexity has been a challenge for real-life applications.

The development of motion planning from graph search algorithms [1] and has led to the modern complicated sampling-based algorithms [2]. Dijkstra's algorithm is one of the simplest and basic algorithms for finding paths on a graph. One of the approaches for the construction of a motion plan in a configuration space is to discretize the space with hyper-cubes and then, execute a graph search algorithm on a discrete structure. However, in practical applications, this approach can rarely be used for tasks with high dimensionality, because the algorithm complexity increases exponentially.

An extension of Dijkstra's algorithm, which is called A* [1], introduced heuristics for graph-search algorithms to significantly improve the algorithm's performance in terms of the computation time, depending on the task.

The other successful approach for motion planning in a high-dimensional space is the Rapidly Exploring Random Tree algorithm [3]. This algorithm rapidly extends a tree

Received 03 December 2013, **Revised** 01 February 2014, **Accepted** 17 February 2014
***Corresponding Author** Eun-Gyung Kim (E-mail: egkim@koreatech.ac.kr, Tel: +82-41-560-1350)
School of Computer Science & Engineering, Korea University of Technology and Education, 1600, Chungjeol-ro, Byeongcheon-myeon, Dongnam-gu, Cheonan 330-708, Korea.

Open Access **http://dx.doi.org/10.6109/jicce.2014.12.2.116** print ISSN: 2234-8255 online ISSN: 2234-8883

structure in a high-dimensional space and has proved to be efficient in many practical applications. However, it suffers in the presence of narrow paths. Various methods have been proposed to overcome this problem [4].

However, in recent years, more attention has been drawn to the area of experience-based planning, because tasks have become increasingly complicated. Re-planning strategies in similar environments are covered in [5-7]. Trajectory libraries approaches are covered in [8-10].

Heuristics are applied in many practical motion planning tasks. However, as planning problems become increasingly difficult, the complexity of heuristics increases, and designing a heuristic itself becomes a challenging task.

The main contribution of this work is the development of an approach for the algorithmic generation of heuristics. The work is done in the following steps:

- Represent a motion planning algorithm's interaction with the environment as a random process.
- Represent a heuristic as a distribution of random variables that control the random process of planning.
- Since a heuristic is usually developed by humans on the basis of their knowledge about the type of environment or the type of task. Assume that knowledge about environment is the knowledge of the distribution of obstacles in the environment.
- Apply machine learning to learn the distribution of obstacles in typical environments. Utilize the learnt distributions as a heuristic to control the extension of a motion planning algorithm.

In the experimental section, the designed heuristic is applied to different types of motion planning algorithms in order to show the performance improvement.

II. PROPOSED METHOD DESCRIPTION

A. Heuristic and Distribution of Obstacles

Let us consider an example of motion planning in a two-dimensional space for a point mass in a free space. The shortest path would be a straight line, connecting the initial and the target configurations. The shortest path is different from a straight line, if there are obstacles between the initial and the target points. Therefore, obstacles do affect the final plan.

Now, let us analyze an example of planning in a real environment with obstacles. Let us have an intelligent agent A, and an observable environment I around it. We can model this environment I in a natural world with a vector of random variables $X = \{X_1, X_2, ..., X_n\}$. If we take another vector of uniformly distributed random variables, then the information content of X would be lower than a uniform random and thus, would have a certain pattern. We use the term information from the information theory developed by Shannon [11]. It means that the average length of the sequence that encodes the environment I would be shorter than the sequence to encode the vector of uniformly distributed independent random variables.

To illustrate the above idea, let us consider an example of a robot, which navigates on the ground. Let us model the surrounding physical world as a set of tuples $\{\boldsymbol{p}, \boldsymbol{o}\}$, where $\boldsymbol{p}$ denotes the position of the object's center mass and $\boldsymbol{o}$ represents the orientation of an object along the longest dimension. This is one example of how the problem may be modeled for the planning algorithm.

We can also model the above-mentioned world with random variables X and Y, the first corresponding to the position and the second to the orientation. What would the distribution of these variables be? Here, the laws of nature and the corresponding constraints start to affect the distribution. For our example, let us take gravitation. Without gravitation, the distribution of Y would be closer to uniform. However, in the real world, most of the objects are either vertical or horizontal. This implies that the entropy $\mathrm{H}(Y)$ is less than the maximum value, which is the entropy of a uniform distribution.

Basic planning algorithms without heuristics are designed to be systematic and sufficiently universal to deal with any kind of input problems. Thus, they consider that obstacles are distributed uniformly. However, in fact, if obstacles have some pattern and are not distributed uniformly, then the planner may work inefficiently. This has led to the introduction of various heuristics to make planning algorithms solve particular problems more efficiently.

Let us represent the interaction between a planner and an environment as a random process. As a heuristic controls the behavior of this interaction at every cycle of the algorithm, the heuristic is considered to be a distribution of the possible actions that the planner could perform.

It should be noted that a heuristic may be deterministic with respect to some calculated values, and have conditional entropy equal to zero (e.g., the A* algorithm), but due to the randomness of obstacles, the full entropy of the heuristic may be more than zero.

The main idea behind this work is as follows: If a human designs a heuristic for a particular task, then it means that he/she tries to model a probability distribution function of the actions that the planner would take. In the proposed approach, we try to model this probability distribution function algorithmically, with the help of machine learning.

B. Generalized Planning Framework

To further analyze motion planning algorithms and heuristics, let us describe a generalized planning framework.

Algorithm 1 Basic Planning Framework

```
1: $G(V,E) \leftarrow (q_{init}, \emptyset)$;
2: repeat
3:     $\{q_{ext}, u_{new}\} \leftarrow selectNodeAndAction(G)$
4:     $q_{new} \leftarrow propagate(q_{ext}, u_{new})$
5:     if $q_{new} \neq \emptyset$ then
6:         $G.addVertex(q_{new})$
7:         $G.addEdge(q_{ext}, q_{new})$
8:     end if
9: until $goalReached()$
```

Algorithm 2 Template Extension for Random Tree

```
1: function $selectNodeAndAction(G)$
2:     $q_{cand} \leftarrow X$
3:     $u_{new} \leftarrow Y$
4: end function
```

Algorithm 3 Template Extention for RRT

```
1: function $selectNodeAndAction(G)$
2:     $q_{cand} \leftarrow rand()$
3:     $q_{ext} \leftarrow nearest(q_{cand}, G)$
4:     $u_{new} \leftarrow propagate^{-1}(q_{ext}, q_{cand}, \Delta t)$
5: end function
```

It is an algorithm template that may be instantiated to describe most of the modern sampling-based planning algorithms. See Algorithm 1.

G(*V,E*) is a graph, which is extended to cover the configuration space. The main extension point of the algorithm is the function *selectNodeAndAction*(*G*).

This function generates a pair {q_{ext}, u_{new}}, where q_{ext} denotes a node of graph *G*, which is to be extended, and u_{new} represents an action that will be applied to the system from the state q_{ext}. The function *propagate*(q_{ext}, u_{new}) serves as a collision checker, which integrates the control input and validates whether the generated path or trajectory is collision free.

The generalized framework may be extended to be a random tree, with configurations sampled according to some random distributions. Algorithm 2 is an extension of a template for a randomly built tree. Nodes q_{cand} to be extended are selected with a random variable *X*, and the control input is selected with a random variable *Y*.

This graph extension strategy is itself sufficiently general, because it can model other sampling-based planning algorithms in terms of the distributions of random variables *X* and *Y*.

EXAMPLE 1: *X* and *Y* have a uniform distribution. Any node in the graph is selected randomly with equal probability for all nodes, and the control input is also selected to be a uniformly random direction. This example is important because we may use it as an unbiased random planner without heuristics. Therefore, we can consider a bias in the distribution of random variables *X* and *Y* as a heuristic.

The generalized algorithm may also be utilized to implement the well-known Rapidly Exploring Random Trees (RRT) [4]. The extension of the function *selectNodeAndAction*(*G*) for the RRT implementation is shown in Algorithm 3. In line 2, a random configuration is selected. The node to be extended is the node in the tree, which is the nearest node to the selected candidate configuration. The control input is then determined as a control input, required to bring a robot from configuration q_{ext} to the configuration q_{cand}, limited by a maximum time step of Δt.

EXAMPLE 2: RRT may be modeled by biasing distributions of *X* and *Y*. For *X*, the distribution will be determined by the current graph *G*, which is being extended. The existing nodes divide the configuration space with Voronoi cells. These cells have different areas, and nodes that are closer to the unexplored areas have a larger area of Voronoi cells around it. In the implementation of the RRT algorithm, the probability of a node *x* to be selected is proportional to the volume of the Voronoi cell around this node. The distribution for *Y* also depends on the size and the shape of the Voronoi cell around the node.

We can conclude from these two examples that biasing of the distribution of *X* and *Y* creates a heuristic, which controls the extension of a search graph. The RRT nodes with large Voronoi regions, which are the nodes closer to the unexplored area, have larger chances to be extended. In other words, RRT has the heuristic of exploring the unknown regions.

For the study presented in the given work, we apply machine learning algorithms to find the distribution of the variable *Y*, which would make our motion planner work more efficiently. During the experiments, we compare the performance of planning with the learnt distribution of *Y* in two cases: {uniform *X* and uniform *Y*} vs. {uniform *X* and learnt *Y*}; and {RRT-based *X* and RRT-based *Y*} vs. {RRT-based *X* and learnt *Y*}.

III. EXPERIMENTS AND DISCUSSION

This section describes how heuristics may adapt to the given typical environment by learning. Learning is implemented as finding the probability distribution function of a planner's parameters from the observations of a typical environment.

In previous sections, sampling-based planners were generalized to a tree structure, which expands in a search space. The expansion is parameterized in two ways. The first is which a node is selected to be extended, and the

Table 1. Performance of self-learning heuristics applied to random tree search algorithm

Metric	Unbiased	Learning	Improvement
Iteration	500,000 (0)	84,000 (42,473)	5.95
Collision	136,510 (848)	28,598 (16,909)	4.77
Path length	fail	1,665 (65)	-

Values are presented as mean (standard deviation) or number of times.

Table 2. Performance of random tree search algorithm with self-learning heuristics when the number of iteration is fixed

Metric	Unbiased	Learning	Improvement
Iteration	1,000 (0)	1,000 (0)	1
Collision	259 (18)	412 (37)	0.62
Path length	fail	fail	-

Values are presented as mean (standard deviation) or number of times.

second is the selection of the control input to be applied to the extending node.

In the presented experiment, the second parameter is utilized to improve the performance of a planning algorithm. The control input is learned from the observations of successfully generated paths. RRT was used to generate the paths for the observations.

After the probability distribution function was approximated with the sum of the Gauss distributions with learned means, it was applied to the sampling of the control input.

In order to show that *z* planner with applied self-learning heuristics performs better, we applied the proposed strategy to different sampling-based planners. Planners were selected to differ in the way nodes for extensions are selected. In the first case, we utilized the uniform random selection of nodes from the whole tree. In the second case, we utilized the selection of nodes for rapid expansion in space. The probability that a node is selected is proportional to the volume of the Voronoi cell that corresponds to the node.

All the algorithms were run 10 times. The number of iterations, number of failed collision checks, and path lengths were recorded. In the first run, the algorithms were run until the path was found. In the second run, planning was run for a fixed number of iterations.

The results of the experiments with random node selections are shown in Tables 1 and 2. As is apparent, random node selection is inefficient, and an unbiased algorithm could not find the solution even after 500,000 iterations. Self-learning heuristics improved the performance of the planner, and thus, it could find a solution. The boundaries of exploration can be seen in Fig. 1.

The results of the experiments with rapidly expanding node selection are shown in Tables 3 and 4. This strategy extends the algorithm faster. An unbiased version of the algorithm is in fact an implementation of an RRT algorithm. Self-learning heuristics improved the performance of the planner, and thus, it could find the solution faster as shown in Fig. 2. Planning is performed from the initial point in the left top corner to the goal point in the right bottom corner.

Unlike expected, the self-learning heuristics caused more collision check fails during the test with a fixed number of iterations (See Table 2). In both cases, motion planners failed to reach the goal; therefore, the number of iterations was considerably small. If compared to the results from Table 1, it may be seen that the self-learning heuristic experiences relatively few collision failures before it reaches the target.

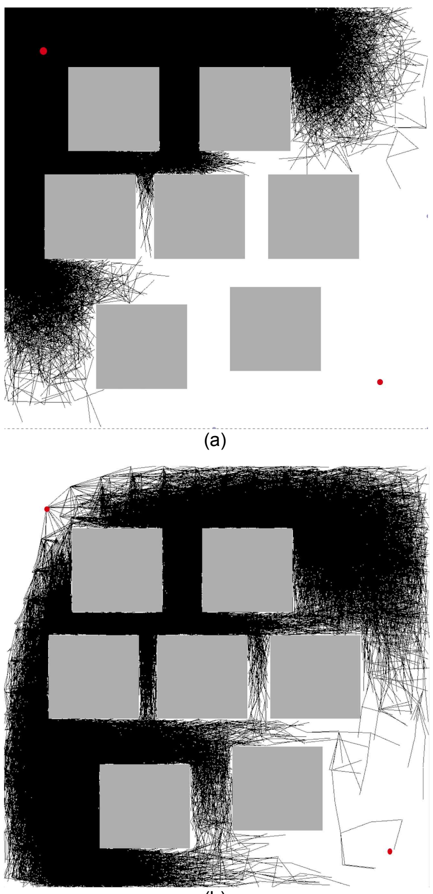
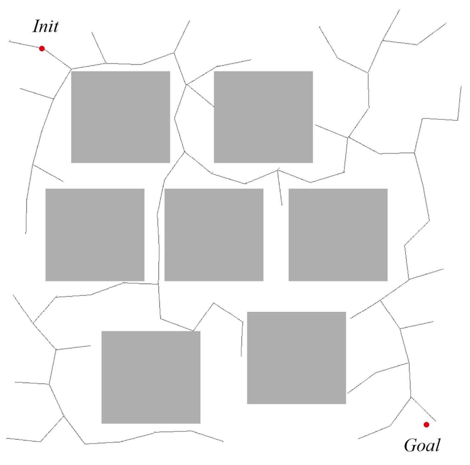


Fig. 1. Comparison of the performance of a planner with random selection of nodes for expansion. The initial point is in the top left corner, and the final point is at the bottom right. (a) Unbiased control input. After 500,000 iterations, a path was still not found. (b) Control input biased by self-learning heuristics.

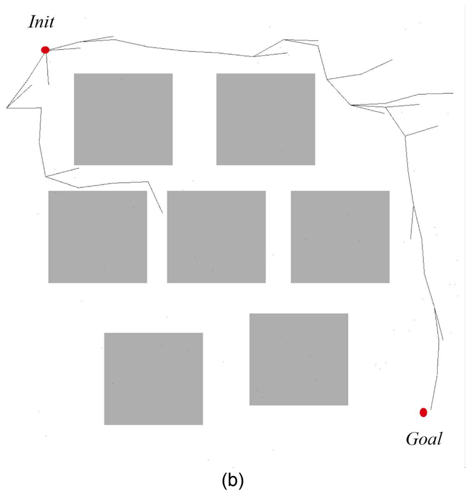
Table 3. Performance of self-learning heuristics applied to the search algorithm with Voronoi cell volume-based node selection

Metric	Unbiased	Learning	Improvement
Iteration	1042 (1521)	203 (94)	5.13
Collision	243 (253)	92 (58)	2.64
Path length	1711 (166)	1620 (107)	1.056

Values are presented as mean (standard deviation) or number of times.



(a)



(b)

Fig. 2. Performance of planner with selection of nodes for expansion with rapid expansion. Extension is performed from the Init configuration to the Goal configuration. (a) Unbiased control input and (b) control input biased by self-learning heuristics.

Table 4. Performance of the search algorithm with Voronoi cell volume-based node selection and self-learning heuristics when the number of iteration is fixed

Metric	Unbiased	Learning	Improvement
Iteration	1,000 (0)	1,000 (0)	1
Collision	206 (42)	392 (27)	0.52
Path length	1,652 (167)	1,642 (52)	1.006

Values are presented as mean (standard deviation) or number of times.

Table 5. Results of running algorithm, until a path is found, on a map with narrow paths: random tree with Voronoi cell volume-based node selection: comparison with and without self-learning heuristics

Metric	Unbiased	Learning	Improvement
Iteration	2,566 (3720)	544 (299)	4.7
Collision	1,831 (2725)	405 (231)	4.5
Path length	1,704 (113)	1,642 (85)	1.03

Values are presented as mean (standard deviation) or number of times.

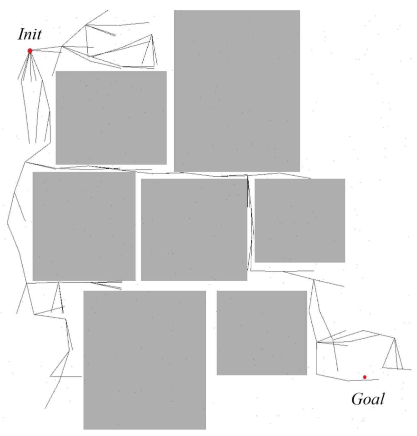


Fig. 3. Performance of the planning algorithm on a map with narrower paths.

The same experiments were also performed on a similar map, with a considerably narrower distance between squares; see Fig. 3. The result is presented in Table 5. As expected, the original RRT's performance started to drop significantly. In our experiments it sometimes took more than 10,000 iterations and the algorithm still could not find the solution. Because of these failures, the average number of iterations is high. In all three experiments, the algorithms enhanced with self-learning heuristics improved the performance of the original algorithm.

IV. CONCLUSIONS

In this work, we have proposed a method, which allows heuristics for a motion planner to be generated by a computer by observing the results of prior successful planning tasks in the environment. We have experimentally shown that if planning algorithms are generalized and parameterized with two random variables, then the observation of the distribution of these parameters on successful paths may be utilized to create a heuristic. The proposed method was applied to different sampling-based planning algorithms in different environments, and in all cases, it showed performance improvements.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge: Cambridge University Press, 2006.

[2] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, pp. 3671-3678, 2012.

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic

planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378-400, 2001.

[4] S. Dalibard and J. P. Laumond, "Control of probabilistic diffusion in motion planning," in *Algorithmic Foundation of Robotics VIII*. Heidelberg: Springer, pp. 467-481, 2009.

[5] J. M. Lien and Y. Lu, "Planning motion in environments with similar obstacles," in *Proceedings of Robotics: Science and Systems*, Seattle, WA, pp. 1-7, 2009.

[6] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation,* Rome, Italy, pp. 1603-1609, 2007.

[7] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, pp. 3757-3762, 2008.

[8] S. Martin, S. Wright, and J. Sheppard, "Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in environments with moving obstacles," in *Proceedings of the IEEE International Conference on Automation Science and Engineering*, Scottsdale, AZ, pp. 1131-1136, 2007.

[9] C. G. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: a trajectory-based approach," in *Proceedings of the Neural Information Processing Systems Conference (Advances in Neural Information Processing Systems)*, Vancouver, Canada, pp. 1611-1618, 2003.

[10] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, FL, pp. 3344-3349, 2006.

[11] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379-423, 623-656, 1948.

Dmitriy Ogay
received his bachelor's degree in July 2004 and his master's degree in February 2008 from the Computer Science Department of National Aviation University, Kiev, Ukraine. He received his Ph.D. degree in February 2014 from the Department of Computer Science & Engineering, Graduate School of KOREATECH. His research interests include motion planning, machine learning, and parallel and distributed programming.

Eun-Gyung Kim
received her bachelor's degree in February 1983 from the Physics Department of Sookmyung Women's University, her master's degree in February 1987 from the Computer Science Department of the Graduate School of Chung-Ang University, and her Ph.D. in February 1991 from the Computer Engineering Department of the Graduate School of Chung-Ang University. Since March 1992, she has been working with the School of Computer Science & Engineering at KOREATECH and is at present, a professor. Her research interests include intelligent agents, smart learning, and TRIZ.