The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Coarse-to-Fine Approach for Fast Path Finding for Mobile Robots

Jae-Yeong Lee and Wonpil Yu

Abstract—**A simple but effective method for fast path finding for mobile robots in grid-based search space is presented. The paper presents a systematic way to reduce the resolution of a grid map preserving the occupancy structure of the original map. Paths are found in two steps with varying quantization levels of the space in coarse-to-fine manner. The resulting paths are practically optimal and experimental results show a great reduction of the searching time, making it possible to implement a real-time global path planner in a very large environment.**

I. Introduction

FAST path finding is one of the most critical issues in the domains of real-time strategy (RTS) games and robotics applications. Conventional approach for the path planning is to use A* search [1] on a uniform grid representation of the world. A* search has been widely used because it ensures to find an optimal or shortest path. However, it is a common observation that A* can be extremely slow for a large search space; it is known that time and space complexity of A* is $O(b^d)$, where b is the branching factor and d the solution depth. This shortcoming of A* has expedited the development of numerous fast search algorithms.

Warren [2] developed a modified A* algorithm that expands a search tree only on a subset of grid cells uniformly sampled from the start cell with a pre-defined step size; the step size is gradually reduced until a path is found. In our implementation of the method we found that it is very fast and effective for most cases but sometimes produces an abnormal path when the optimal path is excluded by the uniform sampling. Anytime algorithms [3-6] are another fast variants of A* search, which use a strategy of finding an initial suboptimal path quickly, then improving the path quality progressively as time allows. ARA* (Anytime Repairing A*) [7] developed by Likhachev *et al*. further increased the efficiency of the anytime heuristic search by reusing previous searching efforts during iterative refinement of path quality. Anytime algorithms generally speed up the search by sacrificing the optimality condition of A* search and are effective mainly when there exists a nearly straight path to a goal position. For very high-dimensional problems, LaValle and Kuffner [8] developed RRT (Rapidly-exploring Random Tree), which is grows a search tree based on random sampling of the search space biasing the growth of this tree towards a particular goal configuration or unexplored regions. RRT generally does not secure the path quality, although it is very fast. Recently RRT method was extended to provide anytime capability [9], replanning capability [10], or both [11].

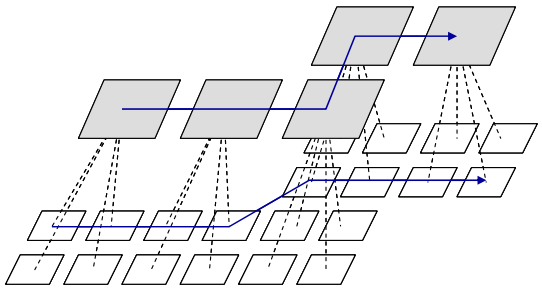

Fig. 1. Hierarchical map abstraction and coarse-to-fine path finding.

As another branch of fast path finding approaches, some researchers have focused on the development of hierarchical abstraction of the environment that can reduce the search space. Kambhampati and Davis [12] reduced the search space by using a quadtree decomposition of the world at the cost of low path quality. Chen *et al*. [13] extended quadtree to framed quadtree by further decomposing the border of each block of quadtree with cells at the highest resolution, which improves the solution quality significantly as it allows crossing a block between any two border cells. Hierarchical approaches [14-17] build special abstraction graphs from original grid maps. Thereafter, initial abstract paths are searched quickly in the abstraction graphs and then refined in the original search space. Rabin [15] proposed a two-level hierarchy that abstracts a map into clusters of rooms in a building or square blocks on a field, which forms a graph with entrance of rooms as vertexes. One drawback of this approach is that the abstraction can't be done in a domain independent way. HPA* [16] builds abstraction graphs automatically by partitioning a map into rectangular blocks of large sectors and calculating optimal local paths between limited sets of entrances and exits to the sectors. The authors reported that HPA* was up to a 10 times faster than A* search with 1 % error from optimal on average.

So far we have briefly reviewed two branches of fast path finding approaches. Fast variants of A* usually speed up the search at the cost of low path quality and have a limited

Manuscript received February 25, 2009. This work was supported in part by the R&D program of the Korea Ministry of Knowledge and Economy (MKE) and the Korea Evaluation Institute of Industrial Technology (KEIT) [2008-S-031-01, Hybrid u-Robot Service System Technology Development for Ubiquitous City].

Jae-Yeong Lee and Wonpil Yu are with the Robot Research Department, Electronics and Telecommunications Research Institute (ETRI), Daejeon, KOREA (e-mails: jylee, ywp@etri.re.kr).

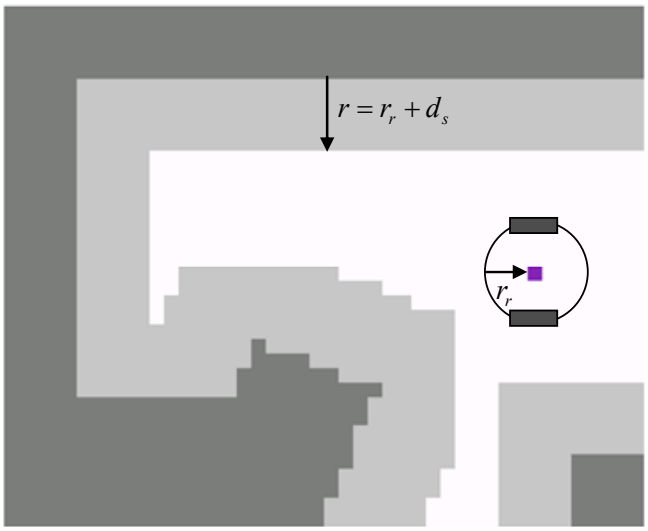

Fig. 2. Compensation of the robot size.

performance for very large maps. On the other hand hierarchical approaches are well-suited for very large maps as they reduce the search space into tractable one. However, previous hierarchical approaches also have a problem of suboptimal solutions. It is mainly because the paths found at abstraction level are used as fixed waypoints for the stage of path refinement. In addition, they need to build and handle special abstraction graphs, which introduce complex implementation issues.

In this paper we present a new hierarchical approach for fast path finding. Our algorithm, *Coarse-to-Fine A** (CFA*), finds paths in two steps with varying quantization levels of the space in coarse-to-fine manner (Fig. 1). Different from previous hierarchical approaches we do not introduce a new graph structure, which needs complex processing, to abstract the search space. Instead, we use simple grid representation to abstract the structure of the world, which forms another grid map with coarse resolution. The key idea or main contribution of the paper is to provide a systematic way to reduce the resolution of a grid map while preserving the occupancy structure of the original grid map. Our approach very simplifies the implementation and enables obtaining practically optimal paths. Experimental results show that our method is roughly $n/10$ times faster than A* search for the problems of path length n with 0.1 % error from optimal on average; it is 20 times faster for paths of length 200 and 40 times for length 400.

The paper is organized as follows. In Sect. II we describe how to build an abstraction map of coarse resolution, while preserving the structure of the original map. We then describe our coarse-to-fine path finding method in Sect. III. We show experimental results in Sect. IV and conclude the paper in Sect. V.

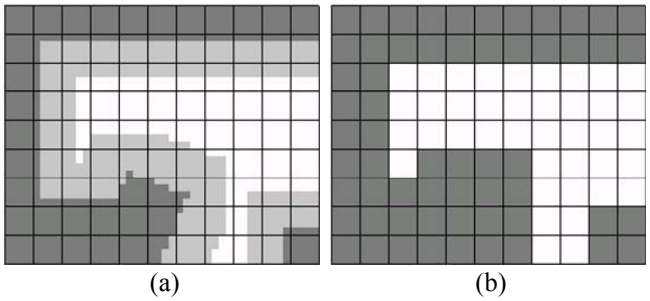

Fig. 3. Building an abstraction: (a) block partitioning of a configuration map of 36 × 44 grid cells (cell boundaries are not drawn), (b) the resulting block map built from (a) with block size $k = 4$.

II. Building Abstraction of the Search Space

In this section we describe how to build an abstraction of the search space. The search space in our problem is represented by a grid map, each cell of which has one of two states: *free* or *occupied*.

The path planning task for a mobile robot is to find an optimal or feasible path from current position to a goal position on the map. For the path planning the robot size should be considered for a robot to safely visit the path cells without causing a physical collision. For a given map of the problem space, therefore, we first compensate the robot size by enlarging the occupied cells in the map by $r = r_r + d_s$ grid cells in all directions, where r_r is the robot radius in the unit of grid cells and d_s is an optional safety term (Fig. 2). This compensation is common in robotics domain [18]. The safety term d_s usually is determined by considering localization error, sensor accuracy, and motion control error. With the compensation the robot can be treated as a point-like vehicle, which very simplifies the path planning task. The resulting grid map with occupied cells being enlarged by r grid cells is then used to build an abstraction map and to find a fine path in online coarse-to-fine search. We will call this map by *configuration map*.

An abstraction of the space, called *block map*, is then built from the configuration map by merging each disjoint neighboring $k \times k$ grid cells into a single block. By this abstraction a $m \times m$ configuration map is reduced into a $\lceil m/k \rceil \times \lceil m/k \rceil$ block map. The state of each block is set to be *occupied* if all of the grid cells within the block are occupied and set to be *free* otherwise. Figure 3 shows an example of building a block map from a sample configuration map.

One of the most important requirements that the resulting block map should satisfy is to preserve the occupancy structure of the original configuration map. This requirement is guaranteed if the block size k is determined to satisfy

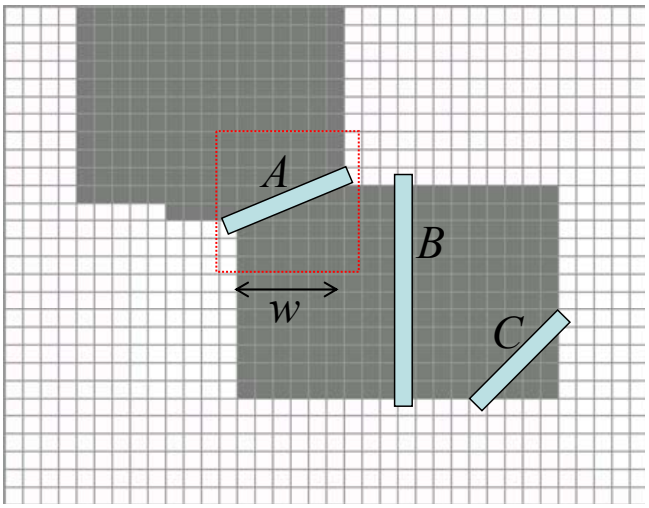

Fig. 4. Determination of the minimal width of the occupied region.

$$k \le \left\lfloor \frac{w+2r+1}{2} \right\rfloor, \tag{1}$$

where w is the minimal width of the occupied region in the original grid map (not in the configuration map). Here we give an intuitive explanation. First, the structure of free region is preserved in the block map as the blocks containing any free cell are set to be free. That means if there is a cell path between two free cells in the configuration map there always is a block path in the block map consisting of blocks that contain the cell path. Second, the structure of occupied region is also preserved since at least one block is completely included in the occupied region and set to be occupied with any displacement of block-partitioning if the block size is equal or less than half of the width of the occupied region. In other words, it prevents a block path from being generated across the occupied region. In Eq. (1), $w + 2r$ represents the minimal width of the occupied region in the configuration map and increased by one for the cases of odd value of w.

The only remaining problem is to determine the value of w. Let us consider a pair of open cells and a line segment that connects the open pair and has the width of one grid cell. One constraint is that the line segment should not intersect with any other open cells except the pair of open cells. Among such line segments, we find one that has the smallest *minimal bounding square*. Let l be the size of the smallest minimal bounding square. Then, the value of w is given by $l - 2$. Figure 4 shows several examples of line segments on a sample grid map. In the figure, line A and line B satisfy the condition because they do not intersect with any other open cell except two ending points. However, line C intersects with two additional open cells and thus doesn't meet the condition. For the case of sample map given in Fig. 4, line A gives the smallest minimal bounding square, and we have $w = 6$.

In a point of implementation view, however, it is not easy to determine w automatically. In addition, there are some exceptional cases that the value of w is not well defined. An empty map with no occupied cell is such an example, where we can use any block size. In order to avoid complex implementation, therefore, we can simply set the value of w as one and determine the block size k by the following formula:

$$k \le r + 1. \tag{2}$$

Eq. (2) gives stricter upper bound on block size k than the real given by Eq. (1) since $w \ge 1$. If w cannot be defined as the case of empty map, we can use any value of k within the map size. Therefore, we always can determine the block size k safely with Eq. (2) not depending on the map structure.

III. Online Coarse-to-Fine Search

The CFA* finds paths in two steps of coarse and fine search using the pre-computed block map and configuration map. We use A* search [1] for both coarse and fine search to ensure optimal solutions.

For a given pair of start position S and goal position G in the configuration map, we first determine the corresponding start block S' and goal block G' in the block map which include S and G respectively. We define a *block path* as the

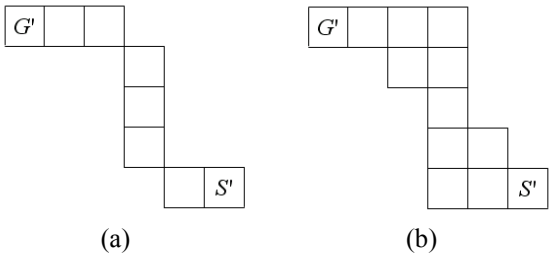

Fig. 5. (a) A block path. (b) Augmented block path.

path found in the block map from a start block S' to a goal block G'. The block path is found by A* search in the block map based on 8-connectivity. The search on a block map can be done very fast since the size of the search space has been greatly reduced. The resulting block path is used to restrict the search space of the configuration map in the next fine search.

Since we use 8-connectivity, some blocks in the block path can be diagonally connected. For each diagonal pair of the block path, we add the other two adjacent cross-diagonal blocks into the block path, which forms an *augmented block path* (Fig. 5). This augmentation is very important in two aspects. Although the block path is optimal in the block map, optimal cell path might not be within the block path. By augmenting the block path, we can minimize the possibility that such a situation occurs. More importantly it guarantees that there always exists a cell path from S to G within the augmented block path. Without augmentation a cell path

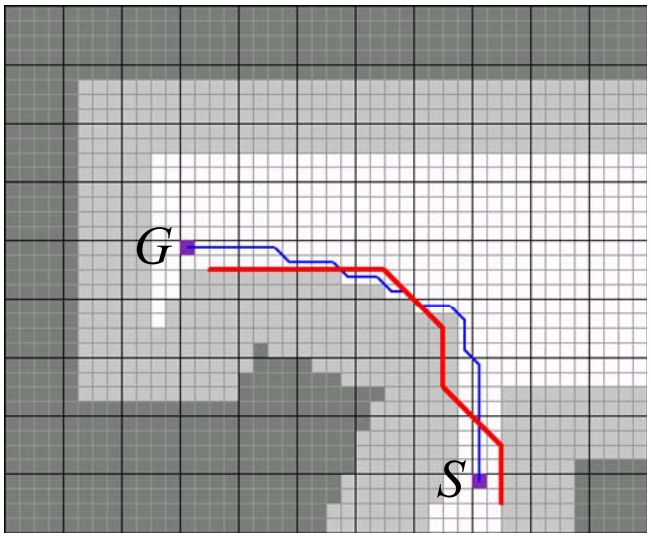

Fig. 6. An example of coarse-to-fine search. The block path is shown with thick line and the cell path with thin line.

should pass through strictly the connecting points of the diagonal block pairs and therefore, the fine search will be fail if any connecting point among them is occupied.

A final path at fine resolution is then obtained by applying A* search on the configuration map under the constraint that the searching space is restricted to the set of all grid cells belonging to the augmented block path. Figure 6 shows an example of coarse-to-fine path finding. The final path is very similar with the result from original A* search but the searching time is greatly reduced.

The proposed CFA* can be viewed as a kind of searching framework and it does not depend on a specific search algorithm. That is we can use any fast or optimal algorithm including A* for both coarse and fine search. For example CFA* might be combined with Anytime A* for further speed up. Processing time for building a block map and configuration map does not affect the online search as they can be computed and saved offline. In a dynamic environment with moving obstacles, we need to update both block map and configuration map. However, the update is very local and can be implemented efficiently.

Another way to speed up the CFA* search is to utilize a concept of partial refinement [17]. Instead of full refinement, we can consider some block on the block path as a local goal and refine only a part of block path to the local goal. Partial refinement is mostly useful in dynamic environment where replanning is frequently required.

IV. Experimental Results

In this section we present experimental performance results of the proposed coarse-to-fine path finder (CFA*) in comparison with low-level A* search. The experiments were performed on a large test map of size 935 × 315, which is shown in Fig. 7. The test map was obtained by decomposing

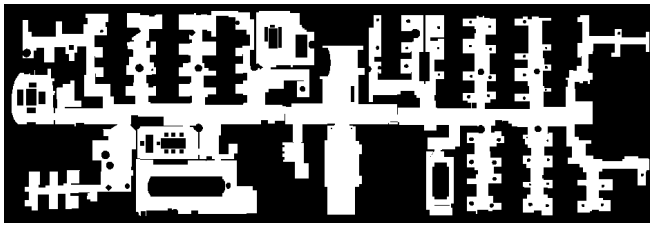

Fig. 7. Selected test map for the experiments.

our real office environment of 74.8 m × 25.2 m in uniform grid with the cell size of 8 cm × 8 cm. We first enlarged the occupied cells in the test map with $r = 3$ to compensate the size of a robot, resulting a test configuration map. The value of r was determined on the basis of the size of Pioneer 3-DX (40 cm × 44.5 cm), which is a popular mobile robot platform manufactured by Mobile Robots. The block size k was set to be 4, resulting a block map of size 234 × 79. Note that we can safely set k to be $r + 1$ from Eq. (2). We then applied CFA* and A* search for each of randomly generated 2,000 free location pairs on the test configuration map and recorded their runtimes and solution path lengths. For CFA* runtime we measured total processing time that included the time for coarse search and fine search as well as the time for augmenting a block path, time for restricting the search space of the configuration map by the cells within the augmented block path, and time for restoring the configuration map after fine search for the next online searches. Although our implementation of A* search was not fully optimized, it doesn't matter because CFA* uses the same A* implementation with low level A* for both coarse and fine search. The searching results were sorted by their A* path lengths and averaged for each interval of path length with interval size 20. All experiments were run on a 3GHz Pentium PC with Linux system.

A. Runtime Performance

Fig. 8(*a*) shows average runtime of the A* and CFA* for each interval of path length. We can see that the runtime of A* increases exponentially as the path length increases but the proposed CFA* does not. More detailed comparison is shown in Fig. 8(*b*), where average, minimal, and maximal runtime ratios of CFA* over A* for each interval of path length are plotted. As expected, the graphs show that the proposed CFA* is much faster than A* and the speed up becomes outstanding as the path length increases. The experimental result shows that the CFA* is roughly $n/10$ times faster than A* search for paths of length n on average.

B. Solution Quality

Fig. 8(*c*) shows the average and maximal ratios of the CFA* path length over A* path length for each interval of path length. The CFA* paths were shown to have only 0.1% error from optimal on average. In detail 79% of the time the

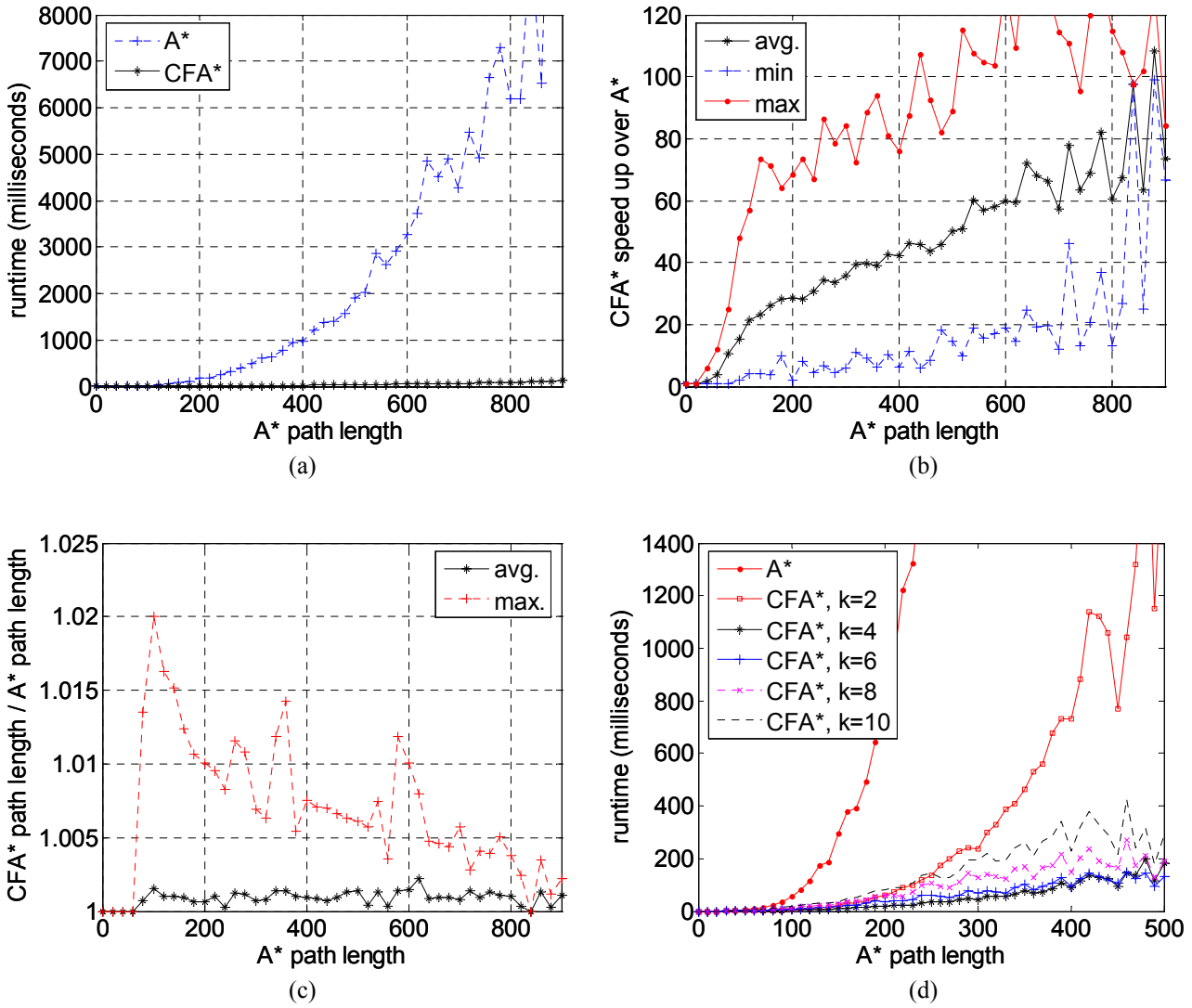

Fig. 8. Experimental results: (a) Runtime performance of CFA* and A*. (b) Runtime speed up of CFA* over A*. (c) Path quality of CFA*. (d) Runtime performance of CFA* with varying block size.

CFA* paths were optimal, and 99% of the time they were within 1% error from optimal in worst cases. The error of CFA* paths in percentile is computed by

$$e = \frac{l_{CFA*} - l_{A*}}{l_{A*}} \times 100, \tag{3}$$

where l_{CFA*} is the length of the CFA* path, and l_{A*} is the length of the optimal A* path. Even worst cases, it appears that it is within 2% error from optimal as shown in Fig. 8(*c*). The performance of the proposed CFA* is comparable with those of HPA* [16] and PRA* [17], where 10 times speed up and 1% error from optimal on average was reported.

The experimental result shows that CFA* can produce suboptimal solutions, although they are nearly optimal. A suboptimal solution can be obtained with CFA* when the optimal cell path is not within the augmented block path. In such cases we obtain locally suboptimal solutions. Figure 9 shows a typical example of suboptimal solution that can be found with CFA*.

C. Block Size

In order to see the effect of block size k on the performance of CFA*, we performed an experiment with an empty grid map of size 512 × 512, where every cell has free state. We chose an empty map so that the block size can be adjusted freely not being limited by Eq. (1). Fig. 8(*d*) shows average runtime of the CFA* with varying block sizes. It appears that the CFA* gives the best performance with $k = 4$. We also can see that the performance of CFA* does not increase any more with the block sizes larger than four. It is because a large block size also increases the search space for fine search, although it expedites the coarse search.

The optimal block size can vary depending on the size or

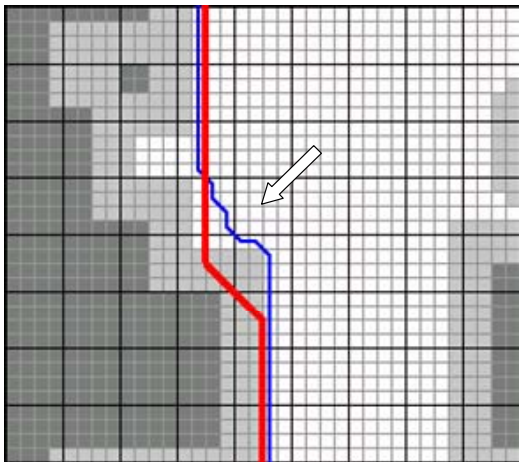

Fig. 9. An example of suboptimal CFA* path.

occupancy structure of a given map. However, performance of CFA* is not degraded so much with varying block size as shown in Fig. 8(*d*), and we empirically found that block size of $k = 3$ or $k = 4$ works well for most cases.

V. Conclusion

In this paper, we have presented a novel coarse-to-fine path finding method. The method was able to greatly reduce the searching time, while preserving the path quality. The real-time performance of our method makes it possible to implement efficient global path planners for mobile robots in a very large environment. In addition, the implementation is very easy and the method can be applicable for both static and dynamic environment.

One constraint of our method is that the block size has to be determined with an upper bound. It means it is hard to apply CFA* for the problems that the robot size is smaller than a grid cell and the map has very narrow passages and thin obstacles like a maze. In robotics applications, however, we usually prefer a map with fine resolution as it can represent the occupancy structure of the problem space in detail and enables fine control of robot motion. Also, small block size of three or four is sufficient for most problems as mentioned previously. It means that we need to enlarge the occupied cells only by two or three grid cells so that the CFA* can be applied effectively. We thus expect that the proposed method can be applicable effectively for most real problems, when considering the ordinary robot sizes and optional safety term.

References

[1] N. Nilsson, Principles of Artificial Intelligence, Tioga Publishing Company, 1980.

[2] C. W. Warren, "Fast Path Planning Using Modified A* Method," *In Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pp. 662–667, 1993.

[3] E. Hansen and S. Zilberstein, "Anytime heuristic search: Preliminary report," In *AAAI-96 Fall Symposium on Flexible Computation in Intelligent Systems: Results, Issues, and Opportunities*, pp. 55–59. 1996.

[4] R. Zhou and E. Hansen, "Multiple sequence alignment using anytime A*," in *Proc. of the National Conference on Artificial Intelligence* (*AAAI*), 2002, Student abstract.

[5] E. Hansen and R. Zhou, "Anytime Heuristic Search," *Journal of Artificial Intelligence Research*, Vol. 28, pp. 267–297, 2007.

[6] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

[7] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, MIT Press, 2003.

[8] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, Vol. 20, No. 5, pp. 378–400, May 2001.

[9] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[10] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[11] D. Ferguson and A. Stentz, "Anytime, Dynamic Planning in High-dimensional Search Space," in *Proc. IEEE International Conference on Robotics and Automation*, pp. 1310–1315, 2007.

[12] S. Kambhampati and L. Davis, "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 3, pp. 135–145, September 1986.

[13] D. Z. Chen, R. J. Szczerba, and J. J. Uhran Jr., "Planning Conditional Shortest Paths Through an Unknown Environment: A Framed-Quadtree Approach," in *Proc. the 1995 IEEE/RSJ Int. Conf. Intelligent Robots and System Human Interaction and Cooperation*, Vol. 3, pp. 33–38, 1995.

[14] R. Holte, M. Perez, R. Zimmer, and A. MacDonald, "Hierarchical A*: Searching Abstraction Hierarchies Efficiently," in *Proceedings AAAI-96*, pp. 530–535, 1996.

[15] S. Rabin, "A* aesthetic optimizations," In Mark Deloura, editor, *Game Programming Gems*, pp. 264–271, Charles River Media, 2000.

[16] A. Botea, M. Muller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of Game Development*, Vol. 1, No. 1, pp. 7–28, 2004.

[17] N. Sturtevant and M. Buro, "Partial pathfinding using map abstraction and refinement," *AAAI-05*, pp. 1392–1397, 2005.

[18] I. Ulrich and J. Borenstein, "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1572-1577, May 1998.