Beamlet-like Data Processing for Accelerated Path-Planning Using Multiscale Information of the Environment

Yibiao Lu, Xiaoming Huo, and Panagiotis Tsiotras

***Abstract*— We consider the deterministic path-planning problem dealing with the single-pair shortest path on a given graph. We propose a *multiscale version* of the well known A* algorithm (m-A*), which utilizes information of the environment at distinct scales. This information is collected via a bottom-up fusion method. Comparing with existing algorithms such as Dijkstra's or A*, the use of multiscale information leads to an improvement in terms of computational complexity.**

I. Introduction

We consider path-planning problems in a deterministic 2-D environment. Without loss of generality, we assume that the information about the environment is given via an $n \times n$ square image, where n is dyadic: $n = 2^J$ and J is a positive integer. Note that such an image-based formulation is well-adopted in the path-planning literature. In some publications, the image is called the *gridworld* [8]. For simplicity, we will assume squared images. We also assume that the image contains two types of pixels: gray pixels (representing non-traversable obstacles) and white pixels (representing traversable *free cells*). The path-planning problem is to find the shortest path between a given pair of *source* and *destination* cells. [1]

Two popular shortest-path search algorithms in a deterministic setting are Dijkstra's algorithm [5] and A* [9]. Both algorithms give the optimal path and can be considered as special versions of dynamic programming [14]. A* operates essentially the same as Dijkstra's algorithm, except that it uses a heuristic estimate to guide its search direction towards the most promising states.

To apply either algorithm, one first needs to construct the search graph. In this graph each free cell is defined to be a vertex and, correspondingly, it is connected only to its free four nearest-neighbors (*four-nearest-neighbor connectivity assumption*). However, both Dijkstra's and A* algorithms have the tendency to be slow as the space that needs to be searched increases. We propose a *multiscale A* (m-A*)* algorithm that takes advantage of the sparse information induced by the quadtree decomposition in a hierarchy of dyadic squares. We show (Section III) that such a strategy can reduce the order of computational complexity in the worst-case.

The objective of the proposed m-A* algorithm is to construct a smaller size graph on which the computational complexity of searching for the shortest path is efficiently reduced. The intuition of the problem size reduction is given as follows. A direct implementation of Dijkstra's or A* algorithm searches through all free cells in the environment. This can be overwhelmingly redundant: if the origin and destination vertices are in the upper-left and bottom-right quadrants respectively, it is not necessary to scan through all the free vertices in the upper-right and bottom-left quadrant. Instead, one only needs to consider the boundary white pixels of these two quadrants. (An illustration of this can be seen in Fig. 1.) Armed with this intuition, we develop a new dynamic programming algorithm for path planning in a cluttered environment, which takes advantage of preprocessed information that is organized in a multiscale fashion, analogous to the quadratic tree structure that has been used in beamlets [7].

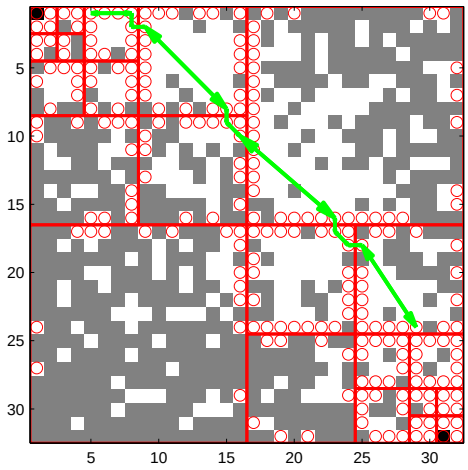


Fig. 1. The free cells that need to be considered in the newly proposed approach. Note that not all free cells need to be called in the multiscale way of running the Dijkstra's or A* algorithm. This is also an illustration of the vertices in the beamlet graph, which will be defined in Section II-B.

This paper follows up on recent advances in the area of multi-resolution path planning [1], [11], [15], [3], [4], [17], [12]. In particular, in [15], [3] wavelets were used to extract the spatial information from the environment that is most relevant to the current location of the vehicle. While wavelets have proven to be very useful in this context, they exhibit a small, fixed number of preferred orientations (horizontal, vertical, diagonal), and are better described as roughly isotropic. Beamlets, on the other hand, add two crucial elements

Yibiao Lu is currently a PhD candidate in the H. Milton Stewart School of Industrial and System Engineering, Georgia Institute of Technology, Atlanta, GA, 30332-0250, USA, e-mail: ylv3@gatech.edu

Professor Xiaoming Huo is with the H. Milton Stewart School of Industrial and System Engineering, Georgia Institute of Technology, Atlanta, GA, 30332-0250, USA, e-mail: huo@gatech.edu.

Professor Panagiotis Tsiotras is with D. Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250, USA, email:tsiotras@gatech.edu.

[1] In our setting, we consider path planning at the pixel level, and therefore the free cell is an equivalent concept as white pixel.

missing from wavelet processing that are crucial for path-planning problems: orientation and elongation information. As a result, they offer optimal approximations of straight lines in the plane [7]. The results of this paper stem from this simple observation.

II. Multiscale Path Planning Strategy with Preprocessed Information

A. Recursive Dyadic Partition of the Environment

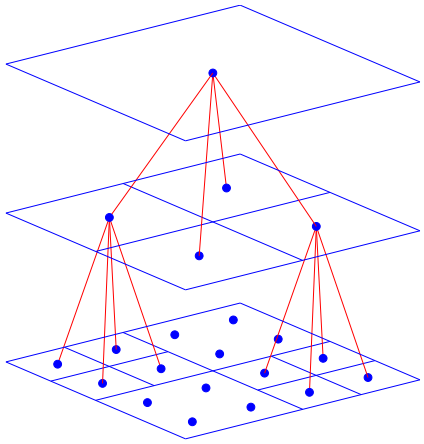
We describe two types of recursive dyadic partitioning (RDP) schemes used in the sequel. The first one is the *complete* version of RDP. We then introduce a *path-planning reduced* RDP (PFR-RDP), which will play an important role in defining the *beamlet graph*—the graphical structure that we will rely on.

The complete recursive dyadic partition can be described in a top-down fashion: a squared image is partitioned into 2×2 smaller d-squares, repeating the partitioning until the finest resolution of the image is reached. For future convenience, let s ($1 \le s \le J$) denote the *scale*. A dyadic square (called a *d-square* from this point on) at scale s, indexed by a, b ($1 \le a, b \le 2^s$) will be denoted by $q(s; a, b)$. We have $q(s; a, b) = \{(i, j) : 2^{J-s}(a-1)+1 \le i \le 2^{J-s}a, 2^{J-s}(b-1)+1 \le j \le 2^{J-s}b\}$. A d-square $q(s; a, b)$ at scale s can be partitioned into four scale $s+1$ d-squares; i.e., we have $q(s; a, b) = q(s+1; 2a-1, 2b-1) \cup q(s+1; 2a-1, 2b) \cup q(s+1; 2a, 2b-1) \cup q(s+1; 2a, 2b)$. We say that the d-square $q(s; a, b)$ has four children. The family of d-squares at all scales forms a quadratic tree.

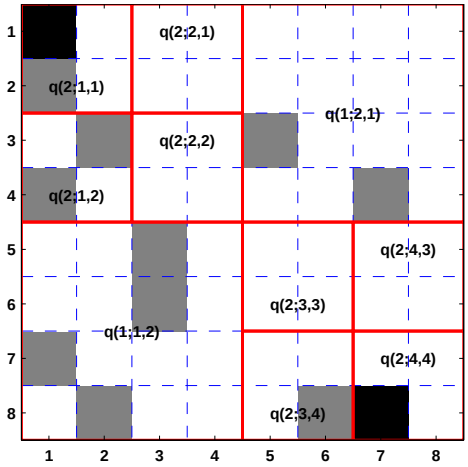
In the path-planning problem, for a given pair of origin and destination cells, only a part of the complete RDP is needed. We call this the *path-planning reduced RDP* (PFR-RDP). The PFR-RDP is generated as follows. The image is first partitioned into 2×2 smaller d-squares. If either the origin cell or the destination cell is in a smaller d-square, the quadratic (i.e. 2×2) partition of this d-square will continue, unless the finest resolution is reached. If a d-square contains neither the origin nor the destination, no further partition is done to this d-square. The resulting partition is a partial recursive dyadic partition—not all d-squares are partitioned to the finest resolution—and corresponds to a partial quadtree. Figure 2 shows a simple example of PFR-RDP and the corresponding *PFR-quadtree*.

B. Beamlet-like Connectivity

In the nearest neighbor graph, only the nearest neighbors are connected by edges. Here we introduce another type of connectivity that takes advantage of connections between far-away cells. We will see that such a connectivity, together with the aforementioned PFR-RDP, can reduce the computational complexity of the algorithm. For a fixed d-square, we only consider the free cells on the boundary of this d-square. A pair of free cells on the boundary of the d-square are said to be connected by a *beamlet* if and only if there exists a feasible path between the two. The beamlet is defined to be the shortest path connecting these two free cells. Note that such a definition is similar to the beamlets defined in [6].



(a) A path-planning reduced quadtree



(b) The corresponding partition of a squared image

Fig. 2. (a) A partial recursive dyadic partition and the corresponding PFR-quadtree (b) a partial recursive dyadic partition on a simple 8×8 image. The black cells are the source and destination. The gray cells are obstacles. The d-squares shown in figure are from the third layer in (a).

The *beamlet graph* is now defined as follows. First of all, a PFR-RDP of the n by n image is obtained. All the free cells on the boundaries of each d-square in the PFR-RDP are defined to be the vertices in the beamlet graph. Two vertices are connected under two conditions: (a) they are nearest neighbors; or (b) they are in a d-square in the PFR-RDP and there is a beamlet (i.e. a feasible path) that connects them. Within each d-square, the weight of an edge is the length of the shortest path connecting the two vertices. An example is shown in Fig. 1. The red grid is the partial dyadic partition of the environment corresponding to the PFR-RDP. The red circles are the free boundary cells, i.e. the vertices in the beamlet graph.

We need to compute the weights of the edges within all the d-squares in a PFR-RDP. This can be achieved by the following bottom-up fusion algorithm.

C. Bottom-Up Fusion Algorithm

The proposed multiscale path planning strategy requires the precomputed inter-distance between any pair of free boundary cells in each d-square. When $s = J$ or $J-1$, there

are one or four pixels in the d-square, respectively. Hence, it is straightforward to compute these distances. In the general case, recall that a d-square $q(s;a,b)$ can be partitioned into four smaller d-squares of scale $s+1$. If we already know the inter-distances between the free boundary cells within each of the smaller d-squares, and after considering the connectivity of free boundary cells that belong to neighbor d-squares, we can treat all free boundary cells of the four scale $s+1$ d-squares as vertices in a "fused" graph, and run Dijkstra's or A* algorithm on it. The distance between free cells from neighboring d-squares can be defined by direct neighbors. Since there are no more than $n2^{2-s}$ of these pixels, the search algorithm can be run efficiently. Figure 3 shows how this "fusion" of distance is conducted recursively.

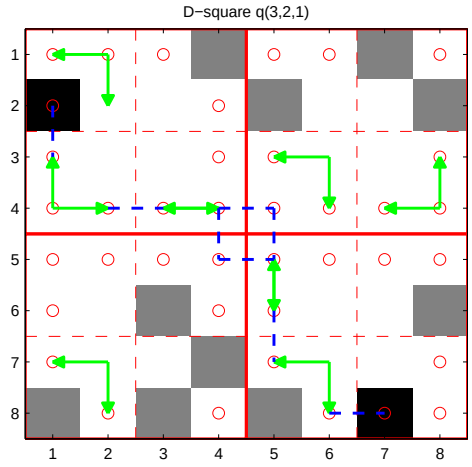


Fig. 3. Magnified view of the fusion in the d-square q(3,2,1) of Fig. 1. The fusion is conducted in a complete dyadic partition. Notice that the solid red grid stands for the partition corresponding to the second layer of the associated quadtree and the dashed red grid corresponds to the partition with respect to the third layer. The green arrow means the inter-distance between free boundary cells. The blue lines show the fusion process.

D. Multiscale A* Algorithm on the Beamlet Graph

Let $BG = (V, E)$ denote the beamlet graph, where V denotes the vertices, i.e. the free boundary cells of all the d-squares in the PFR-RDP, and E denotes the edges representing the shortest distance paths between pairs of free boundary cells.

Our algorithm can be considered to be *a multiscale extension of the A* algorithm (m-A*)*. The main steps mimic the standard A* procedure. For convenience, we repeat the main steps of the algorithm below. In particular, m-A* plans a path from the source cell $v_{origin} \in V$ to the destination cell $v_{destin} \in V$ in an image, where V denotes all the free cells on the boundaries of d-squares in a PFR-RDP. To do this, one stores an estimate $g(v)$ of the path length from the source to each cell v. Initially, we set $g(v) = \infty$ for all cells in V. The algorithm begins by updating the path length of the origin cell to be zero, then places this cell in a priority queue known as the *OPEN* list. Each element v in this queue is ordered according to the sum of its current path length from the start, $g(v)$, and a heuristic estimate of its path length to the destination, $h(v, v_{destin})$. The cell that has the minimum such sum is at the front of the priority queue. The heuristic $h(v, v_{destin})$ should underestimate the cost of the optimal path from v to v_{destin}. In our algorithm, the heuristic estimate is the usual L_1 distance.

E. Optimality of m-A*

The optimal path is identified by m-A* by running the A* algorithm on the beamlet graph. Recall that a search algorithm is complete if, given a search graph that has a destination, that algorithm always finds the destination; a search algorithm is optimal if it always finds the optimal path, if there exists one. We have the following theorem.

Theorem 1 *The m-A* algorithm, using an admissible heuristic, is complete and optimal.*

Proof: To show completeness, notice that the first step of m-A* computes the PFR-RDP and guarantees that the source and destination vertices are in the beamlet graph. Recall that the beamlet graph is assumed to have beamlet-like connectivity besides the four-nearest-neighbor connectivity. Therefore in the search tree, the number of children for each vertex will still be finite. According to the property of A* search algorithm [16], if an admissible heuristic function is used, m-A* is complete.

To show optimality, let p^* and p^{**} denote the shortest paths identified by running the A* algorithm on the nearest neighbor graph and the beamlet graph respectively. By definition, the optimal path has length $|p^*|$. It suffices to show that $|p^{**}| = |p^*|$. To this end, let S_{NNG} and S_{BG} be the sets of paths between the source and destination in the nearest neighbor graph and the beamlet graph, respectively. Because $p^{**} \in S_{NNG}$ and $p^* = \arg\min_{p \in S_{NNG}} |p|$, it follows that $|p^*| \leq |p^{**}|$. On the other hand, imagine that the same PFR-RDP is embedded on the nearest neighbor graph. Then p^* can be regarded as the chain of "path segments" on each d-square it passes through, and hence $p^* \in S_{BG}$. Since $p^{**} = \arg\min_{p \in S_{BG}} |p|$, it follows that $|p^{**}| \leq |p^*|$. ■

III. Complexity Analysis

A. Complexity of Fusion Algorithm

We would like to derive an upper bound of the algorithmic complexity, i.e., we consider the worst case. To this end, consider an environment represented by a grid of dimension $2m \times 2m$. At the first level below, the environment is subdivided to four d-squares of dimension $m \times m$. Let $T(m)$ denote the amount of computation required to find the inter-distances between all pairs of free boundary cells within an m by m d-square. A recursive relationship can be written as follows,

$$T(2m) = 4T(m) + f(2m) \tag{1}$$

where $f(m)$ denotes the effort for solving the all-pair shortest path problem in the fusion search graph. Note that there are $4(m-1)$ boundary pixels per each of the four smaller d-squares. Hence we have $|V| \leq 4 \times 4(m-1)$. Each edge belongs in either one of the following two categories:

(i) Edges connecting the free boundary cells within each smaller d-square.

(ii) Edges connecting the nearest-neighbor pixels between neighboring d-square.

In each d-square, there are at most $\binom{4m-4}{2}$ edges; there are at most $4m$ edges connecting free cells who are nearest neighbors, however not in the same d-square. Hence we have $|E| \leq 4\binom{4m-4}{2} + 4m < 32m^2$.

Lemma 2 *For the aforementioned $f(m)$ in* (1)*, we have $f(m) = O(m^3)$.*

Proof: Johnson's Algorithm [10] is the standard algorithm for solving the all-pair shortest paths problem, considering when the graph is sparse. If one assumes that an intermediate step of Johnson's algorithm (an implementation of the Dijkstra's algorithm) is done via Fibonacci heaps, then the overall complexity is $O(V^2 \log(V) + VE) = O(m^2 \log(m) + m^3) = O(m^3)$. This is the complexity of $f(m)$. ∎

To evaluate $T(m)$, we will need the following Master Theorem [2, Sect. 4.3].

Theorem 3 (Master Theorem) *Consider a recurrent relation (of an algorithm) in the following form:*

$$T(m) = a\, T\Big(\frac{m}{b}\Big) + f(m), \quad a \geq 1,\, b > 1.$$

where m is the size of the entire problem, a is the number of subproblems in the recursion, m/b is the size of each subproblem, (It is assumed that all subproblems are of the same size.) Let $f(m)$ be the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the subproblems. If the following two conditions are satisfied:

(a) $f(m) = O\left(m^{\log_b a+\epsilon}\right)$ *for some constant $\epsilon > 0$, and*

(b) $af\left(\frac{m}{b}\right) \leq cf(m)$ *for some constant $c < 1$ and sufficiently large m.*

Then we have $T(m) = O(f(m))$.

Note that the above is just a special instance of the more general Master Theorem [2]. For our purpose, the above special case is sufficient. We therefore have

Theorem 4 *For the $T(m)$ in* (1)*, we have $T(m) = O(f(m)) = O(m^3)$.*

Proof: Take $a = 4$ and $b = 2$, and apply Theorem 3. ∎

Therefore the complexity for preprocessing in a $n \times n$ gridworld is $O(n^3)$.

B. Complexity of Searching

We now consider the order of complexity for running A* (or Dijkstra's) algorithm on the beamlet graph and the 4-nearest-neighbor graph respectively. We consider the beamlet graph first. The PFR-RDP and the corresponding beamlet graph are shown in Fig. 4. The vertices in our beamlet graph

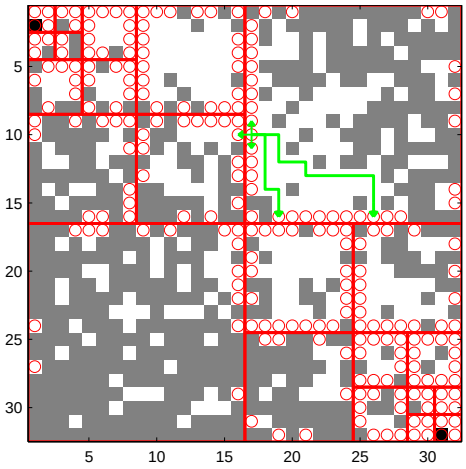


Fig. 4. Illustration of a beamlet graph in a 32×32 grid. The PFR-RDP are plotted in red lines. Free cells are marked by circles: they are free boundary cells of the d-squares in the PFR-RDP. If two free cells belonging to different d-squares are nearest neighbors, they also are connected. All free cells within the same d-square are considered connected, as long as a feasible path exists.

are the free boundary cells in all the d-squares in the PFR-RDP. For an $n \times n$ image, there are two scale-1 d-squares, and six scale-s d-squares when $s \geq 2$. For a scale-s d-square, there are at most $n2^{2-s}$ free boundary pixels. Hence the upper bound of the total number of vertices in the beamlet graph is $2 \times 2n + 6 \times n + 6 \times \frac{n}{2} + 6 \times \frac{n}{4} + \cdots = 4n + 6n + 3n + \frac{3}{2}n + \cdots \leq 16n$. $|V| = O(n)$. Therefore, $|E|$ cannot have order higher than $O(n^2)$. On the other hand, $|E|$ cannot be less than $O(n^2)$, since the two scale-1 d-squares already have $O(n^2)$ edges in the worst case.

Recall that the complexity of running Dijkstra's algorithm with Fibonacci heaps is $O(|E| + |V| \log |V|)$. We therefore have the following theorem.

Theorem 5 *The complexity of running A* or the Dijkstra's algorithm on the aforementioned beamlet graph is $O(n^2)$.*

Proof: The previous calculation has given us the following estimates: $|V| = 16n$ and $|E| \approx 8n^2$. Using the complexity of Dijkstra's algorithm, the upper bound of the complexity is $O(n^2 + n \log n) = O(n^2)$. Recall that A* and the Dijkstra's algorithm have the identical worst-case complexity [16]. ∎

For comparison, in the *nearest-neighbor graph* (NNG), (an illustration of which is provided in Fig. 5) we have $|V| = n^2$ and $|E| = 4n^2$, and one can easily establish the following theorem.

Theorem 6 *If A* or the Dijkstra's algorithm is run on the nearest-neighbor graph, the worst-case complexity is $O(4n^2 + n^2 \log n^2) = O(n^2 \log n)$.*

The proof is evident and is therefore skipped. Note (from Theorems 5 and 6) the reduction by a factor $\log n$ when compared with the m-A* algorithm.

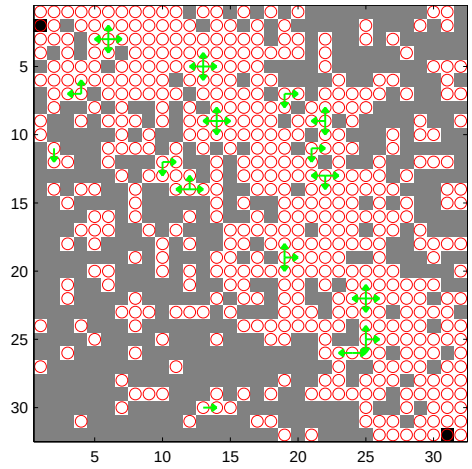


Fig. 5. A 32×32 image with all free cells marked by circles. There are around $O(n^2)$ circles. If two free cells are nearest neighbors, they are connected.

IV. Numerical Simulation Studies

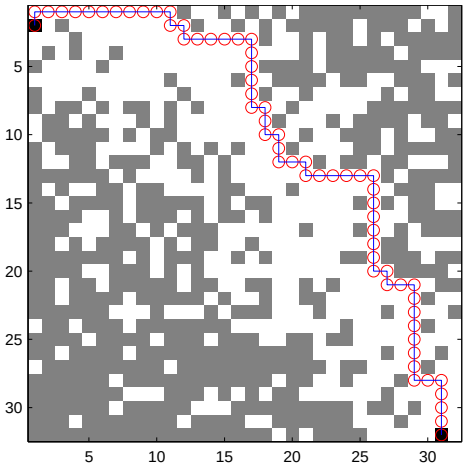
We provide numerical experiments to compare the proposed multiscale search algorithm (multiscale Dijkstra's and multiscale A*) with their single-scale counterparts. In the first simulation, the probability of a certain cell (denoted as (x, y) where $1 \leq x, y \leq n$,) to be a free cell is assigned to be $p(x, y) = \exp(-\gamma|y - x^2/n|)$, where the constant γ will be specified later. The intuition of this model is that pixels near the curve $y = x^2/n$ have higher probability to be free cells than the pixels far away from the same curve. We run Dijkstra's algorithm on the beamlet graph, as a special case of A* algorithm (i.e., setting the heuristic of the cost-to-go to be zero). Table I contains our simulation results when $\gamma = 1/15$. Figure 6 shows the shortest paths from the nearest-neighbor graph and the beamlet graph, respectively. The two paths are different in this example. Both paths have the same length nonetheless. A comparison of the corresponding running times is given in Table I.

TABLE I

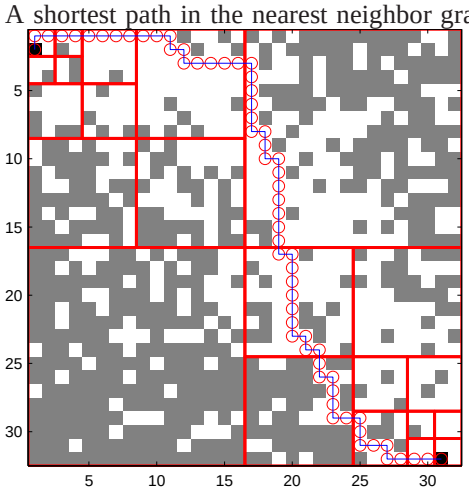
Multiscale Dijkstra's Algorithm Compared to Traditional Dijkstra In Simulation I with $\gamma = 1/15$.

ImageSize	64×64		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	0.616439	0.100009	6.17
Exp2	0.456536	0.126309	3.61
Exp3	0.466612	0.162247	2.88
Exp4	0.447084	0.112116	3.99
Exp5	0.433141	0.092936	4.66
ImageSize	128×128		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	1.315877	0.167777	7.84
Exp2	1.476583	0.283658	5.21
Exp3	1.374446	0.234752	5.86
Exp4	1.259515	0.175048	7.19
Exp5	1.728572	0.212332	8.14

The second simulation involves a more difficult situation. We generate the environment similarly to the previous case, but now we change the parabolic curve to a circle. Figures 7(a) and 7(b) show the shortest paths identified in



(a) A shortest path in the nearest neighbor graph



(b) A shortest path in the beamlet graph

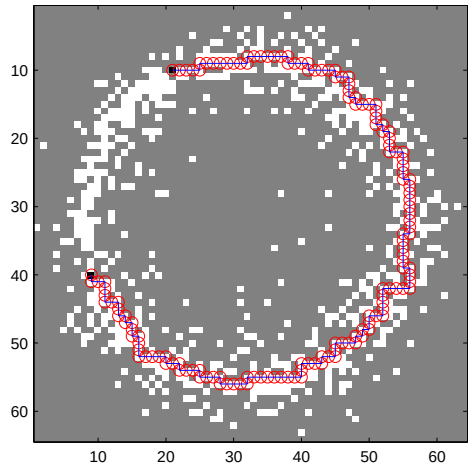
Fig. 6. (a) The shortest path of NNG in a 32×32 image; (b) The shortest-path of in the beamlet graph for the same image.

the NNG and the beamlet graph respectively. This time the two methods identified exactly the same optimal path. In Fig. 7(b), the partial dyadic partition corresponding to the PFR-quadtree is also shown, hoping that the reader would get a more clear view on this type of partition. Table II contains the computation times of the two algorithms for this simulation.

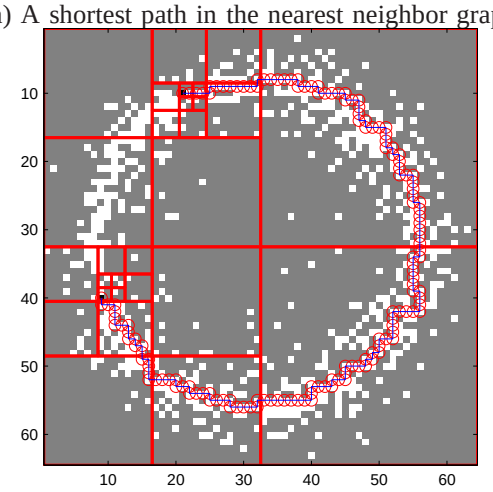
In this simulation, the proposed algorithm outperforms Dijkstra's algorithm. When the image size becomes larger, the computational time is reduced even more.

V. Conclusion

In this paper, we have discussed a newly-developed extension of the well-known A* path-planning algorithm based on a multiscale decomposition of the environment. A bottom-up fusion method is proposed as a multiscale strategy, which preprocesses the information and formulates the beamlet graph for the search algorithm. This multiscale A* algorithm (m-A*) provides a significant reduction in terms of computational time over the original A* algorithm. Incremental versions of the proposed baseline algorithm, along the same lines as in [13], as well as extensions to higher dimensions, are possible and are currently under investigation.



(a) A shortest path in the nearest neighbor graph



(b) A shortest path in the beamlet graph

Fig. 7. (a) The shortest path in the NNG; (b) The partial partition corresponding to the PFR-RDP and the shortest path in the beamlet graph. In both images, the gray cells means "obstacles", and the black dots are the starting point and destination.

TABLE II

MULTISCALE DIJKSTRA'S ALGORITHM COMPARED TO TRADITIONAL DIJKSTRA IN SIMULATION II WITH $\gamma = 1/500$.

ImageSize	64×64		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	0.18965	0.022204	8.54
Exp2	0.163025	0.022509	7.24
Exp3	0.269925	0.026684	10.12
Exp4	0.217668	0.025778	8.44
Exp5	0.114686	0.016336	7.02
ImageSize	128×128		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	0.511047	0.054572	9.36
Exp2	0.548823	0.065516	8.38
Exp3	1.222879	0.066844	18.29
Exp4	1.175211	0.067509	17.41
Exp5	0.640812	0.079966	8.01

Acknowledgement: This research was supported in part by NSF award no. CMMI-0856565 and NASA award no. NNX08AB94A.

REFERENCES

[1] S. Behnke, *Local Multiresolutin Path Planning*, ser. Lecture Notes in Computer Science. Berlin: Springer, 2004, vol. 3020, pp. 332–343.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press and McGraw-Hill, 2001.

[3] R. Cowlagi and P. Tsiotras, "Beyond quadtrees: Cell decomposition for path planning using the wavelet transform," in *46th IEEE Conference on Decision and Control*, New Orleans, LA, Dec. 12–14 2007, pp. 1392–1397.

[4] ——, "Multiresolution path planning with wavelets: A local replanning approach," in *American Control Conference*, Seattle, WA, June 11-13 2008, pp. 1220–1225.

[5] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[6] D. Donoho and X. Huo, "Beamlets pyramids: A new form of multiresolution analysis, suited for extracting lines, curves and objects from very noise image data," in *Proceedings of SPIE*, vol. 4119, no. 1, July 2000, pp. 434–444.

[7] ——, *Multiscale and Multiresolution Methods*. Spring, 2002, vol. 20, ch. Beamlets and multiscale image analysis, pp. 149–196.

[8] D. Ferguson, M. Likhachev, and T. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[9] P. Hart, N. Nilsson, and B. Rafael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE trans. Sys. Sci. and Cyb.*, vol. 4, pp. 100–107, 1968.

[10] D. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[11] D. Jung and P.Tsiotras, "Multiresolution on-line path planning for small unmanned aerial vehicles," in *American Control Conference*, Seattle, WA, June 11-13 2008, pp. 2744–2749.

[12] S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE Journal of Robotics and Automation*, vol. 2, no. 3, pp. 135–145, 1986.

[13] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence Journal*, vol. 155, no. 1-2, pp. 93–146, 2004.

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

[15] P. Tsiotras and E. Bakolas, "A hierarchical on-line path-planning scheme using wavelets," in *European Control Conference*, Kos, Greece, July 2–5 2007, pp. 2806–2812.

[16] P. Winston, "Search complexity," in *Artificial Intelligence Lecture Notes*, Fall 2003.

[17] A. Yahja, A. Stentz, S. Singh, and B. L. Brumit, "Framed-quadtree path planning for mobile robots operating in sparse environments," in *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*. Leuven, Belgium: IEEE, May 1998, pp. 650–655.