The M2M Pathfinding Algorithm Based on the Idea of Granular Computing

Haifeng Wan[1], Yingpeng Zhang[1], Shengzhou Luo[2], Ruijie Liu[1], Wensheng Ye[1]
1) South China University of Technology, Guangzhou, China
2) Shenzhen Institute of Advanced Technology Chinese Academy of Sciences, Shenzhen ,China
Email: wanhaifeng09@gmail.com

Abstract

Macro-to-micro (M2M) model is an implementation model that inherits the GrC idea and extends it to some additional highly desirable characteristics. In this paper we introduce an effective pathfinding algorithm based on the M2M model. This algorithm takes O(n) time to preprocess, constructing the M2M data structure. Such hierarchical structure occupies O(n) bit memory space and can be updated in O(1) expected time to handle changes. Although the resulting path is not always the shortest one, it can make a trade-off between accuracy and time cost by adjusting a parameter - range value to satisfy various applications. At last, we will discuss the advantages of the M2M pathfinding algorithm (M2M-PF) and demonstrate the academic and applied prospect of M2M model.

1. Introduction

Pathfinding is essential to many domains, such as multi-agent system, RTS (Real Time Strategy) game, Internet routing system and car navigation system. Modern hardware has helped pathfinding take one big step forward, with additional processing power and memory. However, the additional complexity of next-gen environments has caused us to take one huge leap backwards; dynamic obstacles, changing environments, 3D sence and large crowds are sending agent's navigational capabilities back a few generations.

Pathfinding tasks on large maps, especially on 3D maps, have a high demand for efficiency of pathfinding algorithm. Additionally, the constant patial changes of the maps, including changes of arcs and vertices, require better dynamicity of pathfinding algorithm.

In this paper we introduce a hierarchical pathfinding algorithm based on the M2M model which is inherited from the GrC ideas. This algorithm takes O(n) time to preprocess, constructing the M2M data structure which occupies O(n) bit memory space. Then it can quickly query the path between any two nodes of the graph. A desirable property is that such data structure can be updated in O(1) expected time to handle changes of an arc or a vertex of the original graph. Although the resulting path is not always the shortest one, its solution quality is very close to optimal. Moreover, it can make a trade-off between accuracy and time cost by adjusting a parameter – *range value* to satisfy various applications.

The basic ideas of Granular Computing (GrC) have been explored in many fields, such as artificial intelligence, interval analysis, quantization, rough set theory, divide and conquer, cluster analysis, machine learning, databases, etc [1]. The M2M model is a concrete model and gives guidance and discipline about how to design and implement the GrC ideas such as multiple views and step-wise refinement.

Furthermore, the M2M model extends the ideas of GrC and emphasizes some additional highly desirable characteristics which are coherent with the behavior of human being but rarely mentioned in other computing models. These desirable characteristics of the M2M model are preprocessing sharing, parallelism, robustness (fault-tolerance) and dynamicity etc.

The general data structure of the M2M model has been used to solve many point-set-based problems effectively, such as nearest neighbor search problem [4] and convex hull problem [5] etc.

2.Related Work

Dijkstra is the most classic single-source shortest path algorithm but has weakness for shrinking search space. Later, A* [6] becomes the most widely used pathfinding algorithm for its heuristic method. And Iterative-Deepening A* (IDA*) [7] is an improvement of A*. A* and IDA* have been explored thoroughly with regard to finding optimal paths in a well-known and stationary environment. D*-Lite [8] is able to do limited replanning if the world changes, but it can't handle changes caused by a moving target.

In recent years, building up abstraction and hierarchical levels of maps is certainly an effective approach to the pathfinding problem. Holte [9] uses abstraction and refinement in a graph representation of the problem space to reduce the time. Abstraction and refinement can also be used to build heuristic functions for unabstracted space [10]. Instead of abstracting directly from a graph representation, Hierarchical Path- finding A* (HPA*) [2] overlays a map with large sectors and calculates optimal paths between limited sets of entrances and exports to the sectors. Through the process of smoothing, the optimal paths can be obtained. Partial-Refinement A* (PRA*) [3] uses automatic state-space abstraction to improve traditional pathfinding algorithm. It uses the hierarchy of map abstractions and computes abstract paths with this hierarchy.

Based on the assumption that a given road network does not change very often. Many literatures [11] [12] introduce approaches with considerable memory usage and time cost for precomputing the shortest path between some nodes to enhance the speed of the subsequent pathfinding. Most of these algorithms guarantee the exact shorest path. But they lack dynamicity for the reason that the change of any arc or vertex will probably lead to the invalidity of the precomputed shortest path.

3.Terminology

Level: The hierarchical data structure of graph $G=(V,E)$ consists of levels of different granularity. A coarser-grained level presents abstracted data classification in macro view, while a finer-grained level presents detailed data classification in micro view.

Part: Part is defined as a subset of the data points which is similar to the concept of grain of GrC. At level k, graph $G=(V,E)$ is divided into squares of the same size. All the data points in the same square belong to this part.

4.The General M2M Data Structure

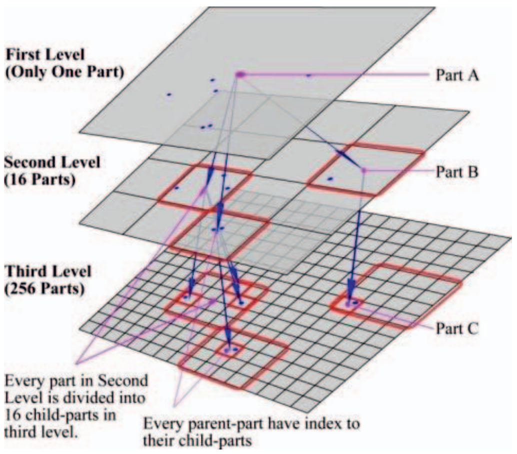
First Level
(Only One Part)
Part A
Second Level
(16 Parts)
Part B
Third Level
(256 Parts)
Part C
Every part in Second Level is divided into 16 child-parts in third level.
Every parent-part have index to their child-parts

Fig.1. General M2M data structure

An example of M2M data structure is shown in Fig.1. The implementation details of the M2M model may be different according to different situations. However, some basical requirements must be satisfied to meet the demands of the GrC idea and the M2M model.

1. Given a query point and a specified level, searching the part which a point belongs to takes O(1) time. This property ensures that the data structure can change granularity easily (GrC Requirement)
2. Insertion or deletion of a data point should take O(1) time. This property ensures that it is a dynamic structure (M2M Requirement).
3. Given the index of the part, it takes O(n) time to visit every child-part of the given part, where n is the number of the child-part. This property ensures the availability and speed of the step-wise refinement (GrC Requirement) process.
4. The time complexity of preprocessing should be O(n) and it supports parallel calculation. (M2M Requirement)

5.The M2M Data Structure for Pathfinding

The data structure of M2M-PF is based on the general M2M structure. And it adds the information of connection among parts to all levels of different granularities. So we can be guided by the connective relation of macro (coarse-grained) level when dealing with the micro (fine-grained) level.

6.The M2M Pathfinding algorithm

Just as the general process of other M2M algorithms, the process of M2M-PF mainly includes two procedures: preprocessing and query. The preprocessing procedure runs just only once to set up the M2M data structure for pathfinding and the update operation will run when the map is changed a little. After the preprocessing procedure, once the search task is given, the query procedure can be executed in a short time.

7.The Preprocessing Procedure of M2M-PF

The preprocessing analyzes connected components of each part and constructs a hierarchical data structure from the finest-grained level to the coarsest-grained level.

At level k, graph $G=(V,E)$ is divided into square parts of the same size: $Part_1(k)$, $Part_2(k)$, ..., $Part_n(k)$. Each Part corresponds to one induced subgraph $G_i(V,E)$ where $V \subseteq Part_i(k)$. Then each induced subgraph is divided into several strongly connected components. Each connected component in $Part_i(k)$ is abstracted to a point located at the center of $Part_j(k+1)$ ($Part_i(k)$ belongs to $Part_j(k+1)$ in the next coarser-grained level). Considering the situation of an undirected graph, given two points in $Part_i(k)$: p_0 and p_1, there are three scenarios listed as follows:

Case 1: If p_0 is an isolated point, which means it is also a connected component, it is abstracted to a point located at the center of $Part_j(k+1)$.

Case 2: If p_0 and p_1 belong to the same connected component, they merge into one point located at the center of Part$_j$(k +1).

Case 3: If p_0 and p_1 belong to different connected components, they are abstracted to different points q_0 and q_1 respectively, where q_0 and q_1 are coincident points located at the center of Part$_j$(k +1).

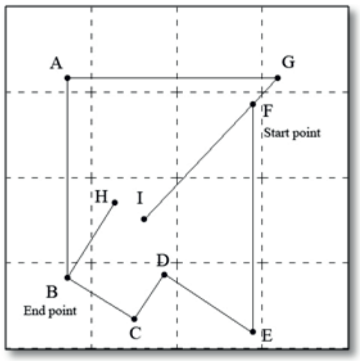

Fig.2. The original connected graph

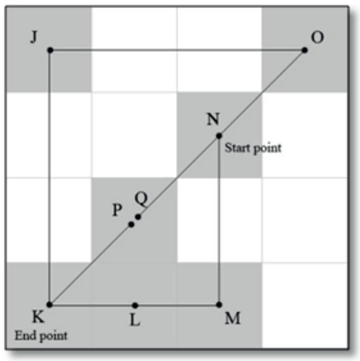

Fig.3. The first step of preprocessing

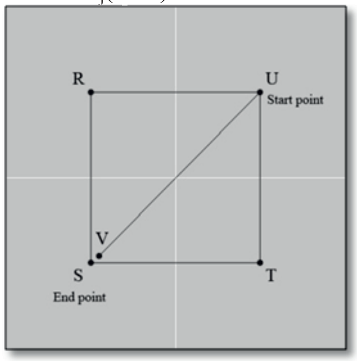

Fig.4. The second step of preprocessing

This strategy ensures that different connected components will not be mixed up during the process of abstraction. Let's take Fig.2 as an example. Suppose we are going to find the shortest path between the point *F* and *B*.

Firstly, the graph is divided into 16 parts. Note that point *C* and *D* belong to the same connected component, while unconnected point *H* and *I* belong to different connected components. According to the strategy of abstraction, connected components {A}, {B}, {C, D}, {E}, {F}, {G}, {H}, {I} (Fig.2) are abstracted to {J}, {K}, {L}, {M}, {N}, {O}, {P}, {Q}, respectively (Fig.3). In Fig.3, point *P* and *Q* both locate at the center of the part. Here , we draw them as separate points just to show they belong to different connected components. If we regard the point *H* and *I* as the same point, then it means that a virtual path is created from *F* to *B*. As a result, the path *F->I->H->B* will probably be considered as the shortest path. That's obviously wrong, for the path *H->I* is actually non-existent!

In the second step of preprocessing, the graph is divided into 4 parts as the granularity becomes coarser-grained. Similar operations will be performed on the basis of the previous process until the graph becomes sufficiently simple. According to the strategy of abstraction, connected components {J}, {K, L, P}, {M}, {N, O}, {Q} (Fig.3) are abstracted to {R}, {S}, {T}, {U}, {V} (Fig.4).

In this way, a connected graph with thousands of points will be simplified to a considerably simple graph (Fig.5), which is beneficial to the fast process of the query steps.

8.The Query Procedure of M2M-PF

The query procedure will query the shortest path based on the preprocessing. It is processed from the coarsest-grained level to the finest-grained level, shrinking the search space by excluding parts where the shortest path rarely goes though. Finally we can use the traditional pathfinding algorithm (such as A*or Dijkstra) to obtain the shortest path in a very small search space.

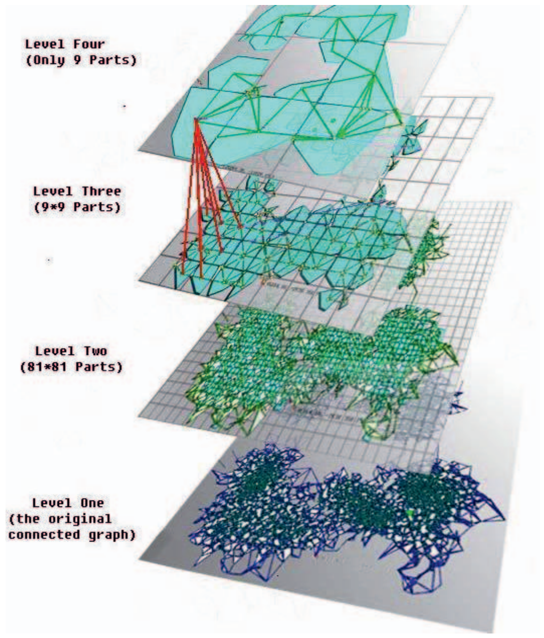

Fig.5. The steps of preprocessing

Before introducing the query procedure, we give some definitions:

$PV_{V_i}(k)$: The parent vertex of V_i in level *k*.

$SP(V_1, V_2)$: The length of the shortest path between vertices V_1 and V_2.

RV: A parameter called *range value* which determines the size of search space.

$PLT(V_1, V_2, RV)$: The path length threshold between vertices V_1 and V_2 with *range value RV*.

$Reach_{V_i}(S, E)$: The length of the shortest path from the start point *S* and the end point *E* through the vertex *Vi*.

$PVS_k(V_1, V_2, RV)$: The preserved vertex set of level *k* between vertices V_1 and V_2 with *range value RV*.

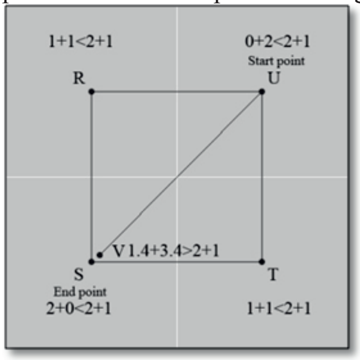

Fig.6. Step 1 of the query process in level 3

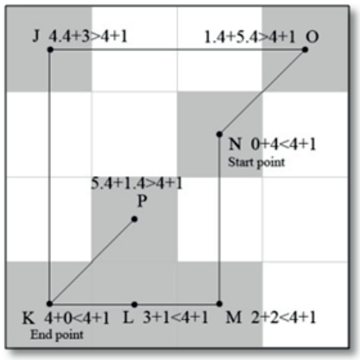

Fig.7. Step 2 of the query process in level 2

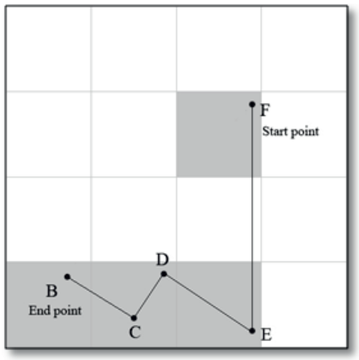

Fig.8. Step 3 of the query process in level 1

According to these definitions, we can deduce the following equations:

$$PLT(S, E, RV) = SP(S, E) + RV$$

$$Reach_{V_i}(S, E) = SP(S, V_i) + SP(V_i, E)$$

Here, we introduce the definition of the strategy of search space preservation. It will be discussed in detail later.

The preserved vertex set of graph G at level k is:

$$PVS_k(S, E, RV) = \left\{ V_i \in GV_k \,\middle|\, \begin{array}{l} Reach_{V_i}(PV_S(k), PV_E(k)) \\ \leq PLT(PV_S(k), PV_E(k), RV) \end{array} \right\},$$

where *k* is the sequence number of a level, GV_k is the vertex set of level *k* graph.

Now, we explain the query procedure in details by continueing the example in previous section. We define the length unit of graph *G* in level *k* as the side length of a part in this level.

In Fig.6 the start point is $PV_F(3)$=*U* and the end point is $PV_E(3)$=S. First we calculate *SP(U,S)*=2 by Dijkstra in level 3. Then we can set *range value* to be 1. So, $PLT(U, S, RV)$ = *SP(U,S)+RV* = 2+1. For point V, $Reach_V(U, S)$= 1.4+3.4>$PLT(U, S, RV)$= 2+1, so the search space won't preserve it for the next finer-grained level. But for point *R*, the search space will preserve it, because $Reach_R(U, S)$= 1+1$\leqslant$$PLT(U, S, RV)$= 2+1. Similarly, the search space contains point *S*, *T* and *U*. So we can get Fig.7 at the next finer-grained level based on the search space preserved in this step.

In Fig.7 the start point is $PV_F(2)$=N and the end point is $PV_E(2)$ =K. And at this level we get *SP(N,K)*=4 after calculation. We also set *range value* to be 1.Similarly, the search space for the next finer-grained won't contain point *J*, *O*, *P*. But it will contain point *K*, *L*, *M*, *N*.

So we get the Fig.8 in the next finer-grained level. Obviosuly the search space is much smaller now. In Fig.8 we can easily obtain the resulting path (*F->E->D->C->B*) by the traditional pathfinding algorithm. However, it is not the shortest path (we will give an explaination in the next section).

Fig.9, 10, 11, 12 based on the problem shown in Fig.5 explain the query procedure more clearly.

Now suppose that we want to search for the shortest path between the orange point in the lower right corner and the red point in the upper left corner (Fig.9).

First in the coarsest-grained level, the algorithm preserves the search space (those 6 parts bounded by the black lines in Fig.9) by the strategy of search space preservation. Then, these 6 parts will be handled as the global search space for the next finer-grained level. Hence, the search space in the next finer-grained level will be smaller and finer-grained (those 12 smaller parts bounded by the dark lines in Fig.10).

Repeat this procedure as the search space preserved becomes smaller and smaller, finer-grained and finer-grained. Finally, we get the resulting path by the traditional pathfinding algorithms at the finest-grained level, where the search space (those 29 small parts bounded by the dark lines in Fig.12) is way smaller than the primitive one (the original graph).

In brief, in level k we first calculate *SP(S,E)* by Dijkstra, then we preserve search space based on a threshold which is $PLT(S, E, RV)$.

So the query steps can be treated as a process of shrinking search space and step-wise refinement search which is shown in Fig. 13.

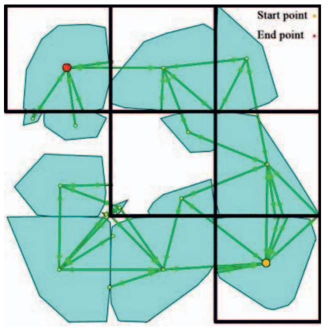

Fig.9. Step 1 of the query process

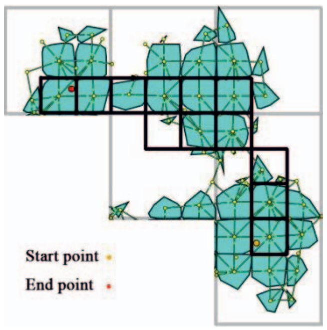

Fig.10. Step 2 of the query process

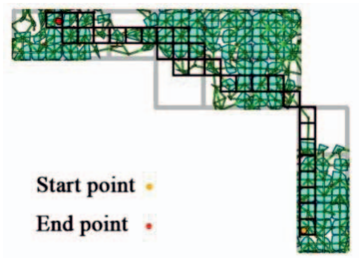

Fig.11. Step 3 of the query process

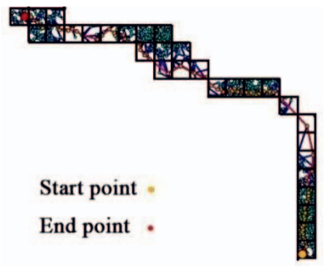

Fig.12. Step 4 of the query process

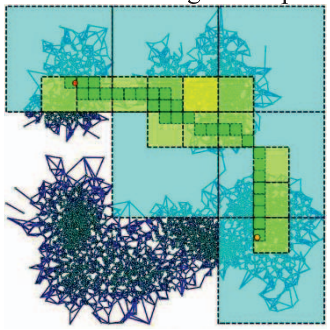
Fig.13. The behavior of shrinking search space and step-wise refinement (Blue->Yellow->Green)

It is necessary to implement the search space preservation strategy to preserve the search space from the coarsest-grained level to the finest-grained level. On one hand, at the coarse-grained level, the reduction of search space is based on large parts, so we can exclude a large region quickly. Otherwise, we have to deal with much more small parts in the same region if this job is done at the finer-grained level, which will be much slower. On the other hand, since reduction of small parts cannot be done on coarse-grained level, it is necessary to go through finer-grained level to exclude finer-grained region.

In those real-time pathfinding situations such as RTS games, M2M-PF is similar to some other hierarchical pathfinding algorithms. They first obtain a path in macro level, without considering micro levels. The search in a certain part in micro level won't start until an agent moves to this part. The technique of interleaving acting and planning [3] spreads the cost of path computation more evenly over the path execution time. Besides, it helps reduce the memory usage, for the memory usage of a path at macro level is much smaller than that of a path at micro level. Moreover, it effectively avoids wastes of computation resources caused by the agent if the agent changes the target halfway.

In summery, the preprocessing and query steps presented here are based on the method that of solving the problem at multiple levels of abstraction, with between-levels guidance operating to reduce overall search.

9.The Strategy of Search Space Preservation and the relationship between Accuracy and Time Cost

As many other M2M algorithms, the search space preservation strategy is the key part of M2M-PF. It affects the accuracy and the time cost of M2M-PF.

For the M2M nearest neighbor search algorithm [4], this strategy affects whether the actual nearest point loses or not. For the M2M convex hull algorithm [5], this strategy affects whether the result is the actual convex hull of the point set or the approximation to the actual convex hull and how close to the actual convex hull. For M2M-PF, this strategy also affects whether the path is the shortest one, and if it merely approximates to the shortest path then how large the difference between them is.

Firstly, we explain the role of *range value* mentioned above .

We preserve the search space based on a threshold which is the sum of the length of the shortest path and *range value*, by bidirectional Dijkstra algorithm with O(n) worst time, where n is the number of the vertices of that level. Note that, the vertices preserved set $PVS_k(S,E,RV)$ contains those vertices traversed by any path whose length is within the threshold. For each vertex $V_i \in GV_k$, if there exists a path from *S* to *E* through V_i whose length is less than $PLT(S,E,RV)$, $Reach_{V_i}(S,E)$ must be less than $PLT(S,E,RV)$. So the vertex V_i will be preserved. Hence, the bigger *range value* is, the bigger $PLT(S,E,RV)$ is, which means the larger the search space preserved is.

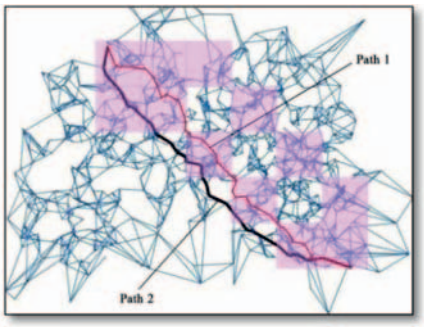

Fig.14. Range value is set to be small

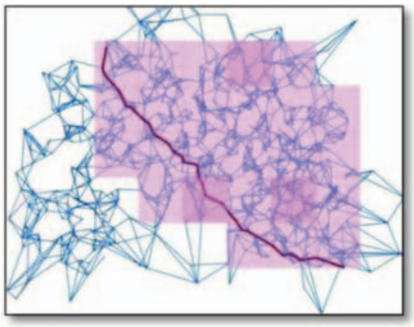

Fig.15. Range value is set to be medium

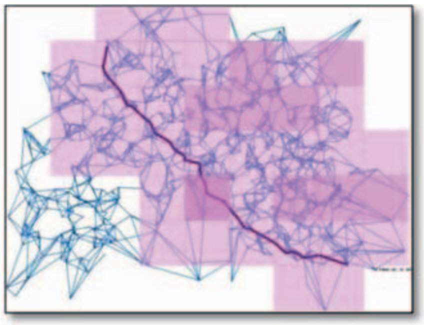

Fig.16. Range value is set to be big

Secondly,we explain how the *range value* determines the size of the search space preserved.

In Fig.6 if the *range value* is set to be 1, the search space preserved for the next finer-grained level contains the point *R*, *S*, *T*, *U*. If *range value* is set to be 3, the search space preserved for the next finer-grained level contains the point *R* (1+1<2+3), *S* (2+0<2+3), *T* (1+1<2+3), *U* (0+2<2+3) and *V* (1.4+3.4<2+3).

In Fig.7 if *range value* is set to be 1, the search space preserved for the next finer-grained level contains the point K, L, M, N. If the *range value* is set to be 3, the searching space contains more points: *K*, *L*, *M*, *N*, *O* (1.4+5.4<4+3), *P* (5.4+1<4+3). Further more, if the *range value* is set to be 4, the searching space contains all points: *K*, *L*, *M*, *N*, *O*, *P*, *J* (4.4+3<4+4).

Hence, if *range value* is bigger, the search space preserved will be bigger. In fact when *range value* is set to be zero, the search space preserved is just the points through which the shortest path at this level traverses.

Finally, we explain how *range value* affects the accuracy of the resulting path.

In Fig.2 we obtain the shortest path from the start point *F* to the end point *B* (*F->G->A->B*) by the traditional algorithm such as A* or Dijkstra. But the resulting path obtained by M2M-PF (*F->E->D->C->B*) is not the shortest path. Hence, when *range value* is very small, the resulting path is perhaps not the shortest one, but it is just a little longer than the shortest path. In constrast, if *range value* is set to be 4 or even bigger in Fig.7, the search space preserved will contain point *J* and *O*, so that we will obtain the shortest path (*F->G->A->B*) in the next finer-grained level.

Fig.14, 15, 16 explain this principle in practical application. In Fig.14 the red line (Path 1) is the resulting path obtained by M2M-PF. It is obtained in the pink squares which are the search space preserved. The black line (Path 2) is the shortest path obtained by Dijkstra. In this situation the red line is a little longer than the black line, for the space the optimal path go through is not preserved in the search space of M2M-PF. In Fig.15 the black line and the red line are coincidence because the space the shortest path goes through is preserved in the search space. In Fig.16 the black line and the red line are also coincidence for the same reason.

Fig. 14 and Fig. 15 show that we need not to choose the biggest *range value* to obtain the shortest path by M2M-PF, just a proper one will do.

In summary, we draw the conclusion that if *range value* is smaller, the search space preserved is smaller and the probability of getting the shortest path is in turn lower. But it consumes less time and the resulting path is just a little bit longer than the shortest one.

So between accuracy and time cost we have to make a balance. And in most cases like RTS games and GIS systems, it is more likely to sacrifice the accuracy to save time.

10.Experiments

We set up a random connected graph generator for generating maps of different distributions as test data and use Dijkstra as the benchmark algorithm. The experimental results show that M2M-PF is much faster than the classical Dijkstra algorithm at the expense of losing a little accuracy. The more the nodes are, the bigger the speed gap between them is. Note that these experiments neglect the time cost of preprocessing and haven’t use the technique of interleaving acting and planning.

Fig.17 and Fig.18 show the comparison of path length and time cost between M2M-PF and Dijkstra algorithm in maps of uniform distribution. Fig.19 shows the memory usage of the hierarchical data structure of M2M-PF.

Fig.17 and Fig.18 clearly show the feature of trade-off between accuracy and time cost. Compared with Dijkstra algorithm, when *range value* is 0.5, the efficiency ratio is from 20 to 40 while the accuracy ratio is from 1.06 to 1.12. When *range value* is 2, the efficiency ratio is from 3 to 8, but the accuracy ratio is almost 1, which means M2M-PF gets the optimal path in this situation.

We haven’t made a comparison of the efficiency between M2M-PF and other hierarchical algorithms, for these algorithms haven’t universally acknowledged implementations and the source codes are not published. The differences between them will be discussed in detail in the next section.

Besides, to evaluate the performance of our approach, computational experiments have been conducted on the Beijing road network.It draws a similar conclusion just like the maps of different distributions

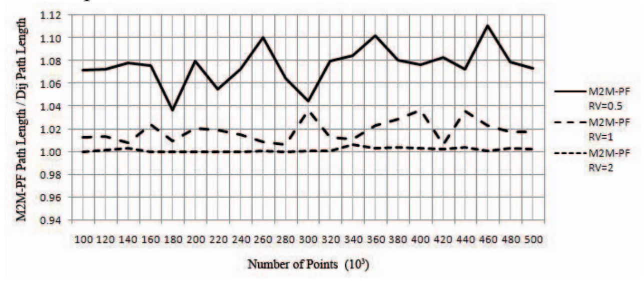

Fig.17. The ratio of path length between M2M-PF and Dijkstra

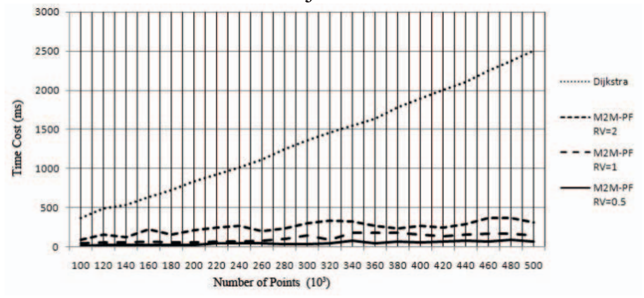

Fig.18. The comparison of time cost of M2M-PF and Dijkstra

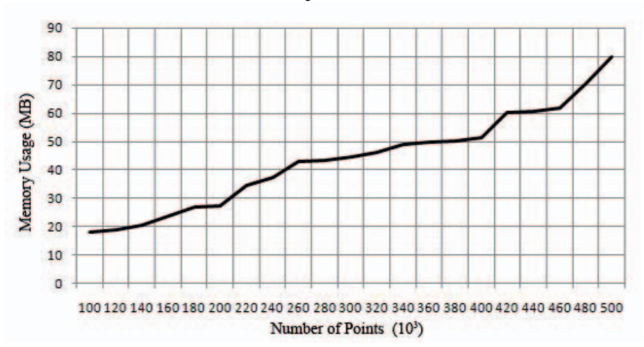

Fig.19. The memory usage of the hierarchical data structure of M2M-PF

11.The Advantages of M2M-PF

For M2M-PF is a hierarchical pathfinding algorithm, it derives some common strengths and weaknesses that hierarchical algorithms hold. All hierarchical algorithms take time to preprocess, constructing hierarchical connected graph. But after preprocessing, they are much faster than non-hierarchical algorithms.

We make a comparison between M2M-PF and other non-hierarchical algorithms. Since M2M-PF takes time to preprocess, non-hierarchical algorithms are more efficient when few pathfinding tasks are assigned or the map changes a lot from time to time. In these situations, M2M-PF is less efficient, for the time cost of preprocessing constitutes a large proportion of the total time cost.

But for those multi-agent pathfinding tasks on maps with complex topology structure and little changes, such as traffic map, game map and net topological graph, M2M-PF demonstrates its advantages. Because once the preprocessing procedure is completed, it can search for the path based on the hierarchical structure. Its dynamic data structure enables it to handle a changing map without redoing the preprocessing. Furthermore, the path is built in real time, interleaving acting and planning, so that the cost of path computation is spread more evenly. (Actually the experiment doesn't demonstrate the advantage brought by real-time pathfinding.) All these features contribute to the high efficiency of M2M-PF, as the experiment shows.

In comparison with other preprocessing-based hierarchical pathfinding algorithms, M2M-PF derives many desirable properties from the M2M model which are significant in practical application.

High parallelism

The time complexity of preprocessing is O(n), which consumes much time by sequential computing. But due to the independency of parts at a certain level, the abstraction of each part can be done in high parallelism. With the help of parallel computation, the preprocessing can be finished in O(1) time. Hence, M2M-PF has great potential to be accelerated by multi-core CPU and GPU..

Dynamic structure

Compared with other hierarchical algorithms, M2M-PF enjoys better dynamicity. Because the preprocessing doesn't involve preconputation and doesn't store any path, the operations of M2M data structure such as adding or deleting an arc can be finished in O(1) time in most of the case. Hence, the hierarchical structure needn't be reconstructed when the map changes a little. Additionally, the preprocessing of M2M structure is a lightweight process which consumes less time than most of the pathfinding algorithms with precomputation.

Robustness and Fault-tolerance

If a little change is made on the map when an agent is moving, the path in micro level becomes invalid, but the topological graph in macro level is still valid in most cases. Since M2M-PF preserves search space at each level, the path at micro level can be adjusted quickly to adapt to the change based on the path at macro level. In contrast, the traditional A* algorithm has to recalculate and some other hierarchical algorithms have to reconstruct the hierarchical topological graph.

Preprocessing sharing

The M2M algorithms series (M2M Path Finding, M2M Convex Hull, M2M Nearest Neighbor and M2M Collision Detection) based on the M2M model share an identical data structure model. Supported by this structure, the M2M algorithms series can be accelerated greatly. Take M2M-CH as an example, it is about 100 times faster than traditional convex hull algorithms. Such characteristic can greatly reduce the total time cost in fields like image process, pattern recognition and multi-agent system.

Trade-off

Trade-off between accuracy and time cost can be made by setting different *range value*. Essentially, the *range value* decides the size of search space, which in turn determines the time cost and the probability of getting the optimal solution.

Extension to 3D

Since the M2M model is based on regular parts, it can be extended to 3D naturally by defining a part as a cube. In this way, M2M-PF can deal with 3D pathfinding problems.

Table 1. The comparison among pathfinding algorithms

	Dijkstra	HA*	HPA*	PRA*	M2M
Preprocessing sharing	No	Poor②	Poor②	Poor②	Good①
Parallelism	Poor	General③	Poor	Good ③	Good③
Dynamics	No④	General⑤	Poor	Good	Good
Multiple levels and various granularities	No	Yes	Yes	Yes	Yes
Step-wise refinement	No	No	Yes	Yes	Yes
Balance between accuracy and efficiency	No	Yes	Yes	Yes	Yes
Spreading time cost over execution time	No	No	Yes	Yes	Yes
Works on generic maps	Yes	No	Yes	No	Yes
Extend to 3D	Yes	Yes	No	Poor	Yes

① The M2M data structure can be shared by many algorithms
② Only available in pathfinding
③ Parallel preprocessing
④ Require recomputation after the changes of point sets
⑤ Require recomputation of the path between the entrance and the exit

12.Conclusions and Future Work

With the help of the M2M model, computer can solve problems like human beings who aim at a wider holistic view of problem solving. So, we can use this model to improve many algorithms, such as nearest neighbor search, convex hull, collision detection, and pathfinding. We can even use it to set up a powerful holistic strategy in RTS game. The pathfinding algorithm based on the M2M model can deal well with various map types including 3D map, and it has many desirable advantages compared with the classical pathfinding algorithms, such as preprocessing sharing, high parallelism, dynamics, etc.

The M2M model and the M2M pathfinding algorithm are full of vitality and prospect. There is much room for improvement. We are planning to introduce parallel computation technology to accelerate the preprocessing and develop new M2M algorithms to enrich the M2M algorithm series.

13.References

[1] Yao, Y.Y. 2007. The art of granular computing, Proceedings of International Conference on Rough Sets and Emerging Intelligent System Paradigms

[2] Botea, A., Müller, M., and Schaeffer, J. 2004. Near optimal hierarchical path-finding. J. of Game Develop.

[3] Sturtevant, N. and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. AAAI.

[4] Y.P.Zhang, Z.Z.Zhang, Q.Chen. 2007a, A New Nearest Neighbour Search Algorithm based on M2M Model.: IMECS.

[5] Y.P.Zhang, Z.Z.Zhang, Q.Chen, Z.M.Zhou, S.Z.Luo. 2007b, A New Randomized Parallel Dynamic Convex Hull Algorithm based on M2M model: International Conference on Convergence Technology and Information Convergence.

[6] Hart, P., Nilsson, N., and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. on Systems Science and Cybern. 4:100.107.

[7] Korf, R. 1985. Depth-first iterative-deepening: an optimal admissible tree search. Artif. Intelligence 27(1):97.109.

[8] Koenig, S., and Likhachev, M. 2002. Improved fast replanning for robot navigation in unknown terrain.

[9] Holte, R., Mkadmi, T., Zimmer, R. M. and MacDonald, A. J. 1996a. Speeding up problem solving by abstraction: A graph oriented approach. Artif. Intell. 85(1.2):321.361.

[10] Holte, R., Perez, M., Zimmer, R. and MacDonald, A. 1996b. Hierarchical A*: Search abstraction hierarchies eficiently. In AAAI/IAAI Vol. 1, 530.535.

[11] P. Sanders and D. Schultes. Fast and Exact Shortest Path Queries Using Highway Hierarchies. InProc. 13th Annual European Symposium Algorithms, 2005.

[12] A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A_ Search Meets Graph Theory. InProc. 16th ACM-SIAM Symposium on Discrete Algorithms, pages 156–165, 2005.