More Flexible Proximity Wildcards Path Planning with Compressed Path Databases

Xi Chen[1,3], **Yue Zhang**[1,3], **Yonggang Zhang**[*,2,3]
[1]College of Software, Jilin University, China
[2]College of Computer Science and Technology, Jilin University, China
[3]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, China
{xichen22, zhang_yue19}@mails.jlu.edu.cn, zhangyg@jlu.edu.cn

Abstract

Grid-based path planning is one of the classic problems in AI, and a popular topic in application areas such as computer games and robotics. Compressed Path Databases (CPDs) are recognized as a state-of-the-art method for grid-based path planning. It is able to find an optimal path extremely fast without state-space search. In recent years, researchers have tended to focus on improving CPDs by reducing CPD size or improving search performance. Among various methods, proximity wildcards are one of the most proven improvements in reducing the size of CPD. However, its proximity area is significantly restricted by complex terrain, which significantly affects the pathfinding efficiency and causes more additional costs. In this paper, we enhance CPDs from the perspective of improving search efficiency and reducing search costs. Our work focuses on using more flexible methods to obtain larger proximity areas, so that more heuristic information can be used to improve search performance. Experiments conducted on the Grid-Based Path Planning Competition (GPPC) benchmarks demonstrate that the two proposed methods can effectively improve search efficiency and reduce search costs by up to 3 orders of magnitude. Remarkably, our methods can further reduce the storage cost, and improve the compression capability of CPDs simultaneously.

Introduction

Path planning, which has been studied for many years, is one of the important problems in artificial intelligence and widely applied in real-world scenes such as robotics and computer games (Freund and Hoyer 1986; Cui and Shi 2011). Grid-based path planning is one of the most active research directions on this problem. Over the years, numerous excellent algorithms have been proposed, and the Grid-Based Path Planning Competition (GPPC) (Sturtevant et al. 2015) has served as the cornerstone for the development of these novel path-planning achievements. The impact of these advancements is essential and far-reaching, and they will revolutionize path planning in the coming years (Botea 2011; Uras and Koenig 2014; Harabor and Grastien 2014; Rabin and Sturtevant 2016; Sturtevant and Rabin 2016; Uras and Koenig 2017; Goldenberg et al. 2017; Cohen et al. 2017; Harabor and Stuckey 2018; Salvetti et al. 2018; Hu et al. 2019, 2021).

The Compressed Path Databases (CPDs) (Botea 2011; Botea and Harabor 2013) known as a group of state-of-the-art technique for grid-based path planning, are designed to speed up the response and reduce the first move delay during the agent's path planning tasks (Zhao 2022). Each CPD is a data structure that provides the optimal first move from any start cell s to any target cell t. The process of finding an optimal path is to iteratively search such a group of CPDs to find the optimal move directions to reach the target without any state space search. The main disadvantage of CPDs with ultra-fast pathfinding speed is the huge build cost. Each CPD needs to be encoded by all-pairs of precomputation, and then compressed with encodes and store the result. In recent years, researchers have mainly tend to focused on reducing the size of the CPD such as single row compression (SRC) (Strasser, Harabor, and Botea 2014), heuristic redundant symbols (Chiari et al. 2019), proximity wildcards (Chiari et al. 2019), bidirectional wildcards (Salvetti et al. 2017), or improving the lookup performance such as two-oracle path planning algorithm (Topping) (Salvetti et al. 2018). Among them, the SRC performed best in GPPC 2014, and has become an important baseline for improving CPDs.

The CPDs enhanced by proximity wildcards (CPDs_PW), replaces the storage symbols of qualified nodes in the largest square centered on any node with wildcards to reduce the preprocessing memory based on heuristic redundant symbols. Their experiments are mainly based on Dragon Age: Origins (DAO) (Sturtevant 2012), which verifies the excellent compression capability of CPDs_PW. It can achieve better on 99% of the maps, and can even compress the map size to 1/55 of the SRC on the map AR0044SR.

Despite the effectiveness of CPDs_PW for advanced compression, there are still some drawbacks:

1. Severely limited by complex terrain. Its area of concern is square, causing it to miss out on expansion opportunities and incur more search costs when it encounters obstacles in any direction.
2. Inefficient search. Compression is not efficient enough, resulting in more binary searches to find the target during pathfinding, leading to inefficient search.

*Corresponding author

In this paper, our idea for the above problems is to expand the proximity area by breaking its shape limitation without weakening the compression capability. We propose the following two methods:

1. Rectangular Proximity Wildcards (RPW): This method breaks the limitation of length and width, and expands the shape of proximity area to a rectangle so that when obstacles are encountered in either the length or width, the other can still expand.
2. Coordinates Proximity Wildcards (CPW): This method breaks the limitations of traditional geometry. To achieve more flexible obstacle avoidance and improve the expansion opportunities, four quadrants are divided with the current node as the center, and the largest rectangle is expanded in each quadrant. The areas drawn by the four quadrants are all proximity areas of the current node.

We expand the scope of the experiment and utilized Baldurs Gate II (BGII), DAO, Dragon Age II (DA), Warcraft, and Starcraft (Sturtevant 2012), a total of 5 benchmarks containing 454 maps, with SRC and CPDs_PW as the experimental baselines. The experimental results indicate that the two methods we propose can efficiently enhance search efficiency and reduce search costs by up to three orders of magnitude. In addition, they can also improve the compression capability of CPDs. Experiments demonstrate that the more flexible the method is in avoiding obstacles, the less it is affected by complex terrain, and the more efficient it is.

The paper is structured as follows: First, we provide an overview of related work. Then, we introduce the fundamental principles of the CPDs and the key technologies utilized. After that, we present a detailed description of the RPW and CPW, along with description diagrams and algorithm pseudo codes. We also present the experimental results and analysis, and conclude the paper at the end.

Related Work

As a leading technique for optimal pathfinding, CPDs (Botea 2011) have received extensive attention since it was proposed. Its main idea is to achieve speedup through preprocessing and additional memory, which has shown excellent performance in solving speed, but its huge memory cost seems to be a huge burden.

Therefore, exploring better compression methods for CPDs have become a research hotspot. The Copa (Botea and Harabor 2013), which combines list trimming, run length encoding, and sliding window compression to improve compression capabilities at the cost of time loss, is one of the best competitors in GPPC 2012. In 2014, SRC (Strasser, Harabor, and Botea 2014), an algorithm that combines with run length encoding (RLE) to compress rows, outperforms Copa in both compression and query time, and is one of the winners in GPPC 2014. Although SRC has excellent performance in compression capability and speed, it still requires huge memory overhead for large maps. To solve this practical bottleneck, wildcard substitutions (Salvetti et al. 2017) are introduced, which use wildcards to replace part of the CPD-encoded data, and can be combined with heuristics to reduce the size of the CPD. Its main idea is that given any start node s and target t, as long as either the route from s to t or from t to s is feasible, a complete optimal path can be established. This method has been shown to be effective, and the proximity wildcards (Chiari et al. 2019) are an extension of it, with a new concept of redundant symbols simultaneously proposed. The proximity wildcards uses redundant symbols to define a square centered on the current node, which is called the proximity area, and all nodes in it can be optimally reached by the heuristic move. Experiments verified that proximity wildcards are one of the most effective methods to improve the compression capability of CPDs. Still it is seriously affected by obstacles and therefore runs with limited efficiency. Our work is related to proximity wildcards, but focuses on improving search performance and using more flexible methods to reduce the impact of obstacles on efficiency.

The above works are all approaches to reduce the memory overhead of CPDs in finding optimal solutions, often accompanied by a significant loss of preprocessing time. In 2020, a centroid-based bounded suboptimal method (Zhao et al. 2020) reduces the storage cost by computing only the first-move array of selected nodes, which ensures that the path costs are within the fixed bound of the optimal solution. This approach innovatively reduces preprocessing time and storage cost by trading some suboptimality.

CPDs are also utilized to solve the Euclidean Shortest Path Problem (ESPP) (Shen et al. 2020). In 2020, Shen et al. proposed End Point Search (EPS). EPS is the state-of-art ESPP algorithm and uses CPD to extract the shortest path from the opposite end vertex $v^{'} \in V_{opp}$ to the current vertex v and caches the shortest distance from each vertex v_x to v on the path.

Some works focus on improving search performance, such as Topping (Salvetti et al. 2018), which is a combination of SRC and Jump Point Search+ (JPS+) (Harabor and Grastien 2014). It first calls SRC to find the best direction for moving from the current node to the target and then calls JPS+ to evaluate the feasible steps in that direction. In most cases, Topping can be more than an order of magnitude faster than SRC, but at the cost of nearly doubling memory consumption. To deal with the huge memory overhead, TOPS (Hu et al. 2021) and Topping+ (Hu et al. 2021) are proposed. The TOPS reduces the preprocessing memory by calculating only the CPD of jump points. The Topping+ extracts a series of complete paths from the successor start nodes to the target. TOPS and Topping are competitive with Topping while having smaller space requirements and better time performance.

Background

Gridmap. A gridmap is a two-dimensional operating environment for mobile agent, where each cell is either traversable (white squares) or obstacle (black squares). It has two common ways of defining neighbor: 4-connected grids and 8-connected grids. In 4-connected grids, a traversable cell has a maximum of four neighbors, and the directions of movements are: East, South, West and North (symbolized as E, S, W, N), with a move cost of 1 per

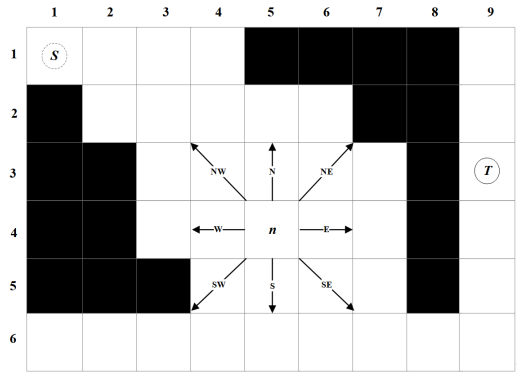

Figure 1: 8-Connected grid map.

N	N	N		N	N,NE	N,NE	N,NE	N,NE
N,W	N	N	N	N	E,NE		E,NE	E,NE
W				N	E,NE		E,NE, SE	E,SE
W	W	W	W	*s*	E	E		E,SE
W				S	SE	E,SE	E,SE	E,SE

Figure 2: Example of the first move array for node s.

cell. As shown in Figure 1, for any node n on the map, the 8-connected grid map allows diagonal movement, adding movable directions: Northwest, Northeast, Southwest, and Southeast (symbolized as NW, NE, SW, SE), the linear move cost $\vec{c}$ is 1, and the diagonal move cost $\vec{d}$ is $\sqrt{2}$. A path planning problem on a gridmap starts at cell s called the start node and ends with cell t called the target node. The goal is to find the most efficient path with the minimum cost from s to t. The formula $s' = s + k\vec{d} + m\vec{c}$ (k and m are integers) means that moving from node s to node s' needs k times along the direction $\vec{d}$ and m times along the direction $\vec{c}$. If k and m are negative integers, it means reverse movement. In this paper, we assume that diagonal movements are not allowed if they will touch an obstacle cell.

Compressed Path Databases (CPDs). CPD is a data structure that encodes the optimal first move from any node s to a target node t on a map. It is the result of compressing the first move array using run length encoding (RLE) (Strasser, Harabor, and Botea 2014). The first move array $T(s)$ of node s stores the first move of the shortest path from node s to each reachable node t, computed by Dijkstra. CPDs are constructed by computing a first move array for each node in the map in offline preprocessing. Chiari et al. redefined CPD as consisting of the first move array (the original meaning of CPD) and auxiliary search data due to the specificity of their methods (such as proximity wildcards). We use this definition, i.e. the CPD size is the sum of the first move array size and the auxiliary data size. The size of the first move array represents the method's compression capability. Each binary search extracts the next move in the first move array of the current node based on the target position.

Run Length Encoding (RLE). RLE compresses the first move array by more compactly representing the substrings (called runs) made up of repetitions of the same symbol. It marks obstacles and source node with the wildcard "*", because they don't need to be searched. Taking Figure 2 as an example, the first row can be compressed into NNN*NNNNN. Usually we use a substring to represent the whole map, so the first row is represented as 1N, and this map is represented as 1N; 15E; 19W; 23N; 24E; 28W; 33E; 37W; 41S; 42SE. Where the letters denote optimal first moves to move the source node s to the corresponding position and the numbers are subscripts starting from 1 after reducing the 2D first-move array to a 1D array.

Heuristic move. The heuristic move refers to the first move of node s to ignore obstacles and move towards the goal. It is determined by the shortest heuristic distance computed by the heuristic function $F_e(n,t)$, where $\omega(s,n)$ is the cost from source s to node n and $f_e(n,t)$ is the heuristic distance function from n to t. E is a set of feasible edges, where $(s,n) \in E$ means that the path from s to t is feasible. In this paper, we use the Euclidean distance to compute the heuristic move. We also use the redundant symbol h to mark the nodes where the heuristic move coincides with the first move to improve efficiency. That is, if the $F_e(n,t)$ returns a first move that belongs to the first move array $T(s)$, then add the redundant symbol h to $T(s)$.

$$f_e(s,t) = \sqrt{(s.x - t.x)^2 + (s.y - t.y)^2} \quad (1)$$

$$F_e(s,t) = \arg\min_{(s,n)\in E}\{\omega(s,n) + f_e(n,t)\} \quad (2)$$

Proximity Wildcards. Proximity wildcards compute the largest square proximity area centered on s, where traversable nodes must have the heuristic redundant symbol h. If the target node t is within the proximity area of s, the heuristic move can be applied directly, otherwise search for the first move.

Rectangular Proximity Wildcards

According to proximity wildcards, nodes within the proximity area of source node s can be optimally reached via heuristic move. However, the proximity area of proximity wildcards is in a square shape, meaning that if the neighbor in any direction of s is an obstacle, the proximity area will stop expanding. This scenario is common in complex terrain, thus making it less effective in such terrain. In this section, we introduce a method called Rectangular Proximity Wildcards (RPW), whose region of interest is rectangular. The main advantage of RPW is that if the length or width of the proximity area stops expanding, expansion can still continue in the other direction, as shown in Figure 3.

Definition 1: Given a node s and a function $F_x(s,t)$, the width of largest proximity rectangle R centered on s is expressed as $rec(s).x$ and length is expressed as $rec(s).y$, for

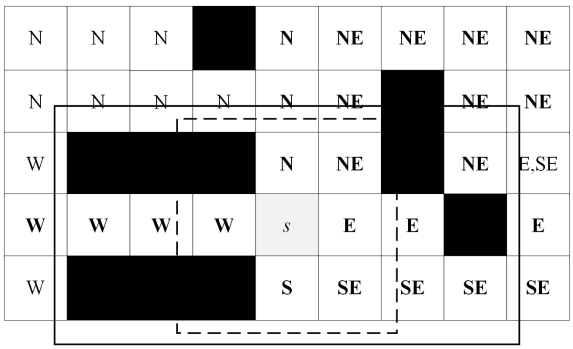

Figure 3: Proximity areas. RPW's line is solid while proximity wildcards 's line is dashed. Source node is shown in gray. Heuristic moves coinciding with the first moves are shown in bold.

Algorithm 1: CPDHRP(s, t)

Input: start node s, target node t
Output: first-move m

```
1: X ← rec(s).x.
2: Y ← rec(s).y.
3: if |s.x − t.x| ≤ X/2 ∧ |s.y − t.y| ≤ Y/2 then
4:     return F_x(s, t)
5: else
6:     m ← CPD(s, t)
7:     if m = h then
8:         return F_x(s, t)
9:     else
10:        return m
11:    end if
12: end if
```

any node $n \in R$, there is $f_x(s, n) \in T(n)$, T is the first-move array of s, and $rec(s).x * rec(s).y$ is the maximum value $d \in N$.

The cells in the proximity rectangle are all optimally reachable from s by heuristic moves, and we denote each such cell with the wildcard character "*". As shown in Figure 3, the compression result of the first move array with the proximity wildcards is 1N; 5h; 10N; 14h; 19W; 26h; 27E; 28h; 37W; 43h, a total of 10 RLE runs and the compression result using RPW is 1N; 5h; 10N; 14h; 19W; 27E; 28h; 37W; 45h, a total of 9 RLE runs.

RPW has a larger proximity area than that of proximity wildcards, which typically leads to more efficient compression and more readable encoding due to the need for fewer RLE runs. This provides a solid foundation for improving search performance. In the online search, RPW is able to flexibly respond to complex terrain changes, and its larger proximity area can help to apply more heuristic moves, effectively improving search efficiency and reducing cost. The CPD lookup function modified using RPW is shown in Algorithm 1.

Coordinates Proximity Wildcards

RPW has effectively improved the search efficiency of CPDs, but the use of traditional geometry as a proximity area still has significant limitations. When faced with extremely complex and narrow maps (such as bridge maps), in more cases, the area of RPW would be very restricted, even as large as proximity wildcards. In this section, we propose a more flexible method called Coordinates Proximity Wildcards (CPW), as shown in Figure 4.

Definition 2: Given a node s and a function $F_x(s, n)$, the CPW area C consists of four largest rectangles R_m with s as the corner. The length and width of each quadrant rectangle R_m are expressed as $rec(s)_m.x$ and $rec(s)_m.y$, $m \in [1, 4]$. For any node $n \in C$, $f_x(s, n) \in T(n)$, T is the first-move array of s.

We replace the traditional geometry centered on the source node with quadrants centered on the source node, and expand the largest rectangle in each quadrant. As shown in Figure 5, the compression result of the first move array using PW is 1NW; 2h; 7SW; 8h, with a total of 4 RLE runs. The compression result using RPW is 1NW; 3h; 7SW; 8h, a total of 4 RLE runs. The compression result using CPW is 1NW; 7SW, a total of 2 RLE runs. From a search perspective, the proximity area of PW is 1, i.e., its proximity area contains only source node d. The proximity area of RPW is 3, which means that two nodes can be reached directly by the heuristic move without calling the CPD lookup function. The proximity area of CPW is 6 times than that of PW. A more flexible expansion method can make the proximity area larger, which effectively reduces CPD calls and helps to improve search efficiency. However, the size of the auxiliary data required by CPW is four times that of PW and RPW, because each of its proximity areas needs to store the side lengths of the four largest rectangles.

The CPW proximity area determination function GetCPW(s, t) is shown in Algorithm 2. GetCPW(s, t) first delimits the coordinate proximity area with the current node s as the origin, then checks which quadrant the target node t is located in, and whether t is within the proximity area of s. The CPD lookup function improved with CPW is based on Algorithm 1 and replaces the third line with GetCPW(s, t).

Algorithm 2: GetCPW(s, t)

Input: start node s, target node t
Output: $true$ or $false$

```
1: for n = 1 to 4 do
2:     X.n ← rec(s)_n.x.
3:     Y.n ← rec(s)_n.y.
4: end for
5: Judge t in the nth quadrant of s
6: if |s.x − t.x| ≤ X.n ∧ |s.y − t.y| ≤ Y.n then
7:     return true.
8: end if
9: return false.
```

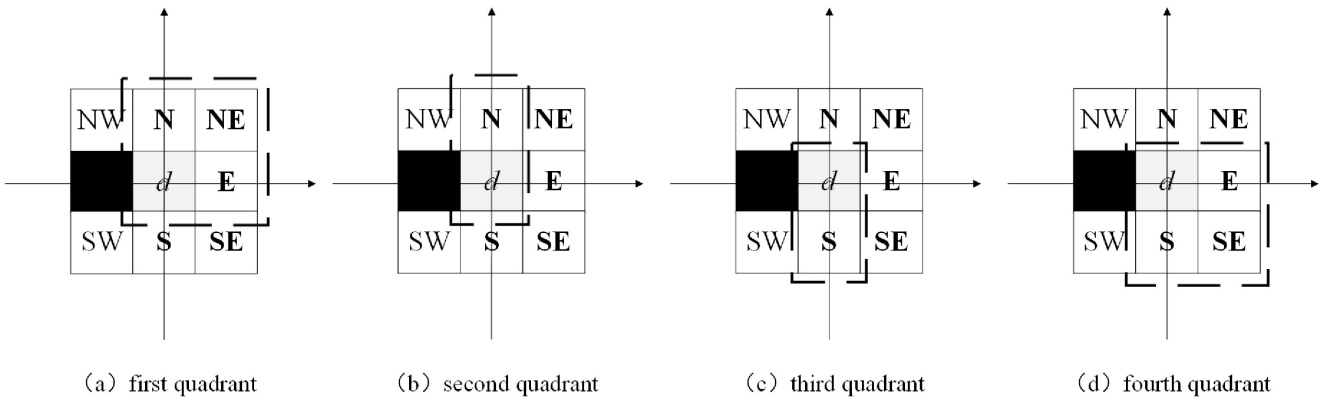

Figure 4: Proximity areas of CPW

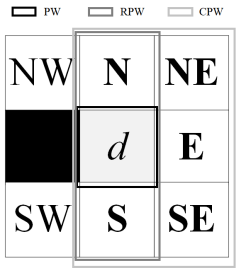

Figure 5: Comparison of proximity areas. For the sake of illustration, we use PW here to represent the proximity wildcards method.

Experiments

We conducted experiments on five game benchmarks from GPPC (Sturtevant et al. 2015): BGII, DAO, DA, Warcraft and Starcraft. BGII is a small map set with 120 maps. Each map has a varying number of nodes ranging from 100 to 60,000. The DAO and DA are medium-sized map sets with 156 and 67 maps respectively, with nodes in each map distributed between 100 and 140,000. Warcraft contains 36 maps, while Starcraft contains 75 maps. Both of them are large map benchmarks with each map having 50,000 to 760,000 nodes. We use SRC and CPDs_PW as experimental baselines. SRC is the original algorithm without the methods we proposed, and it is primarily used to verify the effectiveness of the methods. The CPDs_PW is an essential reference. We have applied our methods to SRC and the results are as follows:

- CPDs_RPW. Algorithm improved by rectangular proximity wildcards (RPW).
- CPDs_CPW. Algorithm enhanced by coordinates proximity wildcards (CPW).

There are two stages in experiments: offline preprocessing and online search. In the preprocessing stage, we primarily observe the compression cost and compression capability through RLE runs and the first move array size after a compression task is completed. We also measure memory storage by CPD size. It is important to mention that the CPD for CPDs_PW, CPDs_RPW, and CPDs_CPW are all composed of the first move array and the auxiliary data, while the CPD size of SRC is equal to the first move array.

The online search is the stage we focus on most. In the online search stage, we measure search efficiency and search cost through runtime and binary searches. Due to the large different in map sizes in the same benchmarks, the data distribution varies greatly. Therefore, we use factor C_{metric_i} to describe the metrics we observed above and define it as Equation 3.

$$C_{metric_i} = \frac{realnum_{SRC}(metric_i)}{realnum_x(metric_i)} \tag{3}$$

The factor C_{metric_i} is the result of dividing the SRC running results for the same map by the running results of each CPD variant. $metric_i$ represents the i-th metric, and $realnum_x(metric_i)$ denotes the real value of the i-th metric of CPD variant x. x can be assigned to CPDs_PW, CPDs_RPW, and CPDs_CPW. We define the factors of each metric as follows:

- $C_{RLE-runs}$. Compression cost factor.
- $C_{first-move}$. Compression capability factor.
- C_{CPD}. Memory factor.
- $C_{binary-search}$. Binary search factor.

All algorithms are implemented in C++, and the experimental environment is Ubuntu 20.04.3 LTS, with processor AMD® Ryzen 9 5900*12core processer*24 and 31.4GiB RAM.

Preprocessing

Compared to CPDs_PW, our methods show evident improvements on all maps in the preprocessing stage. Figure 6 shows the distribution of the compression cost factor $C_{RLE-runs}$. The $C_{RLE-runs}$ of our methods are always higher than CPDs_PW, with CPDs_CPW performing best. Taking the map hrt000d as an example, the RLE runs of CPDs_PW on this map is $2.34e^6$, but CPDs_RPW is $1.18e^4$ less than it, and RLE runs of CPDs_CPW is only $2.04e^6$. Our

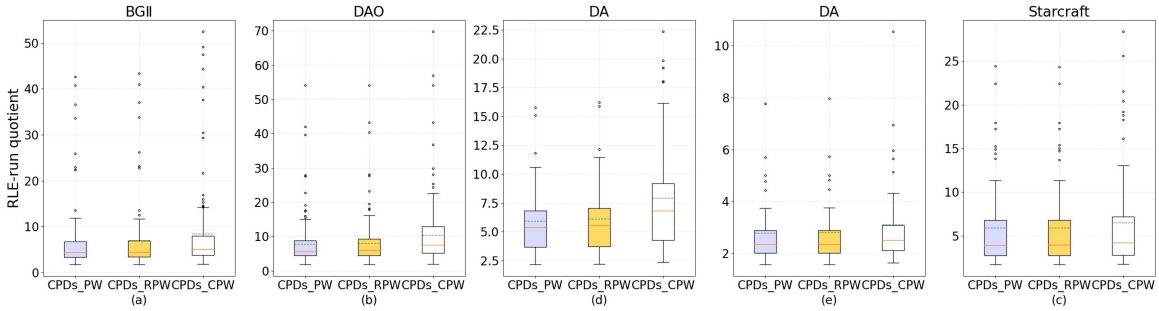

Figure 6: Compression cost factor $C_{RLE-runs}$. The compression cost factor is $C_{RLE-runs}$ for completing a compression task.

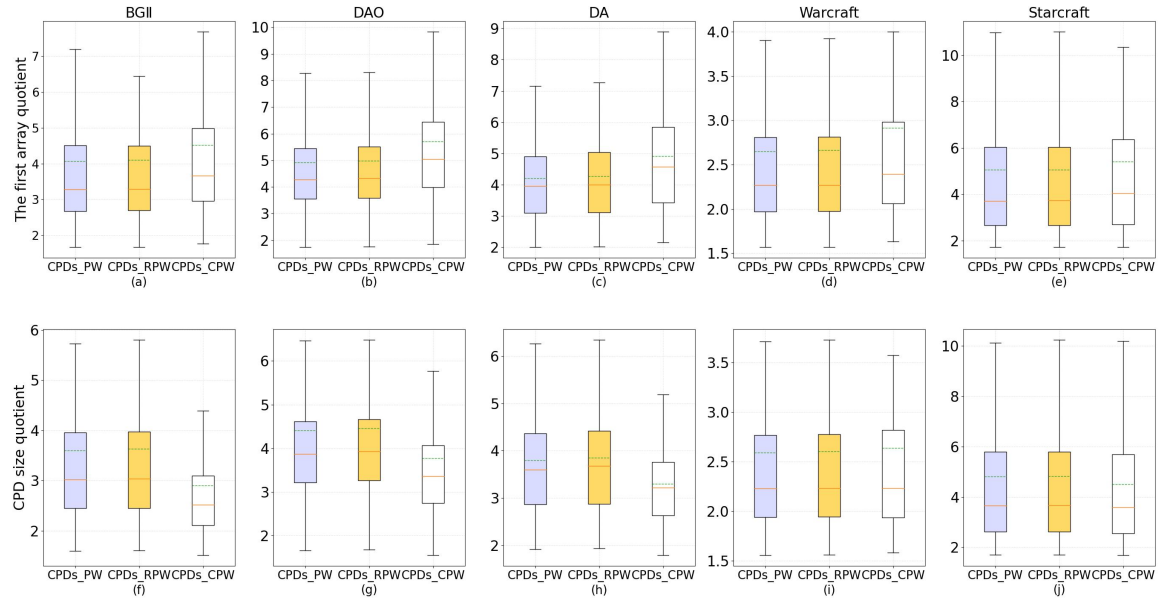

Figure 7: Compression capability factor $C_{first-move}$ and Memory factor C_{CPD}. (a) to (e) show the compression capability factor $C_{first-move}$, indicating the compression capability of methods. (f) to (j) show the memory factors C_{CPD}, indicating the memory occupied by the CPD size, and the CPD size is the sum of the first move array and auxiliary data sizes.

methods can perform the compression task more efficiently and produce more concise and readable compressed results by reducing RLE runs. This is beneficial for cost-saving and search purposes.

Figure 7 (a) to (e) show the distribution of compression capability factor $C_{first-move}$. The $C_{first-move}$ of our methods is significantly higher, resulting in smaller first-move arrays. CPDs_CPW has the highest $C_{first-move}$, which means that it has the best compression capability among the three variants. The memory factor C_{CPD} is shown in Figure 7 (f) to (j). We can observe that CPDs_RPW's C_{CPD} is typically slightly higher than CPDs_PW, but CPDs_CPW only performs better in Warcraft, with noticeable drawbacks in other maps. This is due to the fact that the size of CPDs_CPW's auxiliary data (proximity distance) is equal to four times of CPDs_PW and CPDs_RPW, resulting in larger CPD size and higher memory cost. Therefore, we may sometimes encounter scenarios where the memory gain is less than the expenditure, and this situation will be more common in small maps.

Online Search

The first advantage of our methods during the search stage is more efficient search, as shown in Figure 8. CPDs_RPW and CPDs_CPW are much faster than CPDs_PW, because of the more concise first move array generated by the preprocessing stage and larger proximity areas that effectively speed up the search. It can be observed that in some maps, the CPDs_CPW, which has larger proximity areas, takes more time to search than CPDs_RPW. This is because the complexity of the CPD query function directly affects the time taken by each call, and the CPDs_CPW has the most complex query function, followed by that of the CPDs_RPW.

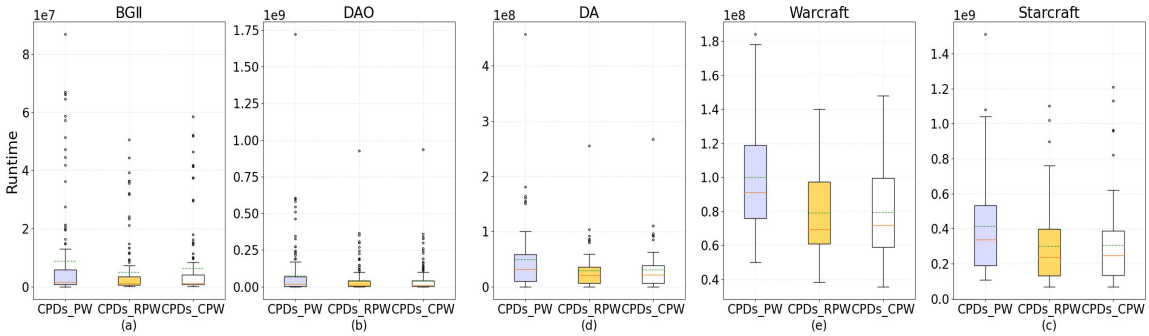

Figure 8: Runtime (unit: ns). Runtime is the time taken to complete a path extraction, and we use it to measure search efficiency

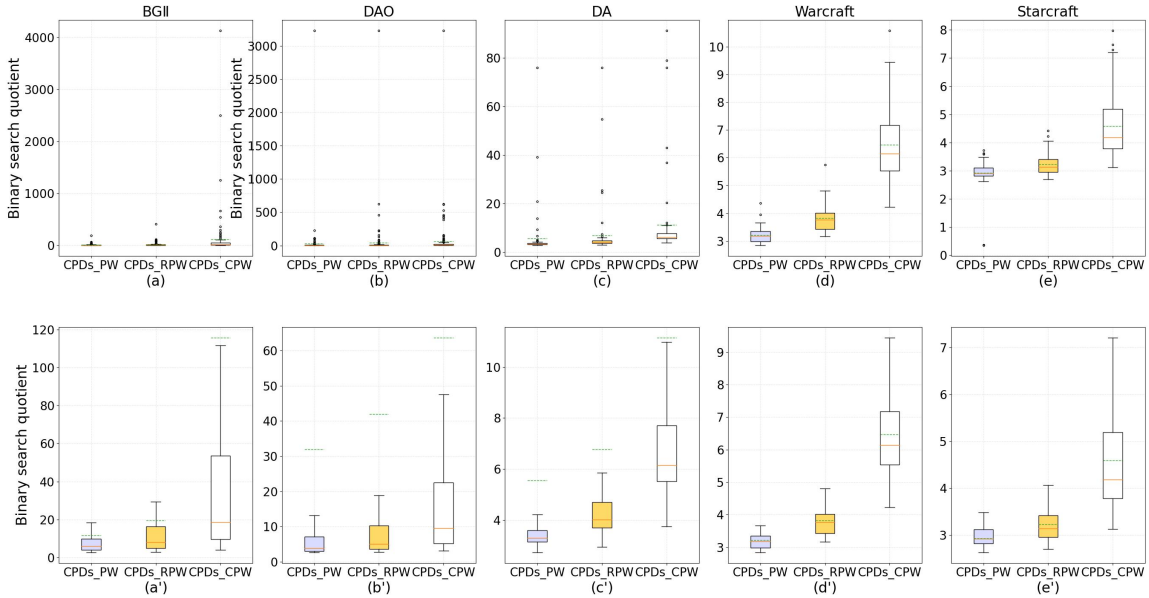

Figure 9: Binary search factor $C_{binary-search}$. (x') is a version of (x) that hides outliers. The binary search factor $C_{binary-search}$ is the quotient of binary searches required by SRC and CPDs_X to complete a search task.

Map	#Cell	The number of binary searches			
		SRC	PW	RPW	CPW
AR0041SR	2282	8258	131	76	2
AR0418SR	1428	2501	13	6	0
AR0408SR	354	1092	25	13	3
orz105d	679	2642	36	12	5
orz107d	637	1791	54	22	4
lgr605d	2983	8169	83	36	21

Table 1: Examples of exceptional binary searches. #Cell is the number of nodes contained in the map.

The second advantage of CPDs_RPW and CPDs_CPW in online search is the lower search cost. Figure 9 shows a significant difference in binary searches required by the three variants to complete search tasks. Of the three, CPDs_CPW has the highest $C_{binary-search}$, indicating that it requires the least number of binary searches. In both BGII and DAO, we obtained some results where $C_{binary-search}$ reached hundreds or even thousands of orders of magnitude. These surprisingly improved effects are often observed in small maps, some of which we list in Table 1. We find that the distribution of $C_{binary-search}$ becomes more stable and concentrated as the map size increases. For example, the interquartile range of CPDs_CPW would be 37 (52-15), 16 (22-6) in DAO, and just under 2 in Warcraft and Starcraft.

We also discovered that methods with larger proximity areas are more effective in search and compression. According to the above experiments, the proximity area primarily enhances methods in the following two aspects:

1. More adequate heuristics. A larger proximity area can contain more heuristic information, increasing the chances of discovering the target.
2. More concise search space. In the preprocessing stage,

Set	Number of wins			
	SRC	PW	RPW	CPW
Starcraft (75)	1	15	19	**28**
DAO (156)	1	53	60	**106**
BGII (120)	0	92	102	**115**

Table 2: Comparative results with Topping on the number of binary searches. The number in brackets is the map number.

a larger proximity area helps to compress the first move array into a more concise substring, increasing the speed of finding the first move.

Our methods vs Topping

In this section, we conduct a search cost comparison experiment with Topping, a representative algorithm in another research area. It significantly improves search performance at the expense of huge storage costs and inefficient compression.

We select three representative benchmarks to experiment with and present the results in Table 2. Experiments confirm the effectiveness of our methods, and CPDs_CPW is even competitive with Topping in terms of search costs. Topping's pathfinding is performed jointly by the SRC oracle and the JPS+ oracle. We only compare the number of binary searches performed by the SRC oracle in Topping. Experimental results show that CPDs_CPW performs best among the variants and has an absolute advantage in binary searches on 68% of medium maps and 96% of small maps, and occupies a place on large maps. Note that our methods use only half the memory size of Topping.

Conclusion and Future Work

As the leading technology for path planning, CPDs have the bottleneck of huge storage overhead. A method called proximity wildcards significantly improves the compression capability of CPDs with surprising results in reducing storage costs. However, it is severely limited by complex terrain, making the search inefficient and and generating more costs.

In this paper, we extend proximity wildcards and propose two methods, RPW and CPW, which can be more flexible in avoiding obstacles to compute larger proximity areas in response to complex terrain changes. Meanwhile, we have extended the experiment scale to five benchmarks. The experimental results demonstrate that rectangular proximity wildcards (RPW) and coordinates proximity wildcards (CPW) can flexibly deal with complex terrain and significantly improve search performance while gaining compression benefits.

Our future work mainly focuses on the following directions:

1. Exploring ways to reduce time while maintaining optimality, compression and search capability. Investigating new encoding alternatives to RLE is an interesting research direction.
2. Exploring a combination algorithm compatible with CPDs that has powerful search performance without causing internal interaction storage costs.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (Grant No. 62076108, 61872159) and Key R&D projects of Jilin Provincial Science and Technology Development Plan (Grant No. 20240207002JH).

References

Botea, A. 2011. Ultra-fast optimal pathfinding without runtime search. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 7, 122–127.

Botea, A.; and Harabor, D. 2013. Path planning with compressed all-pairs shortest paths data. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, 293–297.

Chiari, M.; Zhao, S.; Botea, A.; Gerevini, A. E.; Harabor, D.; Saetti, A.; Salvetti, M.; and Stuckey, P. J. 2019. Cutting the size of compressed path databases with wildcards and redundant symbols. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 106–113.

Cohen, L.; Uras, T.; Jahangiri, S.; Arunasalam, A.; Koenig, S.; and Kumar, T. 2017. The FastMap algorithm for shortest path computations. *arXiv preprint arXiv:1706.02792*.

Cui, X.; and Shi, H. 2011. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1): 125–130.

Freund, E.; and Hoyer, H. 1986. Pathfinding in multi-robot systems: Solution and applications. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, 103–111.

Goldenberg, M.; Felner, A.; Palombo, A.; Sturtevant, N.; and Schaeffer, J. 2017. The compressed differential heuristic. *AI Communications*, 30(6): 393–418.

Harabor, D.; and Grastien, A. 2014. Improving jump point search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 128–135.

Harabor, D.; and Stuckey, P. 2018. Forward search in contraction hierarchies. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, 55–62.

Hu, Y.; Harabor, D.; Qin, L.; and Yin, Q. 2021. Regarding goal bounding and jump point search. *Journal of Artificial Intelligence Research*, 70: 631–681.

Hu, Y.; Harabor, D.; Qin, L.; Yin, Q.; and Hu, C. 2019. Improving the combination of JPS and geometric containers. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 209–213.

Rabin, S.; and Sturtevant, N. 2016. Combining bounding boxes and JPS to prune grid pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 746–752.

Salvetti, M.; Botea, A.; Gerevini, A.; Harabor, D.; and Saetti, A. 2018. Two-oracle optimal path planning on grid maps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 227–231.

Salvetti, M.; Botea, A.; Saetti, A.; and Gerevini, A. E. 2017. Compressed path databases with ordered wildcard substitutions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, 250–258.

Shen, B.; Cheema, M. A.; Harabor, D. D.; and Stuckey, P. J. 2020. Euclidean pathfinding with compressed path databases. In *International Joint Conference on Artificial Intelligence-Pacific Rim International Conference on Artificial Intelligence 2020*, 4229–4235. Association for the Advancement of Artificial Intelligence (AAAI).

Strasser, B.; Harabor, D.; and Botea, A. 2014. Fast first-move queries through run-length encoding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 5, 157–165.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2): 144 – 148.

Sturtevant, N.; Traish, J.; Tulip, J.; Uras, T.; Koenig, S.; Strasser, B.; Botea, A.; Harabor, D.; and Rabin, S. 2015. The grid-based path planning competition: 2014 entries and results. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, 241–250.

Sturtevant, N. R.; and Rabin, S. 2016. Canonical Orderings on Grids. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 683–689.

Uras, T.; and Koenig, S. 2014. Identifying hierarchies for fast optimal search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 878–884.

Uras, T.; and Koenig, S. 2017. Feasibility study: Subgoal graphs on state lattices. In *Proceedings of the International Symposium on Combinatorial Search*, volume 8, 100–108.

Zhao, S. 2022. *Improving Pruning and Compression Techniques in Path Planning*. Ph.D. thesis, Monash University.

Zhao, S.; Chiari, M.; Botea, A.; Gerevini, A. E.; Harabor, D.; Saetti, A.; and Stuckey, P. J. 2020. Bounded suboptimal path planning with compressed path databases. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 333–341.