CPDs path planning algorithms combined with improved proximity wildcards strategys

Yue ZHANG[1,3] , **Yong-Gang ZHANG**[2,3]
[1]College of Software, Jilin University
[2]College of Computer Science and Technology, Jilin University
[3]Key Laboratory of Symbolic Computing and Knowledge Engineering of Ministry of Education, Jilin University
1436388626@qq.com, zhangyg@jlu.edu.cn

Abstract

Path planning on gridmap is a hot issue in the field of artificial intelligence. As an important gridmap path planning algorithm, CPDs (Compressed Path Databases) are widely used in video games, robotics and other fields. The newly proposed proximity wildcards strategy can significantly compress the preprocessing memory of CPDs algorithm and improve the search efficiency, so it is recognized as one of the best memory compression strategies in the field. The paper proposes two improved strategies for expanding the proximity wildcards area to make use of more heuristic information, including RPW(Rectangular Proximity Wildcards) and CPW(Coordinates Proximity Wildcards), and two corresponding algorithms. The experimental results on grid-based path planning competition(GPPC) benchmark data set show that CPDs_RPW algorithm is superior to CPDs algorithm combined with the proximity wildcards strategy in all indexes during the preprocessing and search stages. Though CPDs_CPW algorithm performs slightly worse in CPD size, it performs best in all algorithms in the first-move array size.

1 Introduction

Path planning is an important research problem of artificial intelligence. It has been widely studied and has been widely used in real scenes such as robots and computer games [Freund and Hoyer, 2003; Xiao and Hao, 2011]. Among them, the path planning problem based on gridmap environment modeling is an active research field. Around the grid-based path planning competition[Sturtevant *et al.*, 2015], many excellent algorithms have emerged, which has an important and far-reaching impact on the research of path planning algorithms[Uras and Koenig, 2014; Uras and Koenig, 2017; Rabin and Sturtevant, 2016; Sturtevant and Rabin, 2016; Hu *et al.*, 2019; Harabor and Stuckey, 2018; Cohen *et al.*, 2018; Salvetti *et al.*, 2018a].

As one of the follow-up work of A* algorithm, CPDs algorithm has attracted extensive attention since it was proposed[Salvetti *et al.*, 2018a]. Its main idea is to preprocess the environment map and adopt the idea of space for time to significantly improve the search efficiency. CPDs algorithm is divided into two steps: offline preprocessing and online search. In the offline preprocessing step, each data structure called CPD provides the best first move from any cell *s* to any cell *t* of gridmap. Then, in the online search step, iteratively find the best first move to reach the target from the starting node until reaching the target, so as to quickly calculate the shortest path without any state space search. CPDs algorithm can also be used to quickly provide any prefix of the shortest path, which is very important to reduce the movement delay of agents. For example, the search based A* algorithm only knows the first best move after knowing the complete shortest path.

The main disadvantage of CPDs algorithm is that when the map is complex, the preprocessing memory consumption is large. Recently, many improvements have focused on reducing the preprocessing memory, that is, improving the compression efficiency of CPD, mainly including heuristic redundant symbols[Chiari *et al.*, 2019], proximity wildcards[Chiari *et al.*, 2019], bidirectional wildcards[Salvetti *et al.*, 2017], etc. This paper deals with this problem by improving the proximity wildcards strategy. On the basis of heuristic redundant symbols, proximity wildcards replaces the storage symbols of qualified nodes in the square area centered on any node with wildcards to reduce the preprocessing memory. This paper further reduces the preprocessing memory by expanding the area of proximity wildcards, and proposes two improved proximity wildcards strategies:

1. RPW (Rectangular Proximity Wildcards) strategy: expand the proximity wildcards area from a square centered on any node to a rectangle.
2. CPW (Coordinates Proximity Wildcards) strategy: take any node as the origin, divide the CPD structure into four quadrants, and find the largest rectangular area that meets the conditions in each quadrant.

Based on these two strategies, we propose two path planning algorithms: CPDs_RPW and CPDs_CPW.

The experimental map in this paper is from the GPPC benchmark data set. The experimental results show that CPDs_RPW algorithm is better than CPDs_PW algorithm(CPDs algorithm combined with the proximity wildcards strategy) in CPD size and search time. CPDs_CPW al-

gorithm is the best among all the comparison algorithms in terms of the size of the first-move array. However, due to the slightly complex query function, CPD size and search time are slightly inferior.

2 Related Work

As one of the leading algorithms in the field of path planning based on gridmap, CPDs algorithm has attracted extensive attention in academic circles. In recent years, the main content of the improvement of CPDs algorithm is to improve the compression efficiency of CPD.

The Copa algorithm proposed by botea et al. significantly improves the compression capacity of the algorithm by combining rectangular list pruning, default movement, run length coding and sliding window compression technology[Botea and Harabor, 2013]. The SRC(Single Row Compression) algorithm proposed by Strasser et al. further improves the compression performance of the algorithm by reordering the rows and columns of the first-move array and applying run length coding to each row[Chiari *et al.*, 2019]. Salvetti et al. proposed the concept of wildcard. The main idea is: given any two nodes s and t, the standard CPDs algorithm encodes the best first move from s to t, and also encodes the best first move from t to s. the CPDs algorithm based on wildcard encodes only one direction between s and t, so as to improve CPD compression[Salvetti *et al.*, 2017]. Chiari et al. proposed heuristic redundant symbols and proximity wildcards to compress the size of CPD[Chiari *et al.*, 2019]. Our work is based on this, and the details will be introduced later. Zhao et al. proposed a bounded suboptimal CPDs algorithm based on centroid, which only calculates the first mobile data of the selected node (centroid)[Zhao *et al.*, 2020], so as to reduce the storage cost. Among them, the selection of centroid ensures that the path cost is within the fixed range of the optimal solution.

In addition, Salvetti et al. proposed the Topping (Two Oracle Path Planning) algorithm in combination with SRC and another well-known path planning algorithm JPS+ (Joint Point Search+)[Salvetti *et al.*, 2018b]. The main idea is: given a current node s and a target node t, first call the SRC algorithm to obtain the best moving direction from s to t, and then call the JPS algorithm to calculate how many steps can be repeated in this direction, without the above query between these steps. In most cases, Topping algorithm can improve the performance of SRC algorithm by more than one order of magnitude. Hu et al. proposed Tops and Topping+ algorithms[Hu *et al.*, 2021]. Compared with Topping, Tops algorithm calculates the CPD data of all nodes. It only calculates the CPD data of jump points and reduces the preprocessing memory. Topping+ algorithm extracts a series of complete paths from the successor node of each start node to the target node for the motivation that the number of hops from the successor node of each start node to the target node may be very small relative to the number of single grid steps. Among all the extracted paths, the path with the lowest total cost is selected and returned as the optimal solution. In terms of online query speed, Tops and Topping+ are competitive compared with Topping algorithm.

Algorithm 1 CPDs(s, t)

Input: start node s, target node t
Output: shortest path p[]

```
p[]
while s ≠ t do
    (s, n) ← CPD(s, t)
    p ← p + [(s, n)]
    s ← n
end while
return p[]
```

3 Background

We introduce the background information of compressed path database, heuristic redundant symbols and proximity wildcards, which is necessary to understand the new contribution of this paper.

CPDs. CPDs is a path planning algorithm based on preprocessing acceleration technology. It uses pre calculated all pairs shortest paths (APSP) data to quickly find the shortest path without any state space search[Wu *et al.*, 2012]. CPDs algorithm calculates APSP data through a data structure called CPD. Building CPD requires a series of iterations. Each iteration uses different nodes s as the source (search root) and runs Dijkstra algorithm. The Dijkstra algorithm can be slightly modified to generate the first-move array *T(s)*. In *T(s)*, all nodes t reachable from s are assigned a label called the first-move, which identifies the initial movement of all shortest paths from s to t, that is, the first-move. Finally, compress the first-move array *T(s)* to end the current iteration. Since each iteration is independent, the time efficiency can be improved by parallel computing technology. The gridmap shown in Figure 1 demonstrates the preprocessing process of building a CPD from the source node s. For example, there are two best first moves from s to the node in the lower right corner of the map: E and SE.

The compression method adopts RLE, which compresses the first-move array by more compactly representing the substring(called runs) composed of the repetition of the same symbol. For example, the first line of symbol string in Fig.1 W;W;W;(W,E);E;E;E can be represented by two runs WW and EEE. Each such substring can be effectively replaced by a pair of values, one of which represents the start index (substring start position) and the other represents the relevant symbol. RLE finally represents the symbol string in the first line of the example as 1W;5E more concisely. The entire first-move array is compressed into 11 runs: 1W;5E;8W;12E;15W;20E;22W;26E;29SW;32S;33SE. Because there is no need to find the movement from s to the obstacle node and the source node, such nodes are assigned a label called the wildcard "*" by default.

After the preprocessing step is completed, the online search stage can retrieve the shortest path by repeatedly calling the CPD search function CPD(s, t), which performs a binary search on the compressed string of a given source node s to find the best first-move of the target node t. The extraction process of the shortest path is shown in Algorithm 1[Chiari *et al.*, 2019].

W	W	W	W,E	E	E	E
W	W	W	W,E	E	E	E
W	W				E	E
W	W	W	*s*	E	E	E
W,SW	W,SW	SW	S	SE	E,SE	E,SE

Figure 1: The optimal first moves to each cell of the gridmap from the cell marked *s*. Black cells indicate obstacles.[Chiari *et al.*, 2019]

Heuristic Redundant Symbols. Heuristic redundant symbols reduce the size of CPD by using heuristic information. The specific contents are as follows

Set s as the start node, t as the target node, and function $F_x(s,t)$ returns the movement(heuristic move) of the minimum estimated distance through node *n*, and the expression is[Chiari *et al.*, 2019]:

$$F_x(s,t) = \arg\min_{(s,n)\in E}\{w(s,n) + f_x(n,t)\} \tag{1}$$

Where $w(s,n)$ is the cost of walking, the straight line direction is 1, the oblique direction is $\sqrt{2}$, and $f_x(n,t)$ is a predefined heuristic distance function. If function $f_x(n,t)$ returns the first-move array *T(t)* belonging to *s*, then the heuristic redundant symbol "h" is added to *T(t)* of *s*. Heuristic redundant symbols make the compression step more flexible to select the symbols to be stored. Heuristic distance function can be calculated simply, such as octile distance function

$$\begin{aligned} f_o(n,t) &= \sqrt{2} * c + |n.x - t.x| - c + |n.y - t.y| - c \\ where \quad c &= min(|n.x - t.x|, |n.y - t.y|) \end{aligned} \tag{2}$$

or the Euclidean distance function

$$f_e(n,t) = \sqrt{(n.x - t.x)^2 + (n.y - t.y)^2}. \tag{3}$$

Proximity Wildcards. Proximity wildcards further exploits the potential of heuristic move by paying attention to the adjacent areas around nodes. Generally, for any node, the nodes contained in the surrounding adjacent area will use heuristic move as the first-move. Using this idea, the concept of proximity wildcards is introduced to further improve the compression efficiency of CPD.

Definition 1. *Given a node s and a function $F_x(s,n)$, the proximity distance pd(s) is the lowest value $d \in N$ such that there exists a cell n for which $|s.x\text{-}n.x| \leq d+1$ or $|s.y\text{-}n.y| \leq d+1$, and $F_x(s,n) \notin T(n)$ for s, where T is the first-move array for s.[Chiari* et al.*, 2019]*

Definition 2. *The proximity square of a node s is the square centred in s and with the edge size equal to $2 \cdot pd(s) + 1$.[Chiari* et al.*, 2019]*

Algorithm 2 CPDHP(*s*, *t*)

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

```
d ← pd(s)
if |s.x-n.x| ≤ d ∧|s.y-n.y| ≤ d then
    return F_x(s,t)
else
    m ← CPD(s,t)
    if m = "h" then
        return F_x(s,t)
    else
        return m
    end if
end if
```

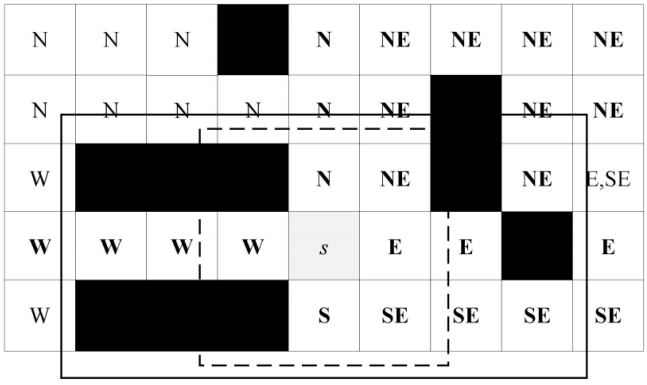

Figure 2: Heuristic move for source cell s. Heuristic moves that coincide with optimal moves are shown in bold.

In short, the grid cells in the adjacent square of any node *s* can be reached from *s* optimally by heuristic move. Add the proximity wildcards symbol "*" to each such cell. Algorithm 2 shows the CPD lookup function CPDHP(*s*, *t*) using proximity wildcards[Chiari *et al.*, 2019].

4 RPW Strategy

In this paper, we propose RPW strategy to further improve the compression efficiency of CPD by expanding the adjacent area concerned by proximity wildcards. The main idea is to expand the focus area of proximity wildcards from square to rectangle centered on any node, so as to consider more heuristic move.

Definition 3. *Given a node s and a function $F_x(s,n)$, the width of the largest adjacent rectangle R centered on s is expressed as sq(s).x and length are expressed as sq(s).y, any node $n \in R$, $f_x(s,n) \in T(n)$, T is the first-move array of s.*

The cells in the RPW area can arrive from *s* optimal by heuristic move. Add the RPW symbol "*" to each such cell. Fig.2 shows the map heuristic move and the first move coincidence area, in which the dotted line area is the proximity wildcards area and the solid line area is the improved RPW area.

As can be seen from the example, the compression result of the first-move array added with proximity wildcards is:

Algorithm 3 CPDHRP(s, t)

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

$X \leftarrow$ sq(s).x
$Y \leftarrow$ sq(s).y
if |s.x-n.x| $\leq X/2 \wedge$|s.y-n.y| $\leq Y/2$ **then**
  return $F_x(s,t)$
else
  $m \leftarrow$ CPD(s, t)
  if m = "h" **then**
    return $F_x(s,t)$
  else
    return m
  end if
end if

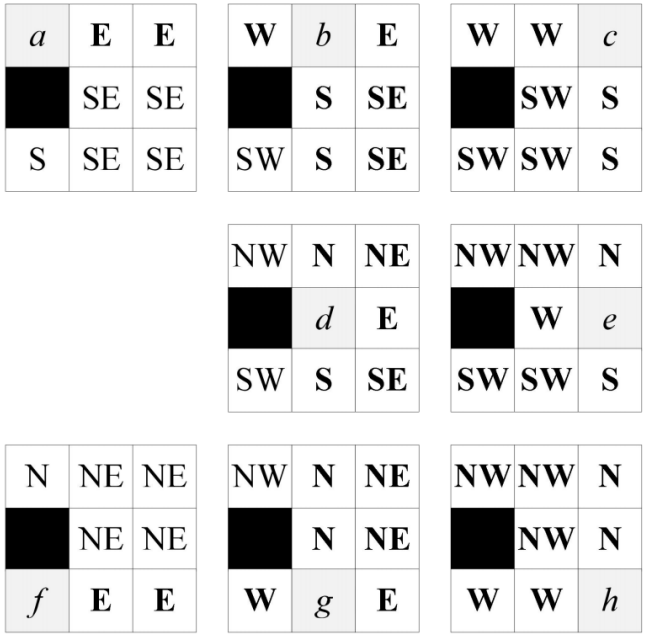

Figure 3: Heuristic moves, for each source cell.

1N;5h;10N;14h;19W;26h;27E;28h;37W;43h, 10 RLE runs in total. The compression result of the first-move array added with RPW is: 1N;5h;10N;14h;19W;27E;28h;37W;45h, 9 RLE runs in total.

It can be seen from the example that RPW produces less RLE runs. In the preprocessing stage, fewer RLE runs produce smaller CPD, that is, improve the compression efficiency of CPD. In the online search stage, the adjacent area is improved from square to rectangle, which expands the area of heuristic move nodes and reduces the number of binary search nodes, so as to improve the search efficiency. Algorithm 3 shows the CPD lookup function CPDHRP(s, t) using RPW.

5 CPW Strategy

Based on the idea of RPW, we further tap the potential of heuristic move and propose CPW. The main idea is to divide four quadrants with any node as the origin, and calculate the maximum rectangle with the origin as the corner of each quadrant, including the heuristic move as the first move.

Algorithm 4 CPDHCP(s, t)

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

if GetCPW(s, t) **then**
  return $F_x(s,t)$
else
  $m \leftarrow$ CPD(s, t)
  if m = "h" **then**
    return $F_x(s,t)$
  else
    return m
  end if
end if

Algorithm 5 GetCPW(s, t)

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

for $n = 1$ to 4 **do**
  $X.n \leftarrow sq(s)_n.x$
  $Y.n \leftarrow sq(s)_n.y$
end for
Judge that node t is relative to s, which belongs to the nth quadrant
if |s.x-n.x| $\leq X.n \wedge$|s.y-n.y| $\leq Y.n$ **then**
  return true
else
  return false
end if

Definition 4. *Given a node s and a function $F_x(s,n)$,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 $sq(s)_m.x$ and $sq(s)_m.y$, m∈[1,4], any node $n \in C$, $f_x(s,n) \in T(n)$, T is the first-move array of s.*

Fig.4 is a gridmap for explaining CPW, with 8 cells labelled from *a* to *h*. As can be seen from the Fig.4, the compression result of the first-move array added with proximity wildcards is: 1E;1S;1*;1N2h7S8h;1*;1E;1N;1*, 11 RLE runs in total. The compression result of the first-move array added with CPW is: 1E;1S;1*;1N7S;1*;1E;1N;1*, 9 RLE runs in total.Algorithm 4 shows the CPD lookup function CPDHCP(s, t) using CPW.

6 Experiments

Our experimental map uses GPPC benchmark data set, and the detailed content is available on the website https://movingai .com/benchmarks/grids.html. The experiment compares SRC and CPDs_PW agorithms. The former is the fastest optimal routing algorithm in GPPC2014, and the latter is one of the latest significant improvements to improve the compression efficiency of CPDs algorithm proposed by Chiari et al. In 2019. A total of 20 maps were randomly selected in the experiment, including 10 large maps (the number

Maps	SRC	CPDs_PW	CPDs_RPW	CPDs_CPW
AcrosstheCape	4.91604e+08	3.17361e+08	**3.11477e+08**	4.19476e+08
Aftershock	1.55618e+08	7.75273e+07	**7.18444e+07**	1.24055e+08
Ost000a	2.25754e+08	1.77554e+08	**1.60905e+08**	2.60633e+08
Ost000t	5.36637e+08	2.04589e+08	**1.80405e+08**	4.51991e+08
Hrt000d	1.92737e+08	1.25186e+08	**1.17753e+08**	1.84425e+08
Aurora	6.18983e+08	9.15982e+08	**5.24567e+08**	5.5833e+08
Room-400-40	2.18523e+08	8.22203+07	**7.74602e+07**	1.12926e+08
Brushfire	3.85427e+08	1.06776e+08	**1.00989e+08**	1.70673e+08
Backwoods	2.41827e+08	1.49382e+08	**1.49275e+08**	2.30434e+08
Archipelago	3.77199e+08	1.45075e+08	**1.4068e+08**	1.95728e+08
Lt-0-lowtown	9.4645e+07	3.86112e+07	**3.51392e+07**	5.8889e+07
Hrt201n	7.4938e+07	3.00384e+07	**2.81146e+07**	4.87035e+07
Lak100c	2.67651e+08	7.70831e+07	**7.48599e+07**	1.17073e+08
Brc999d	7.226e+06	4.85746e+06	**4.67299e+06**	7.13153e+06
Brc000d	4.7518e+07	6.14427e+07	**2.02706e+07**	2.7048e+07
Combat	2.5543e+07	1.80605e+07	**1.12542e+07**	1.45937e+07
Combat2	2.2722e+07	2.22453e+07	**1.14053e+07**	1.47449e+07
Den000d	7.7793e+07	8.33914e+07	**4.44911e+07**	6.48382e+07
Isound1	2.297e+06	1.21058e+06	**9.45433e+05**	1.34359e+06
Orz000d	6.122e+06	3.03397e+06	**2.96867e+06**	3.60955e+06

Table 1: Search time (ns)

of nodes ranges from 100000 to 300000) and 10 small maps (the number of nodes ranges from thousands to tens of thousands).

In Table 1, we compare the search time of the algorithm. It can be seen that the search time of our proposed CPDs_RPW algorithm on all maps is better than that of CPDs_PW algorithm and performs best in all comparison algorithms. Although compared with CPDs_PW algorithm, CPDs_CPW algorithm has higher search time efficiency on only 25% of the maps, because its query function is slightly complex. However, compared with CPDs_PW algorithm, the search time is less than SRC algorithm on 85% of the maps, and CPDs_CPW algorithm has better time performance than SRC algorithm in 95% of the maps.

In Table 2, we compare the CPD size of the algorithm. The CPD size (the third row) of the last three column algorithm is composed of the size of the first-move arry (the first row) and auxiliary data (the second row) It can be seen from the table that the CPD size generated by CPDs_RPW algorithm on all maps is smaller than that of CPDs_PW algorithm, and the average compression efficiency is increased by 0.626% and the maximum is increased by 3.644%. Although CPDs_CPW algorithm only has a smaller CPD size than CPDs_PW algorithm on the three maps of ost000a, ost000t and brc000d, CPDs_CPW algorithm performs best among all algorithms in the first-move arry size. The reason for this result is that although CPDs_CPW generates a smaller first-move arry by using more heuristic information, its auxiliary data is slightly complex. Compared with CPDs_RPW, it needs to record auxiliary data of 4 times the size. In the future, the overall compression capacity can be improved by proposing a suitable data structure to store auxiliary information.For the first-move arry size index, the CPDs_CPW algorithm is smaller than the CPDs_RPW algorithm on all maps, and the CPDs_RPW algorithm is smaller than the CPDs_PW algorithm, because the CPW area is greater than or equal to the RPW area, and the RPW area is greater than or equal to the PW area.

7 Conclusions

In this paper, we propose a strategy to expand the area of proximity wildcards, which makes the selection of storage symbols more flexible in the compression process of CPDs algorithm, and makes more heuristic information available in the online search process, so as to compress the preprocessing memory of the algorithm and speed up the search speed. The experimental results show that our CPDs_RPW algorithm is better than the famous CPDs_PW algorithm in CPD size and search time. Although CPDs_CPW algorithm is only smaller than CPDs_PW algorithm in CPD size on three maps, it performs best in all comparison algorithms in the first-move arry size index, and has the value of further mining. For the search time, although CPDs_CPW algorithm is only less than CPDs_PW algorithm on 25% of the maps, compared with 85% of the maps, CPDs_PW algorithm has better search time than SRC algorithm, an increase of 10 percentage points.

Future work includes further improving the storage structure of CPW, or extending the method proposed in this paper to a more general map environment.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (62076108,61872159) and the Natural Science Foundation of Jilin Province,China (20210101172JC).

Maps	#cells	SRC	CPDs_PW	CPDs_RPW	CPDs_CPW
AcrosstheCape	392287	393,982,688	93,442,946	93,399,202	**90,807,038**
			1,569,152	1,569,148	6,276,592
			95,012,098	**94,968,350**	97,083,630
Aftershock	166076	98,649,280	8,489,870	8,431,926	**7,031,206**
			664,308	664,304	2,657,216
			9,154,178	**9,096,230**	9,688,422
Ost000a	130478	43,694,864	15,235,418	15,140,750	**13,665,046**
			521,916	521,912	2,087,648
			15,757,334	**15,662,662**	15,752,694
Ost000t	105707	37,412,692	12,905,754	12,832,906	**11,588,290**
			422,832	422,828	1,691,312
			13,328,586	**13,255,734**	13,279,602
Hrt000d	106608	63,647,320	10,623,498	10,576,214	**9,446,582**
			426,436	426,432	1,705,728
			11,049,934	**11,002,646**	11,152,310
Aurora	493772	721,667,952	339,836,578	339,707,250	**337,770,534**
			1,975,092	1,975,088	7,900,352
			341,811,670	**341,682,338**	345,670,886
Room-400-40	152811	115,268,864	73,630,454	73,532,894	**72,066,842**
			611,248	611,244	2,444,976
			74,241,702	**74,144,138**	74,511,818
Brushfire	104896	53,078,056	8,689,822	8,670,174	**8,148,766**
			419,588	419,584	1,678,336
			9,109,410	**9,089,758**	9,827,102
Backwoods	208734	104,640,280	12,977,478	12,976,142	**12,171,646**
			834,940	834,936	3,339,744
			13,812,418	**13,811,078**	15,511,390
Archipelago	131770	77,202,656	35,521,790	35,394,978	**34,175,678**
			527,084	527,080	2,108,302
			36,048,874	**35,922,058**	36,283,980
Lt-0-lowtown	25259	5,340,272	1,783,246	1,761,462	**1,609,202**
			101,040	101,036	404,144
			1,884,286	**1,862,498**	2,013,346
Hrt201n	23652	3,214,980	876,982	841,578	**719,314**
			94,612	94,608	378,432
			971,594	**936,186**	1,097,746
Lak100c	34832	8,267,200	1,467,790	1,457,886	**1,271,870**
			139,332	139,328	557,312
			1,607,122	**1,597,214**	1,829,182
Brc999d	12847	6,233,664	691,242	687,446	**604,238**
			51,392	51,388	205,552
			742,634	**738,834**	809,790
Brc000d	28963	10,904,540	3,276,682	3,257,610	**2,970,982**
			115,856	115,852	463,408
			3,392,538	**3,373,462**	3,371,390
Combat	32967	27,875,952	2,769,594	2,740,054	**2,500,334**
			131,872	131,868	527,472
			2,901,466	**2,871,922**	3,027,806
Combat2	32929	27,717,888	2,756,078	2,726,894	**2,494,490**
			131,720	131,716	526,864
			2,887,798	**2,858,610**	3,021,354
Den000d	58085	23,714,500	9,540,390	9,512,398	**8,970,534**
			232,344	232,340	929,360
			9,772,734	**9,744,738**	9,899,894
Isound1	2976	587,424	60,002	59,574	**54,242**
			11,908	11,904	47,616
			71,910	**71,478**	99,858
Orz000d	4057	973,936	150,822	150,634	**124,230**
			16,232	16,228	64,912
			167,054	**166,862**	189,142

Table 2: The size of the CPD in MB

References

[Botea and Harabor, 2013] Adi Botea and Daniel Harabor. Path planning with compressed all-pairs shortest paths data. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, Rome, Italy*, pages p.10–14, 2013.

[Chiari *et al.*, 2019] Mattia Chiari, Shizhe Zhao, Adi Botea, Alfonso Emilio Gerevini, Daniel Harabor, Alessandro Saetti, Matteo Salvetti, and Peter J. Stuckey. Cutting the size of compressed path databases with wildcards and redundant symbols. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, Berkeley, CA, USA*, pages p.11–15, 2019.

[Cohen *et al.*, 2018] Liron Cohen, Tansel Uras, Shiva Jahangiri, Aliyah Arunasalam, Sven Koenig, and T. K. Satish Kumar. The fastmap algorithm for shortest path computations. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, Stockholm, Sweden*, pages p.13–19, 2018.

[Freund and Hoyer, 2003] E. Freund and H. Hoyer. Pathfinding in multi-robot systems: Solution and applications. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, 2003.

[Harabor and Stuckey, 2018] Daniel Damir Harabor and Peter J. Stuckey. Forward search in contraction hierarchies. In *Proceedings of the Eleventh International Symposium on Combinatorial Search, Stockholm, Sweden*, pages p.14–15, 2018.

[Hu *et al.*, 2019] Yue Hu, Daniel Harabor, Long Qin, Quanjun Yin, and Cong Hu. Improving the combination of JPS and geometric containers. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, Berkeley, CA, USA*, pages p.11–15, 2019.

[Hu *et al.*, 2021] Yue Hu, Daniel Harabor, Long Qin, and Quanjun Yin. Regarding goal bounding and jump point search. *J. Artif. Intell. Res.*, 70:631–681, 2021.

[Rabin and Sturtevant, 2016] Steve Rabin and Nathan R. Sturtevant. Combining bounding boxes and JPS to prune grid pathfinding. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA*, pages p.12–17, 2016.

[Salvetti *et al.*, 2017] Matteo Salvetti, Adi Botea, Alessandro Saetti, and Alfonso Emilio Gerevini. Compressed path databases with ordered wildcard substitutions. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, Pittsburgh, Pennsylvania, USA*, pages p.18–23, 2017.

[Salvetti *et al.*, 2018a] Matteo Salvetti, Adi Botea, Alfonso Emilio Gerevini, Daniel Harabor, and Alessandro Saetti. Two-oracle optimal path planning on grid maps. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, Delft, The Netherlands*, pages p.24–29, 2018.

[Salvetti *et al.*, 2018b] Matteo Salvetti, Adi Botea, Alfonso Emilio Gerevini, Daniel Harabor, and Alessandro Saetti. Two-oracle optimal path planning on grid maps. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, Delft, The Netherlands*, pages p.24–29, 2018.

[Sturtevant and Rabin, 2016] Nathan R. Sturtevant and Steve Rabin. Canonical orderings on grids. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, USA*, pages p.9–15, 2016.

[Sturtevant *et al.*, 2015] Nathan R. Sturtevant, Jason M. Traish, James R. Tulip, Tansel Uras, Sven Koenig, Ben Strasser, Adi Botea, Daniel Harabor, and Steve Rabin. The grid-based path planning competition: 2014 entries and results. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, Ein Gedi, the Dead Sea, Israel*, pages p.11–13, 2015.

[Uras and Koenig, 2014] Tansel Uras and Sven Koenig. Identifying hierarchies for fast optimal search. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Québec City, Québec, Canada*, pages p.27–31, 2014.

[Uras and Koenig, 2017] Tansel Uras and Sven Koenig. Feasibility study: Subgoal graphs on state lattices. In Alex Fukunaga and Akihiro Kishimoto, editors, *Proceedings of the Tenth International Symposium on Combinatorial Search, Pittsburgh, Pennsylvania, USA*, pages p.16–17, 2017.

[Wu *et al.*, 2012] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proc. VLDB Endow.*, 5(5):406–417, 2012.

[Xiao and Hao, 2011] C. Xiao and S. Hao. A*-based pathfinding in modern computer games. *International Journal of Computer ence Network Security*, 11(1):p.125–130, 2011.

[Zhao *et al.*, 2020] Shizhe Zhao, Mattia Chiari, Adi Botea, Alfonso Emilio Gerevini, Daniel Harabor, Alessandro Saetti, and Peter J. Stuckey. Bounded suboptimal path planning with compressed path databases. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France*, pages p.26–30, 2020.