Optimally Solving Cooperative Path-Finding Problems Without Hole on Rectangular Boards with Heuristic Search

Bruno Bouzy
LIPADE, Université Paris Descartes, FRANCE
bruno.bouzy@parisdescartes.fr

Abstract

This work optimally solves Cooperative Path-Finding (CPF) problems without hole on rectangular boards $M \times N$ by using databases and heuristic search. It fills a gap in the CPF literature that contains few work specific to CPF without hole. In a first part, we show that small boards are optimally solved with a direct database technique. Intermediate size boards (inferior to 4×5) are optimally solved with IDA* using a set of admissible heuristics: stage-one heuristics of a two-stage algorithm and maxspan heuristics on tuples. Moreover, some 5×5 boards are optimally solved with the same technique. In a second part, we assess the effects using stage-one heuristics and maxspan heuristics with tuples of size at most 4.

1 Introduction

Very informally, a Cooperative Path-Finding (CPF) problem consists in moving agents situated on locations to goal locations, one for each agent [Silver, 2006]. At their turn or during a timestep, the agents may move to adjacent locations or stay. Basically, they must avoid collisions. (1) No two agents can be on the same location at the same time. (2) No two agents can swap locations when they are situated on adjacent nodes. In the sequential background, the goal is to minimize the number of elementary movements. In the parallel context, the goal is to minimize the number of timesteps. In the former context, and under the assumption that some empty locations - holes - exist, many work have been achieved [Boyarski *et al.*, 2015a]. However, very few work have been done in the latter formulation under the assumption that no hole exists [Yu and LaValle, 2012b]. The primary goal of the current paper is to assess to what extent databases and heuristic search can be used to optimally solve CPF without hole on rectangular boards. Besides, the CPF problems without hole share an analogy with the Rubik's cube which can be seen as a CPF problem without hole in three dimensions in which the cubies are the agents. The secondary goal is to assess to what extent the techniques successful for the Rubik's cube [cub, 2014] can be transfered to CPF without hole on rectangular boards: databases, two-stage-algorithm based heuristics [Kociemba, 1990], and IDA* [Korf, 1985].

The paper is structured as follows. Section 2 sums up the work performed in CPF, and Rubik cube techniques. Section 3 defines the CPF problem without hole on rectangular boards. Section 4 describes the principle underlying our approach. Before conclusion, section 5 gives the results.

2 Related work

This section first relates work achieved in CPF and, secondly, work done to solve the Rubik's cube.

2.1 CPF

The work on CPF is twofold: mostly work done in the sequential context on the one hand, and little work done in the simultaneous context on the other hand. In the sequential context, the agents act elementarily one after each other, and the goal is to minimize the number of elementary actions. In this context, the existence of empty locations or holes is an assumption to enable some agents to move. The problem is Pspace-hard [Hopcroft *et al.*, 1984]. Finding a shortest solution to the N puzzle with one hole is intractable [Ratner and Warmuth, 1990]. Most of the work done in this context use algorithms derived from A* [Hart *et al.*, 1968] and look for optimal solutions. The first algorithm derived from A* was WHCA* by Silver [Silver, 2006]. A family of algorithms uses A* on conflicts between agents. Conflict-Based Search (CBS) launches the agents on their individual paths, detects the conflicts, determines how to solve them by forbidding some adequate actions, and so on iteratively [Sharon *et al.*, 2012a]. CBS has been extended in several ways: adding a meta-agent [Sharon *et al.*, 2012b], bypassing some conflicts [Boyarski *et al.*, 2015a], or adding all extensions: Improved CBS (ICBS) [Boyarski *et al.*, 2015b]. Operator Decomposition (OD) [Standley, 2010] and Independence Detection (ID) [Standley and Korf, 2011] are two important features of algorithms derived from A*. Multi-Agent Path Planning (MAPP) [Wang and Botea, 2011] presents a method complete on the slideable class of problems. M* [Wagner and Choset, 2011] develops searches to solve conflicts between robots. [Peasgood *et al.*, 2008] [Ryan, 2007] [van den Berg *et al.*, 2009] are other work in multi-robot path planning. Non-optimal approaches have been published [Röger and Helmert, 2012] [Wilson, 1974] [Kornhauser *et al.*, 1984]. Domain-dependent polynomial-time algorithms such as Push and Swap [Luna and Bekris, 2011], TASS [Khorshid *et al.*, 2011], BIBOX

[Surynek, 2009] complete the picture. We remind that the algorithms mentioned above assume that at least one hole exists.

Little work has been published on the parallel context. Network-flow based approaches by Yu and LaValle [Yu and LaValle, 2012b] solves optimally 4×4 and 5×5 CPF problems without holes [Yu and LaValle, 2012a] with a network flow approach using linear programming, and Bouzy solves a wide class of problems with a Monte-Carlo approach [Bouzy, 2013]. Importantly, these work deal with CPF problems without hole in the parallel context. [Kornhauser *et al.*, 1984] and [Surynek, 2009] may deal with problems without hole, but not in the parallel context.

2.2 Rubik's cube

The Rubik's cube is a famous puzzle invented by Rubik in 1974. The cube is made up with 27 cubies situated on 27 locations. The cube has 6 faces: Up (U), Down (D), Right (R), Left (L), Front (F) and Back (B). Each cubie has facets: 3 facets for the 8 top cubies, 2 facets for the 12 edge cubies and 1 facet for the 6 central face cubies. A facet has one of the six colors. There is a set of 18 legal actions G. An action corresponds to one of the 6 faces of the cube, and makes the cubies situated on this face turn clockwise, counter-clowise, or half-turn around the axis perpendicular to the face. For instance, action $U+$ (respectively $U-$, and $U2$) makes the cubies of the U face turn clockwise (respectively counter-clockwise, and half-turn). The goal is to obtain the ordered cube: a cube with each of the 6 faces coloured with one color only. The success of the Rubik's cube is legendary. The eighties witnessed the use of personal computers, and many Rubik cube solvers were created. Thistlethwaite devised a clever algorithm for solving the Rubik's Cube in less than 52 moves [Thi, 1981]. His idea was to solve it in four stages. Later on, Herbert Kociemba simplified the idea, and devised the famous two-stage algorithm [Kociemba, 1990] decreasing the upper bound down to 30 moves. The second stage of the two-stage algorithm, corresponds to positions belonging to the "identity coset", and to a specific subset of actions of the cube: $G1 = \{U+, D+, U-, D-, U2, D2, R2, L2, F2, B2\}$. The positions of the identity coset have the following properties. The U face and the D face are correctly completed. The four cubies of the intermediate level are situated in the intermediate level, but not necessarily at the correct location. By only applying actions in G1, the positions belonging to the identity coset remain in the identity coset. Previously, there is a first stage in which each position belong to a specific coset. During stage 1, the goal is to reach a position of the identity coset.

The insight of splitting the solving process into two stages is not hazardous regarding the size of computer memories. The number of cube's positions equals $3^7 \times 2^{11} \times 8! \times 12!/2 \simeq 43 \times 10^{18}$ which does not fit actual size of current computer memories. However, the size of a coset, $40320 \times 40320 \times 24/2 \simeq 19 \times 10^9$ fits current computer memories when considering that two values can be stored in one byte. Moreover, the number of cosets, $2187 \times 2048 \times 495 \simeq 2,2 \times 10^9$, fits current computer memories as well. Thus, Kociemba idea was to build one database of cosets with the distance to the identity coset, and one database of positions of the identity coset with the distance to the ordered cube. Then, to solve a specific Rubik's cube position, proceed in two stages. First, look at the coset database to get the distance $d1$ to the identity coset, browse the coset database until the identity coset, then read the distance $d2$ to the ordered cube. A very fast but sub-optimal solution is obtained this way. $d1$ is an admissible heuristic in stage 1, and $d2$ in stage 2. In addition, $d1 + d2$ provides an upper bound on the optimal length. Then, IDA* [Korf, 1985], combined with an appropriate use of the two heuristics, optimally solves any position in a reasonable time on current computers. The hardest positions can be solved in few hours on current computers [Rokicki, 2008]. The Kociemba's approach with databases is more efficient than the Korf's similar approach [Korf, 1997]. Recently, the diameter of the graph of the Rubik's cube graph has been shown to be equal to 20 in the half-turn metric [Rokicki *et al.*, 2013] and 26 in the quater-turn metric [cub, 2014]. To obtain these results, many clever ideas have been necessary. However, these techniques are not used in the current work.

4	8	0
7	5	3
2	1	6

0	1	2
3	4	5
6	7	8

Figure 1: Agents: their positions (left) and their goals (right).

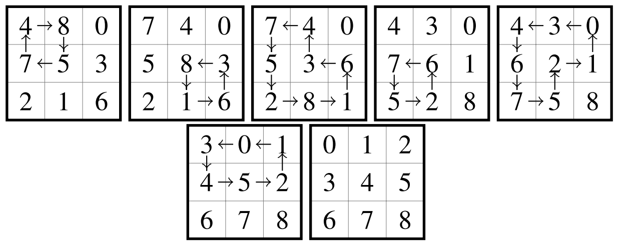

Figure 2: An example of solution.

3 Definition of the problem

Figure 1 on the left is a 3×3 CPF position without hole with 9 agents. Each agent has a number and is situated on one cell. Figure 1 on the right is the goal position. Each agent has to reach its own goal. For instance, agent 4 has to move from the leftmost and topmost cell to the central cell. During one timestep, all the agents act simultaneously while respecting the CPF rules. An agent can move to one of its adjacent cells, or stay on its cell. The agents must avoid collisions. (1) No two agents can be on the same cell at the same time. (2) No two agents can swap cells when adjacent. The problem consists in moving the set of agents toward its goal in a minimal number of timesteps. Figure 2 shows a solution to the problem of Figure 1. An arrow between two cells indicates the movement of the agent for the next timestep. After 6 timesteps, the agents are ordered according to their goal. The board has a size $M \times N$, where M is the height and N is the width of the board. A denotes the number of agents. Because, we consider CPF problems without hole, we have

Table 1: Number of states.

$M \times N$	2	3	4	5	6
2	24	720	40,320	3.6 10^6	4.8 10^8
3		3.6 10^5	4.8 10^8	1.3 10^{12}	6.4 10^{15}
4			2.1 10^{13}	2.4 10^{18}	6.2 10^{23}
5				1.5 10^{25}	2.6 10^{32}
6					3.7 10^{41}

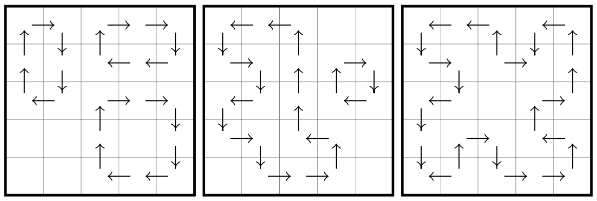

Figure 3: Three examples of 5×5 joint actions.

Table 2: Number of joint actions.

$M \times N$	2	3	4	5	6
2	2	6	16	40	98
3		26	116	492	2,090
4			950	7,454	58,924
5				107,150	1,559,148
6					41,921,394

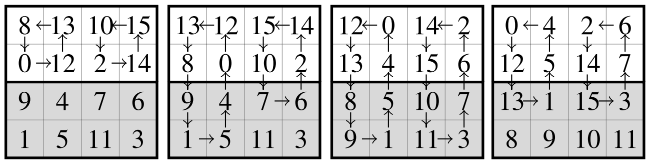

Figure 4: The first stage of the 2-stage solution of a 4×4 position: driving the low (respectively high) numbered agents to the white (respectively grey) 2×4 sub-board.

$A = M \times N$.

4 Our approach

Our approach heavily depends on the complexity of the problem in terms of number of states and number of joint actions. According to this complexity, our approach uses either a direct construction of a database, or IDA* with two set of heuristics: maxspan-over-tuples heuristics and stage-one heuristics derived from 2-stage algorithms. These heuristics are admissible.

4.1 Complexity

The number of states equals $A!$. Table 1 gives the number of states. For $A \leq 12$, $A!$ is less than half a billion. In these cases, CPF problems can be solved directly and optimally with a database of distances to the goal state, which is possible for boardsizes inferior to 3×4 or 2×6. Because the number of states for Rubik's cube is $4,3 \times 10^{19}$, we expect to be able to solve CPF problems with Rubik's cube techniques, for boardsizes inferior to 4×5 ($A <= 20$). However, solving CPF problems for higher sizes with the same techniques remains challenging.

In a CPF problem, a joint action is composed with elementary actions. An elementary action is a movement from one cell to an adjacent cell, or a stay. To be valid a joint action must respect the CPF rule (1) and (2). Therefore, as illustrated by the three examples of Figure 3, the elementary actions build up cycles. A joint action is made up with one or several cycles. On the first example, there are three cycles. The joint action is made up with 20 elementary actions. On the second example, there are two cycles. One is made up with 16 elementary actions, and the other one has 4 elementary actions. On the third example, there is one unique cycle of length 24.

Generating SJA, the complete set of joint actions, corresponding to a $M \times N$ board needs care. The joint action generator must be able to generate the cycles and combine them into a joint action. Because the boards are always completely filled with agents, the good news is that SJA depends only on M and N and not on the agents themselves. Consequently, SJA can be generated offline, once for each boardsize. For each boardsize, Table 2 gives $|SJA|$, the number of joint actions. When $A <= 12$, $|SJA|$ remains more or less inferior to one hundred. Consequently, generating SJA is easy and solving problems remain simple. For $(M, N) \in \{(3,5),(3,6),(4,4),(4,5)\}$, $|SJA|$ reaches few thousands, generating SJA remains simple, and optimally solving problems remains possible. For 5×5, $|SJA|$ reaches about 10^5. Generating SJA is not an issue. However, optimal problem solving becomes an issue. For 6×6, $|SJA|$ roughly equals 40×10^6, generating SJA is possible, but optimal problem solving becomes nightmarish. However, to find out short solutions to problems, the high number of possible actions in a given state must be seen as an advantage. The higher the number of actions, the shorter the solutions.

4.2 An example of a two-stage algorithm for CPF

When $12 < A <= 24$, $A!$ is to high to fit current computer memory. However, a 2-stage algorithm inherited from the Rubik's cube solving loosens up the situation. CPF 4×4 is an appropriate illustration of the 2-stage algorithm. Figure 4 shows a 4×4 position with the first stage of the solution output from the 2-stage algorithm. In this stage, the goal is to drive agents 0, 1, ..., 7 into the upper half of the board, and agents 8, 9, ..., 15 into the lower half. The first stage lasts 4 moves. Figure 5 shows the second stage of the solution output from the 2-stage algorithm. In this stage, agents 0, 1, ... 7 remains situated in the upper 2×4 board, and agents 8, 9, ..., 15 in the lower 2×4 board (in grey). The goal of the second stage is to solve the two 2×4 problems separately and simultaneously. The upper 2×4 problem is solved in 6 moves. Meanwhile the lower problem is solved in 6 moves as well. Overall, the second stage contains 6 moves. Globally, on this problem, the 2-stage algorithm outputs a sequence of $4 + 6 = 10$ moves in total. To achieve this kind of two-stage solving, two databases must be built beforehand, one database for each stage.

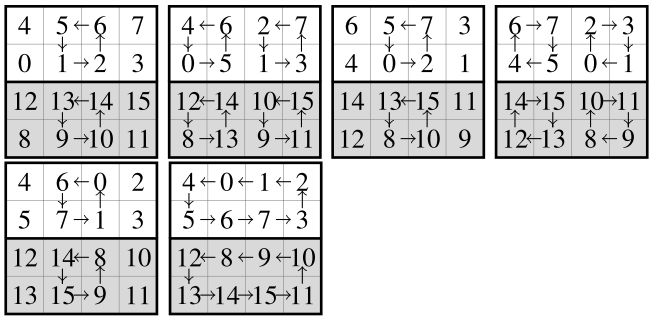

Figure 5: The second stage of the 2-stage solution of a 4×4 position: solving the grey sub-board and the white sub-board independently and simultaneously.

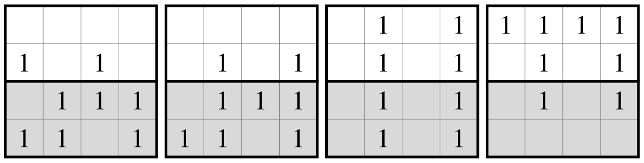

Figure 6: The sub-positions corresponding to the stage-one positions of Figure 4.

4.3 Stage-one heuristics

Each stage-one heuristic corresponds to a specific frontier splitting the whole board into two sub-boards, $sb1$ and $sb2$. For $2 \leq a \leq N$, $2 \leq b \leq N$, $a \neq b$ and $\max(a, b) = N$, we call $HSO_{N,a,b}$ the stage-one heuristic corresponding to the $N \times N$ board splitted into a $a \times b$ board, $sb1$, and its complement $sb2$. On 4×4 boards, we use two heuristic functions: $HSO_{4,2,4}$ and $HSO_{4,4,2}$. $HSO_{4,2,4}$ corresponds to the horizontal split shown by Figure 4 in which $sb1$ (respectively $sb2$) corresponds to the white (respectively grey) cells. $HSO_{4,4,2}$ corresponds to the vertical split. We call $HSO_N = \max_{a,b}(HSO_{N,a,b})$ the stage-one heuristic for $N \times N$ boards. Because all stage-one heuristics are admissible [1], HSO_N is admissible.

Databases $HSO_{N,a,b}$ containing $HSO_{N,a,b}(sp)$ for all sub-positions sp are built beforehand. A sub-position sp, or an entry, is a subset of the whole board such that each cell of sp is occupied by an agent whose goal is in $sb1$. Figure 6 shows the sub-positions corresponding to the positions of Figure 4. An entry contains the optimal distance to the goal.

The number of entries $NE_{SO,N,a,b}$ of the database $HSO_{N,a,b}$ equals $C_{N^2}^{N \times b}$. We have $NE_{SO,4,4,2} = 12,870$ and $NE_{SO,5,5,2} = 3,268,760$. The building process of a database is iterative. First, the goal entry is set with distance 0. Then, iteratively, for d increasing from 1 up to d_{max}, for any entry with a computed distance d, the distances of the neighbouring entries without distance are set to $d + 1$. The process stops when all entries have their distances computed. d_{max} is the maximal distance of an entry to the goal.

[1]Completing stage one is necessary to solve a problem

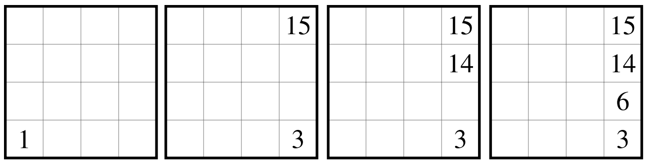

Figure 7: Four examples of tuples of size $T = 1, 2, 3, 4$ corresponding to the left position of Figure 4.

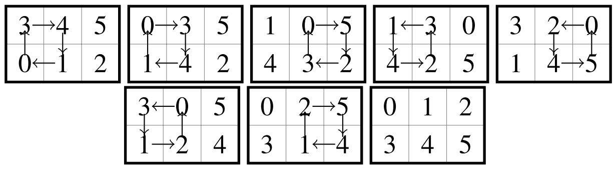

Figure 8: A hard 2×3 position with its optimal solution.

4.4 Stage-two heuristics

Stage-two heuristic can be defined as the maximum over the lengths of the solutions of the sub-boards. Unfortunately, Figure 10 shows the optimal solution of the initial position of Figure 5. Its length equals 4 which is less than 6, the value of stage-two heuristic. Consequently, stage-two heuristic is not admissible in CPF problems without hole. We do not use it in our work.

4.5 Maxspan-over-tuples heuristics

We use heuristic functions based on maxspan over tuples of agents. We call $HMT_{N,T}$ the maxspan heuristic over all the tuples of size T for boards of size $N \times N$. p being a position and t a tuple of agents, we define $HMT_{N,T}(p) = \max_t(HMT_{N,T}(p, t))$ where $HMT_{N,T}(p, t)$ is the optimal distance between tuple t and the tuple of goals on p, assuming that the agents not belonging to tuple t are removed. When $T = 1$, this heuristic corresponds to the usual maxspan heuristic: maximum over all the agents of the distance between the agent and its goal. When $T = 2$, this heuristic corresponds to the maximum over all pairs of agents of the distance between the pair of agents and their goals.

Figure 7 shows four examples of tuples of size $T = 1, 2, 3, 4$ that match the left position of Figure 4. We have $HMT_{4,1} = 4$ because of agent 1. We have $HMT_{4,2} = 5$ because of tuple (3, 15). We have $HMT_{4,3} = HMT_{4,4} = 5$. Because solving a given problem is harder than solving this problem with less agents, $HMT_{N,T}$ are admissible heuristics for any $T \leq N \times N$. For all N and p, we have $HMT_{N,T_1}(p) \geq HMT_{N,T_2}(p)$ when $T1 > T2$.

Databases $HMT_{N,T}$ containing $HMT_{N,T}(t)$ for all t of size T are built beforehand. The number of entries $NE_{N,T}$ of the database $HMT_{N,T}$ equals the number of tuples of agents, considering that the agents of the tuple can be ordered by increasing number, times the number of tuples of goals: $NE_{N,T} = C_{N^2}^{T} \times N^{2^T}$. We have $NE_{4,4} = 119,275,520 \approx 120M$ and $NE_{5,4} = 4,941,406,250 \approx 5G$. Consequently, due to memory and computation limitations, we built $HMT_{N,T}$ for $N = 4, 5$ and for $T = 1, 2, 3, 4$. The larger T, the tighter the admissible heuristic, and the higher its compu-

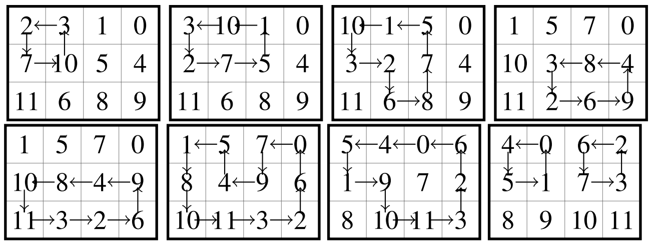

Figure 9: A hard 3×4 position with its optimal solution.

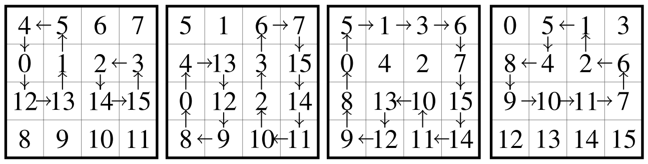

Figure 10: An optimal solution of the initial position of Figure 5.

tation time.

5 Experimental results

First, we give the results obtained in the size of the board, i.e. in A. We have 3 cases: $A \leq 12$, $12 < A \leq 24$, $A = 5 \times 5$. Secondly, we focus on admissible heuristics and their experimental evaluation. The experiments are performed on a 3.2 Ghz computer with 12 Gbytes of memory. The source programs are written in C++.

5.1 Results in A

First, for CPF with $A \leq 12$, we show two problems with their optimal solutions obtained with the database technique. Figure 8 shows one of the hardest 2×3 CPF position in which two agents situated on the same column have to swap (agent 0 and agent 3, 1 and 4, 2 and 5 respectively). The length of the optimal solution to this problem is 7. Figure 9 shows one of the hardest 3×4 CPF position whose the length of the solution is 8.

Secondly, for CPF with $12 < A \leq 24$, with our two sets of admissible heuristics (HSO_N and $HMT_{N,T}$), IDA* solves a CPF problem in a few seconds on average. Figure 11 shows an optimal solution of the CPF 4×4 problem of Figure 4 in 6 actions. Our approach also solves 3×6, 4×5 and some 4×6 problems. The higher A the longer the solving process.

Thirdly, for 5×5 boards, stage-one heuristics becomes almost useless. Considering that $25 = 12 + 13$, for stage 1, we have $25!/(12!13!) = 5,200,300$ states. However, several days of computations were necessary to obtain the database for stage 1. Using HMT heuristics is better than using HSO heuristics actually. Figure 12 shows the optimal solution of the CPF 5×5 position found in [Yu and LaValle, 2012b]. Our solution is obtained by IDA* using the $HMT_{5,4}$ heuristic after one day of computations. In our approach, 5×5 CPF with 10^5 actions and more than 10^{25} states remains an obstacle for heuristic search.

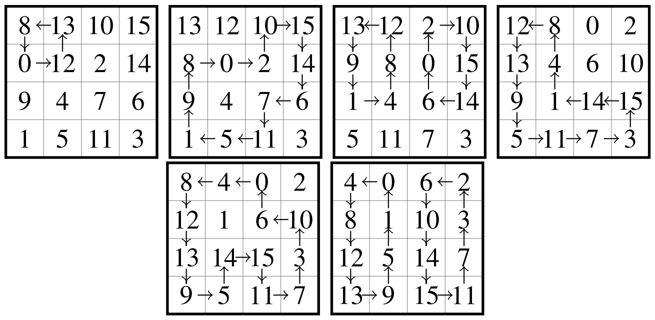

Figure 11: An optimal solution of the initial position of Figure 4.

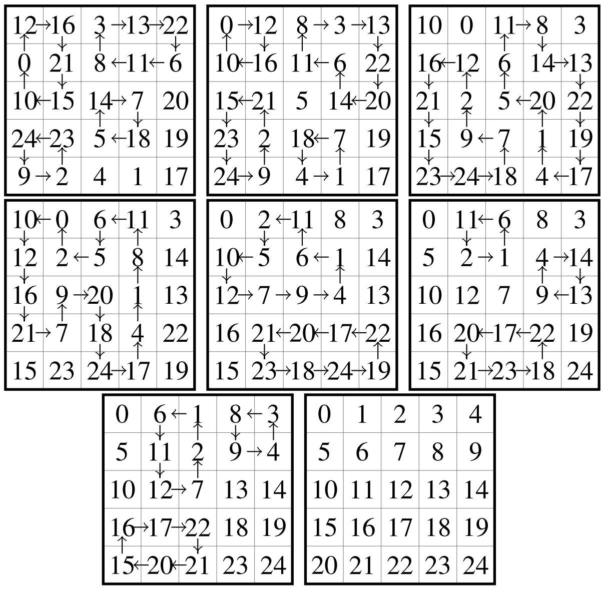

Figure 12: Optimal solution for a problem also solved in Yu and Lavalle.

5.2 Evaluating the heuristics

This section gives an assessment of HSO_N and $HMT_{N,T}$ for $N = 4,5$ and $T = 1,2,3,4$. For 4×4 boards, we use a set of positions randomly generated, i.e. likely hard to solve. For 5×5 boards, we use a set of *easy* positions obtained by swapping two agents (SA), or swapping two lines of agents (SL) or two columns (SC) or both (SLC), and the Yu's position.

Because the stage-two heuristic is not admissible, it cannot be used with optimality garantee. Thus, HSO_N cannot be used with its complementary heuristic. HSO_N is admissible but unfortunately it is not a tight heuristic. We tried to use it alone and the results were disappointing. $HMT_{N,T}$ is a better choice. However, to underline the effect of HSO_N, we compared the effect of using both HSO_4 in addition to $HMT_{4,1}$ with the effect of using $HMT_{4,1}$ alone. Table 3 shows the results in term of time used and number of nodes expanded by IDA*. The positive point is minor: the number of nodes expanded by using both HSO_4 and $HMT_{4,1}$ is slightly lower than the number of nodes expanded by using $HMT_{4,1}$ alone.

Table 3: HSO_4: time used (t) and number of nodes expanded (nn) by IDA* using $HMT_{4,1}$ alone, and using HSO_4 and $HMT_{4,1}$.

		$HMT_{4,1}$		$HMT_{4,1}$+HSO_4	
P	L	t	nn	t	nn
B6	6	0.2”	259k	0.7”	257k
B2	6	0.4”	806k	1.9”	803k
B1	6	0.7”	2.1M	4.6”	2.1M
S5	7	11”	42M	1’30”	39M
B7	7	16”	66.5M	2’25”	66M
B8	7	50”	196M	7’	195M
S6	6	1’15”	286M	10’30”	280M
C2	7	1’50”	434M	16’	433M
B3	6	11’15”	2.70G	1h40	2.70G
C1	7	7’20”	1.70G	1h	1.63G
B9	7	52’	12.1G	7h20	12.0G
S7	7	16’	3.77G	2h20	3.67G

Table 4: $HMT_{4,T}$: time used (t) and number of nodes expanded (nn) by IDA* for $T = 2, 3, 4$.

		2		3		4	
P	L	t	nn	t	nn	t	nn
B6	6	0.2”	174k	0.2”	113k	0.2”	72k
B2	6	0.2”	238k	0.3”	210k	0.3”	91k
B1	6	0.4”	956k	0.4”	670k	0.6”	427k
S5	7	5”	18M	5”	12M	5”	5.5M
B7	7	10”	38M	7”	19M	10”	10M
B8	7	25”	99M	17”	50M	23”	25M
S6	6	15”	60M	14”	37M	21”	24M
C2	7	33”	132M	25”	71M	40”	45M
B3	6	1’	210M	25”	59M	33”	34M
C1	7	2’	520M	90”	270M	150”	164M
B9	7	5’	989M	2’	280M	3’	165M
S7	7	6’	1.3G	4’	582M	5’	332M

The negative point is important: the time used is multiplied by ten when using HSO_4. HSO_4 is experimentally worse than $HMT_{4,1}$.

On our set of test positions of size 4×4, Table 4 shows the time and the number of nodes expanded by IDA* using $HMT_{4,T}$ for $2 \leq T \leq 4$. The higher T, the less expanded nodes by IDA*, but the less the reduction. Between $T = 2$ and $T = 3$, we observe a time reduction, but not between $T = 3$ and $T = 4$. The reason lies in the time to compute $\max_t(HMT_{4,T}(p, t))$ with t in a set of size C^T_{16}. For instance, $C^3_{16} = 560$, and $C^4_{16} = 1820$.

On a set of test positions of size 5×5, Table 5 shows the time and the number of nodes expanded by IDA* when using tuples of size T for $2 \leq T \leq 4$ on 5×5 boards. The higher T, the higher the number of solved problems, the less expanded nodes by IDA*, and the smaller the elapsed time. Between $T = 3$ and $T = 4$, the time reduction is still clear although the time to compute $HMT_{5,T}(p) = \max_t(HMT_{5,T}(p, t))$ with t in a set of size C^T_{25} can be long ($C^3_{25} = 2300$ and $C^4_{25} = 12650$). Even with the use of tuple-based maxspan heuristics, our heuristic search approach shows its limitations on 5×5 boards. The Yu’s position is solved in 4 days with $T = 3$, and in one day with $T = 4$. However, this position can be considered as easy compared to other 5×5 problems not solved by our approach.

Table 5: $HMT_{5,T}$: time used (t) and number of nodes (nn) expanded by IDA* for $T = 2, 3, 4$.

		2		3		4	
Pos.	L	t	nn	t	nn	t	nn
SA2	3	8”	10.3M	4”	2.3M	3”	1.5M
SLC2	4	2h	9.11G	4’	421M	3’	374M
SL2	4			23’	2.2G	7’	729M
SL1	4			24’	2.0G	12’	1.7G
SLC1	4			1h	6.5G	24’	3.3G
SL5	5			26h	164G	10h	77G
SL3	5			33h	199G	8h	55G
YU	7			90h	900G	23h	207G

6 Conclusion

Our contribution based on databases, stage-one heuristics and maxspan heuristics over tuples and IDA*, is adapted to solve $M \times N$ boards without hole. The maxspan heuristics over tuples of size $T = 1, 2, 3, 4$ surpass the stage-one heuristics in terms of number of nodes explored and time used. On 4×4 boards (respectively 5×5 boards), $T = 3$ (respectively $T = 4$) is an appropriate choice. The limit in terms of boardsize remains 4×5 for which most of problems can be solved optimally in a reasonable time, and 5×5 for which some easy problems can be solved optimally at the cost of days of computations on current computers. To this extent, our approach remains inferior to the Linear Programming approach of Yu [Yu and LaValle, 2012b].

However, our work is the first one using heuristic search to solve CPF problems without hole. For larger boards, optimal solving seems very hard to achieve with heuristic search, and, within the framework of combinatorial search, approximate approaches seem mandatory [Bouzy, 2013].

In addition to the weak relevance of stage-one heuristics that we experimentally observed, the stage-two heuristic is not an admissible heuristic for our CPF problems. Thus, the two-stage algorithm used for solving the Rubik’s cube is not easily transferable to CPF problems without hole within the heuristic search context. Instead, we experimentally showed the efficiency of tuple-based heuristics by providing explicit results with tuple size at most 4.

This work opens up new questions for future research. Could other techniques optimally solve CPF boards larger than 5×5 boards? Could problems with obstacles be solved? Could a no-hole CPF solver be incorporated within an open-space CPF solver with success? Is it possible to use tuple-based heuristics with tuple of size at least 5 ?

References

[Bouzy, 2013] Bruno Bouzy. Monte-Carlo Fork Search for Cooperative Path-Finding. In T. Cazenave, M. Winands, and H. Iida, editors, *Workshop on Computer Games*, number 408 in CCIS, pages 1–15, 2013.

[Boyarski *et al.*, 2015a] E. Boyarski, A. Felner, G. Sharon, and R. Stern. Don't split, try to work it out: Bypassing conflicts in multi-agent path finding. In *ICAPS*, 2015.

[Boyarski *et al.*, 2015b] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony. ICBS: Improved conflict-based search algorithm for multi-agent path finding. In *IJCAI*, 2015.

[cub, 2014] God's number is 20. http://www.cube20.org/, 2014.

[Hart *et al.*, 1968] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Sciences and Cybernetics*, SSC-4(2):100–107, 1968.

[Hopcroft *et al.*, 1984] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem". *IJRR*, pages 76–88, 1984.

[Khorshid *et al.*, 2011] M.M. Khorshid, R.C. Holte, and N.R. Sturtevant. A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding. In *SoCS*, pages 76–83, 2011.

[Kociemba, 1990] Herbert Kociemba. Cube explorer 5.12 HTM and QTM. http://kociemba.org/cube.htm, 1990.

[Korf, 1985] Richard Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[Korf, 1997] Richard Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *Proc. of the 14th National Conf. on Artif. Intell. and 9th Conf. on Innovative Applic. of Artif. Intell.*, pages 700–705. AAAI Press, 1997.

[Kornhauser *et al.*, 1984] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS*, pages 241–250, 1984.

[Luna and Bekris, 2011] Ryan Luna and Kostas E. Bekris. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *IJCAI*, pages 294–300, 2011.

[Peasgood *et al.*, 2008] M. Peasgood, C. Clark, and J. McPhee. A complete and scalable strategy for coordinating multiple robots. *IEEE Transactions on Robotics*, 24(2):282–292, 2008.

[Ratner and Warmuth, 1990] D. Ratner and M. Warmuth. Finding a shortest solution for the NxN-extension of the 15-puzzle is intractable. *Journal of Symbolic Computations*, 10:111–137, 1990.

[Röger and Helmert, 2012] Gabriele Röger and Malte Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In *AAAI*, 2012.

[Rokicki *et al.*, 2013] Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge. The diameter of the Rubik's cube group is twenty. *SIAM Journal of Discrete Math.*, 27(2):1082–1105, 2013.

[Rokicki, 2008] Tomas Rokicki. Twenty-five moves suffice for Rubik's cube. *CoRR*, abs/0803.3435, 2008.

[Ryan, 2007] Malcolm Ryan. Graph decomposition for efficient multi-robot cooperation. In *IJCAI*, pages 2003–2008, 2007.

[Sharon *et al.*, 2012a] G. Sharon, R. Stern, A. Felner, and N. Sturtevant. Conflict-based search for optimal multi-agent path finding. In *AAAI Conference on Artificial Intelligence*, 2012.

[Sharon *et al.*, 2012b] G. Sharon, R. Stern, A. Felner, and N. Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. In *Annual Symposium on Combinatorial Search*, 2012.

[Silver, 2006] D. Silver. Cooperative Pathfinding. *AI Programming Wisdom*, 2006.

[Standley and Korf, 2011] T.S. Standley and R. Korf. Complete Algorithms for Cooperative Pathfinding Problems. In *IJCAI*, pages 668–673, 2011.

[Standley, 2010] T.S. Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*, 2010.

[Surynek, 2009] Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 3613–3619, 2009.

[Thi, 1981] Thistlewaite's 52-move algorithm. http://www.jaapsch.net/puzzles/thistle.htm, 1981.

[van den Berg *et al.*, 2009] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Proc. of Robotics: Science and Systems*, 2009.

[Wagner and Choset, 2011] Glenn Wagner and Howie Choset. M*: A complete multi-robot path planning algorithm with performance bounds. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 3260–3267, 2011.

[Wang and Botea, 2011] Ko-Hsin Cindy Wang and Adi Botea. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research (JAIR)*, 42:55–90, 2011.

[Wilson, 1974] Richard Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory*, B(16):86–96, 1974.

[Yu and LaValle, 2012a] Jingjin Yu and Steven LaValle. Planning Optimal Paths for Multiple Agents on Graphs. arXiv:1204.3830, 2012.

[Yu and LaValle, 2012b] Jingjin Yu and Steven LaValle. Time Optimal Multi-agent Path Planning on Graphs. In *The First AAAI Workshop on Multiagent Pathfinding (WoMP)*, 2012.