

Shadoks Approach to Low-Makespan Coordinated Motion Planning

Loïc Crombez   




Université Clermont-Auvergne and LIMOS, France

Guilherme D. da Fonseca   

Aix Marseille Université and LIS, France

Yan Gerard   

Université Clermont-Auvergne and LIMOS, France

Aldo Gonzalez-Lorenzo   

Aix Marseille Université and LIS, France

Pascal Lafourcade   

Université Clermont-Auvergne and LIMOS, France

Luc Libralesso   

Université Clermont-Auvergne and LIMOS, France

Abstract

This paper describes the heuristics used by the **Shadoks**¹ team for the CG:SHOP 2021 challenge on motion planning. Using the heuristics outlined in this paper, our team won first place with the best solution to 202 out of 203 instances and optimal solutions to at least 105 of them.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Motion path planning

Keywords and phrases heuristics, motion planning, digital geometry, shortest path

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.63

Category CG Challenge

Related Version *Full Version*: <https://arxiv.org/abs/2103.13956>

Supplementary Material

Audiovisual: <https://vimeo.com/515040997>

Software (Source Code): <https://github.com/gfonsecabr/shadoks-robots>
archived at `swh:1:dir:88b0688cdc71890d6582cdfc8e8c9c0e37b831b4`

Funding *Loïc Crombez*: This work has been sponsored by the French government research program “Investissements d’Avenir” through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).

Guilherme D. da Fonseca: This work is supported by the French ANR PRC grant ADDS (ANR-19-CE48-0005).

Yan Gerard: This work is supported by the French ANR PRC grants ADDS (ANR-19-CE48-0005) and ACTIVmap (ANR-19-CE19-0005).

Pascal Lafourcade: This work is supported by the French ANR PRC grant MobiS5 (ANR-18-CE39-0019), DECRYPT (ANR-18-CE39-0007), and SEVERITAS (ANR-20-CE39-0005).

Luc Libralesso: This work is supported by the French ANR PRC grant DECRYPT (ANR-18-CE39-0007).

Acknowledgements We would like to thank H el ene Toussaint, Rapha el Amato, Boris Lonjon, and William Guyot-L enat from LIMOS, as well as the Qarma and TALEP teams and Manuel Bertrand from LIS, who continue to make the computational resources of the LIMOS and LIS clusters available to our research. We would also like to thank the challenge organizers and other competitors for their time, feedback, and making this whole event possible.

¹ The team name comes from the animated series https://en.wikipedia.org/wiki/Les_Shadoks.



  Lo c Crombez, Guilherme D. da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso;
licensed under Creative Commons License CC-BY 4.0

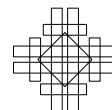
37th International Symposium on Computational Geometry (SoCG 2021).

Editors: Kevin Buchin and  ric Colin de Verdi ere; Article No. 63; pp. 63:1–63:9



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum f ur Informatik, Dagstuhl Publishing, Germany



1 Introduction

We explain some heuristics used by the **Shadoks** team to solve the CG:SHOP 2021 challenge that considers a coordinated motion planning problem in the two-dimensional grid \mathbb{Z}^2 . The goal is to move a set of n labeled unit squares called *robots* between given start and target grid cells without collisions. For more details, see the overview [3] and also [2].

The objective of the problem is to minimize the makespan² m . A trivial lower bound to the makespan is the largest obstacle-avoiding L_1 distance between the start and the target of a robot. Since the problem is symmetric with respect to the time, we may always exchange start-target positions and reverse the paths, keeping the best solution found.

The challenge CG:SHOP 2021 provided 203 instances containing between 10 and 9000 robots, out of which 202 of our solutions were the best ones among all the 17 teams who participated. To our surprise, we succeeded in finding 105 solutions that match the trivial lower bound. Our strategy consists of two steps, presented in Section 2 and 3: finding a feasible solution and reducing its makespan.

At the beginning of the challenge, we tried some approaches from the multi-agent path finding literature [4, 7] (notably CBS [5]) to solve the challenge instances. To our surprise, they did not perform well. Indeed, the challenge instances are much denser than the ones in the literature. These instances are usually sparse in the number of agents (there are from 2 to 120 agents placed in grids with over 100,000 cells in these instances, where instances of the challenge contain hundreds or thousands of agents placed in grids never larger than 100×100). This structural difference has a dramatic effect on the performance of CBS and in our experiments, CBS fails to find solutions for most of the challenge instances (with the exception of some very small ones).

2 Initial Solutions

Feasibility is guaranteed for the challenge instances since the number of obstacles is finite and every start and target are located in the unbounded region of space. In this section, we show how to obtain feasible solutions with a moderate makespan. We divide the heuristics in two categories. In Section 2.1, we compute the solution one step at a time, considering multiple robots simultaneously. In Section 2.2, we compute the solutions one robot at a time.

The heuristics of the first category are not guaranteed to find a solution, but when they do they often find solutions of lower makespan than those of the second category. The algorithms of the second category are guaranteed to find a solution, but the resulting makespan may potentially be high.

2.1 Step by Step Computation

The problem of finding a solution for coordinated motion planning in a given number of steps can be modeled as an Integer Linear Problem (ILP) or equivalently as a SAT problem (see [8, 9] and references therein). While applying such an approach is intractable even for small instances, it can be adapted to find an initial solution. The general idea of the *Greedy* solver is to plan only a small number k of steps for the robots such that the overall distance to the targets decreases as much as possible and repeat until reaching the targets.

² The challenge also considered the objective of minimizing the sum of the distances, but we did not optimize our solutions for this version.

Our ILP model considers a Boolean variable $x_{r,P}$ for each robot r and for each possible path P of length k starting at the position of the robot. Constraints of having one and only one path per robot and avoiding obstacles and collisions between robots are easily expressed as linear inequalities. The objective function we maximize is the sum of all the variables with weight

$$\text{weight}(r, P) = \left(\delta_r(p(0)) - \delta_r(p(k)) \right) \cdot \left((\delta_r(p(0)))^2 + 1 \right),$$

where $p(0)$ and $p(k)$ are the first and the last positions of path P and $\delta_r(p)$ is the obstacle-avoiding distance from a point p to the target of robot r . The first factor encourages the solution to push the robots towards their targets, since it is better to get closer to target. The second factor prioritizes moving robots that are farther from their targets and we add one so that robots that are already at their target position are encouraged to remain there.

In practice, we set $k = 3$ and we only perform the first step of the planned moves so that the robots can *anticipate* the moves of the other robots. Using the CPLEX [1] library to solve these problems, we can handle instances with up to roughly 200 robots. Note that this Greedy algorithm is not guaranteed to find a solution, and it fails to solve instances with *corridors* such as `universe_bg_000`.

2.2 Robot by Robot Computation

The algorithms in this section compute the solution one robot at a time using an A* search. The search happens in 3-dimensional space where each robot state has integer coordinates of the form (x, y, t) for position coordinates x, y and time t . There are 5 possible *movements*, all of which increase t by one unit. One movement keeps the position x, y unchanged, while the other 4 movements increment or decrement one of the two coordinates. A movement is feasible if it does not violate any of the problem constraints, considering the current path of the other robots.

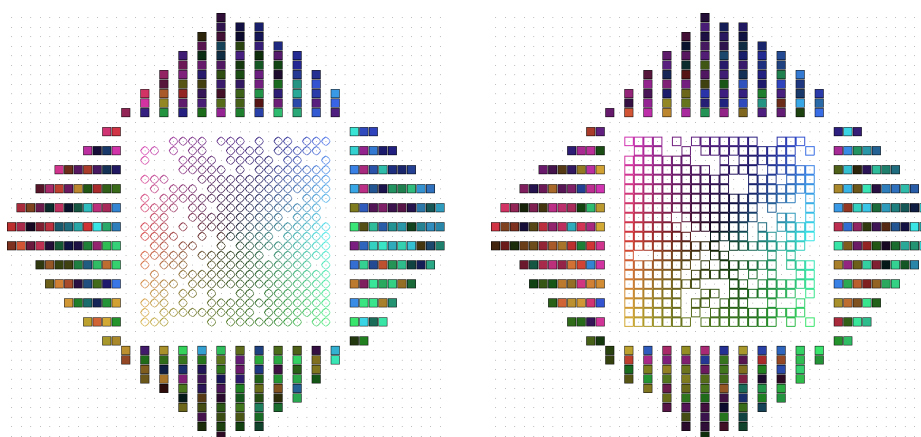
We refer to the *bounding box* as an integer axis-aligned rectangular region containing all the start, target positions and obstacles inside its strict interior (not on the boundary). Given a set of obstacles and a bounding box, the *depth* of a position p is the minimum obstacle-avoiding distance from p to a position outside the bounding box.

All algorithms in this section are based on a *storage network* N . A storage network is a set N of positions outside a predetermined bounding box such that for every position p in N , there exists a path that avoids all other positions of N and goes from p to some point in the bounding box. Each robot r_i is assigned to a distinct element of N , called the *storage* of r_i .

Initially, we set the path of each robot to be stationary at the start position. We sort the robots by *increasing start depth* and for each robot in order, we use A* search to find the shortest path from start to storage, replacing the previous stationary path. The order by which the robots are sorted guarantees that such a path exists.

After finding paths from start to storage for every robot, we proceed to the next phase of the algorithm. We now sort the robots by *decreasing target depth*. Again, the order of the robots guarantees that a path from storage to target exists. However, we do not compute such a path. Instead, we compute a path from start to target directly, whose existence is guaranteed by the existence of a path from start to storage and another one from storage to target. The following paragraphs describe the design of four different storage networks.

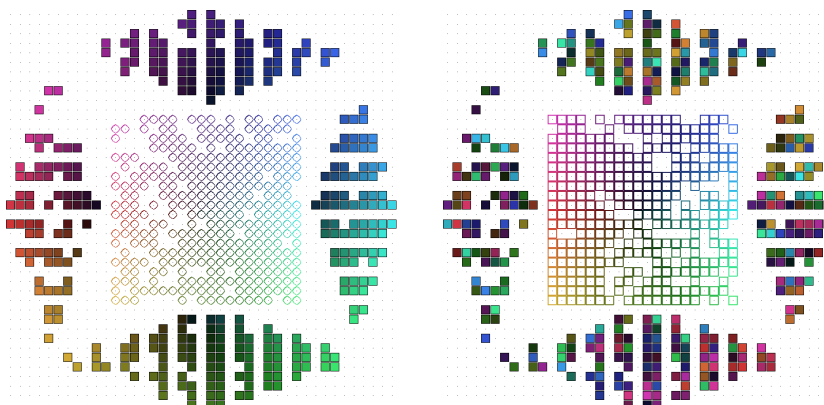
Cross. In the *Cross* strategy, we define the storage network N as the set of columns of even x coordinate lying directly above or below the bounding box and the set of rows of even y coordinate lying directly to the left or right of the bounding box, hence the name



■ **Figure 1** Cross storage network for the `small_free_016` instance colored based on start and target locations, respectively.

Cross. Then, we compute a maximal cardinality matching between the robots and N . We tried both minimum-weight matching and greedy matchings, minimizing a weight function that considers the distance from start to storage as well as the distance from storage to target. In the greedy matching version, robots are assigned a storage ordered by decreasing start-to-target distance. The result is represented in Figure 1.

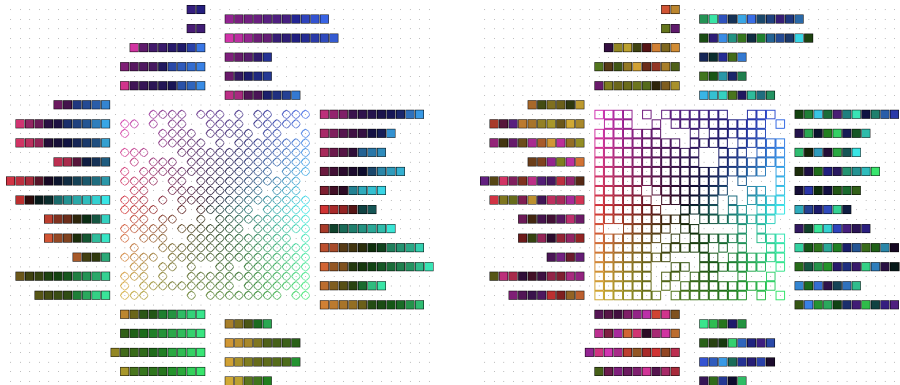
Cootie Catcher. The previous strategy works very well for small or sparse instances. However, the different directions of the flow of robots from start to storage make the solutions inefficient for large dense instances. The *Cootie Catcher* strategy computes the storage using only the start location, in order to better exploit parallel movement of the robots. The storage network shape consisting of four diamonds is presented in Figure 2. For instances without obstacles, the strategy is guaranteed to find a path from start to storage using at most $w/2 + \mathcal{O}(1)$ steps, where w is the largest bounding box side. Surprisingly, this strategy also works well for many instances with obstacles.



■ **Figure 2** Cootie Catcher storage network for the `small_free_016` instance colored based on start and target locations, respectively.

Dichotomy. The weakness of the previous method is that robots may be assigned storage in a location that is opposite to the direction from start to target. In order to exploit parallel movements while taking the target location into consideration, we developed the *Dichotomy* strategy. The strategy only works for instances without obstacles.

We translate the coordinate system so that the origin is the center of the bounding box. The robots are partitioned into two sets called *left side* and *right side* according to the sign of the target location's x -coordinate. Left-side robots are assigned storage with positive x -coordinate while right-side robots are assigned storage with negative x -coordinate, as represented in Figure 3.



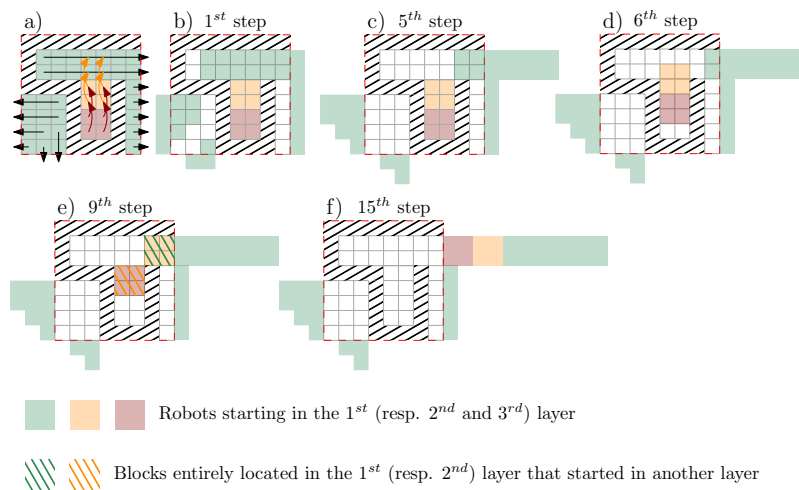
■ **Figure 3** Dichotomy storage network for the `small_free_016` instance colored based on start and target locations, respectively.

Escape. This strategy focuses on instances with obstacles, especially on dense instances where the obstacles create bottlenecks. The goal of the *Escape* strategy is to clear the bounding box as quickly as possible. To do so, we move the robots by blocks as large as possible making efficient use of parallel movements. The *Escape* strategy defines layers inside the bounding box. Robots located in each layer will move in a straight line to reach the previous layer and then in another straight line outside of the bounding box as represented in Figure 4. We used a naive algorithm to define the layers, and partially redefined them by hand for the most complicated instances and the unsatisfying results.

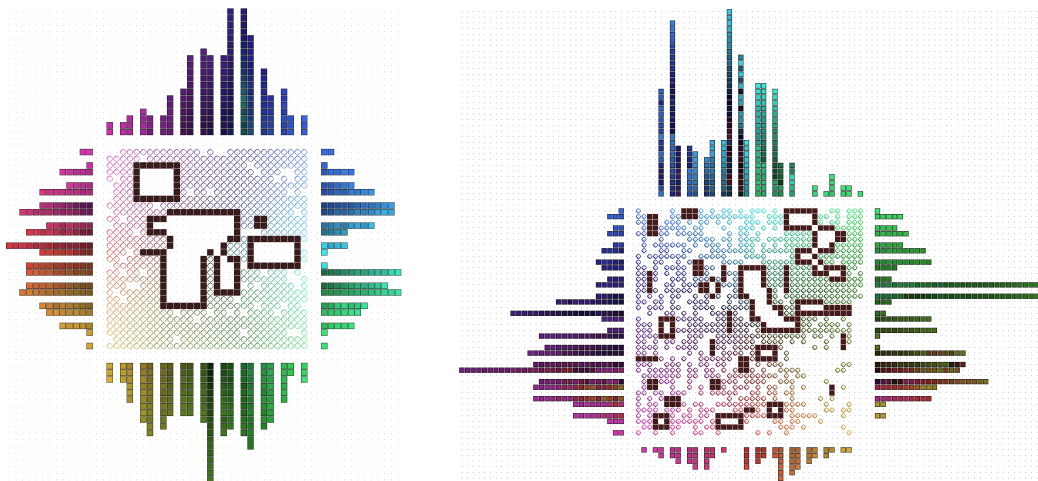
3 Improving Solutions

In this section we discuss the two heuristics that we used to reduce the makespan of a given feasible solution. The first heuristic makes local changes to the solution, which remains feasible throughout the process, and possibly reduces the makespan. The second heuristic destroys the feasibility of the solution and either finds another solution of reduced makespan, or no feasible solution at all. Throughout, let m be the makespan of the input solution.

Feasible Optimizer. The idea of the *Feasible Optimizer* is the following. We iteratively remove the path of a robot r from the solution, and then use the A* algorithm to find a new (hopefully different) path for r . The A* algorithm may be tuned in several ways to produce different paths, and we do so in such a way that the makespan of the solution never increases and also that a robot is only allowed to move at time m if it already did so in the original path. This way, not only the makespan but also the number of robots moving at time m never increase. Next, we list some examples on how to modify the A* search.



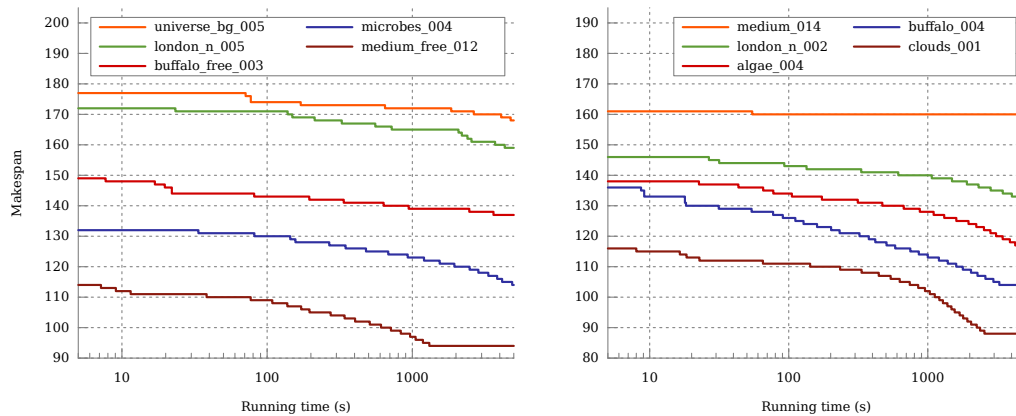
■ **Figure 4** Escape strategy.



■ **Figure 5** Escape storage network for the `medium_007` and `buffalo_003` instances, respectively.

- Find the path from start to target that reaches the target as quickly as possible but break ties using the sum of random weights given to each grid cell the robot passes through.
- Reversing the direction of time and then finding a path from target to start that reaches the start as quickly as possible. In the original time direction, that means that the robot will remain at the start for as long as possible.
- In the reversed case, force the robot to stay at target for a certain number of steps.

Conflict Optimizer. The previous optimization strategy may take very long to reduce the makespan. Next, we describe a more aggressive approach that leaves the feasible solution space and works far better than we expected. The algorithm uses a modified A* search that allows for a robot to go over another robot's path, which we call a *conflict*. We start by creating a *queue* with all the robots that move at makespan time m . While the queue is not empty, we repeat the following procedure for a robot r popped from the front of the queue. Let $q(r)$ be the number of times a robot r has been popped out of the queue. We define the *weight* of a robot r as $1 + (q(r))^2$.



■ **Figure 6** Improved makespan over computation time using the Conflict Optimizer.

1. Erase r 's path.
2. Find a path for r from start to target that arrives no later than time $m - 1$ and minimizes the weighted sum of conflicting robots.
3. Add all conflicting robots to the queue.

For sparse instances, the Conflict Optimizer can even be used to compute solutions from scratch by choosing an initial makespan and putting all the robots in the queue.

4 Results

Tables 1 and 2 show the makespan obtained using different heuristics on some selected challenge instances and the makespan lower bound. Figure 6 shows the improvement obtained by the Conflict Optimizer over a little more than one hour of execution.

The two other teams UNIST [10] and *gitastrophe* [6] on the podium of the CG:SHOP 2021 challenge used similar strategies, also computing an initial solution through storage networks. The methods used to optimize a solution are somewhat similar to our Feasible Optimizer, plus simulated annealing for UNIST and optimization of samples of k robots chosen according to their makespan, distance, or conflicts for *gitastrophe*. None of them used an optimization algorithm with a dynamic queue of robots as our Conflict Optimizer.

■ **Table 1** Makespan of different heuristics for selected instances without obstacles.

| instance | n | w | initial solution | | | | optimizer | | lower bound |
|------------------|------|-----|------------------|------------|-------|------------|-----------|------------|-------------|
| | | | Gree. | Cross | Coot. | Dich. | Feas. | Conf. | |
| small_free_002 | 40 | 10 | 17 | 22 | 27 | 22 | 17 | 15 | 15 |
| small_free_003 | 70 | 10 | 20 | 31 | 27 | 26 | 20 | 16 | 14 |
| small_free_010 | 200 | 20 | 34 | 46 | 54 | 45 | 33 | 32 | 32 |
| small_free_015 | 280 | 20 | . | 60 | 68 | 65 | 51 | 40 | 32 |
| small_free_016 | 320 | 20 | 63 | 68 | 77 | 68 | 60 | 47 | 36 |
| medium_free_007 | 630 | 30 | 148 | 89 | 103 | 95 | 81 | 60 | 52 |
| medium_free_009 | 800 | 40 | 93 | 97 | 124 | 109 | 81 | 71 | 71 |
| medium_free_012 | 1000 | 50 | . | 114 | 125 | 127 | 96 | 94 | 94 |
| microbes_004 | 1250 | 50 | . | 132 | 159 | 135 | 125 | 91 | 91 |
| buffalo_free_003 | 1440 | 60 | . | 149 | 165 | 158 | 125 | 87 | 78 |
| london_night_005 | 1875 | 50 | . | 179 | 190 | 173 | 157 | 124 | 92 |
| universe_bg_005 | 2000 | 50 | . | 194 | 198 | 177 | 173 | 141 | 82 |
| galaxy_c2_008 | 3000 | 100 | . | 198 | 258 | 234 | 168 | 163 | 163 |
| large_free_004 | 3938 | 75 | . | 274 | 276 | 256 | 240 | 204 | 127 |
| large_free_005 | 5000 | 100 | . | 260 | 316 | 293 | 252 | 184 | 184 |
| large_free_007 | 6000 | 100 | . | 297 | 343 | 325 | 295 | 236 | 189 |
| sun_009 | 7500 | 100 | . | 424 | 395 | 361 | 354 | 345 | 187 |
| large_free_009 | 9000 | 100 | . | 514 | 440 | 391 | 378 | 374 | 182 |

■ **Table 2** Makespan of different heuristics for selected instances with obstacles.

| instance | n | w | initial solution | | | | optimizer | | lower bound |
|------------------|------|-----|------------------|------------|------------|------------|------------|------------|-------------|
| | | | Gree. | Cross | Coot. | Esca. | Feas. | Conf. | |
| small_005 | 63 | 10 | 27 | 28 | 32 | 37 | 25 | 20 | 18 |
| sun_000 | 143 | 20 | 32 | 39 | 46 | 61 | 29 | 27 | 27 |
| small_011 | 183 | 20 | 56 | 60 | 70 | 67 | 48 | 40 | 37 |
| small_016 | 276 | 20 | . | 67 | 72 | 79 | 57 | 43 | 36 |
| medium_005 | 407 | 30 | . | 119 | 110 | 106 | 94 | 74 | 58 |
| london_night_002 | 825 | 50 | . | 149 | 162 | 165 | 142 | 94 | 84 |
| microbes_002 | 958 | 50 | . | 111 | 135 | 173 | 97 | 89 | 89 |
| clouds_001 | 912 | 50 | . | 117 | 138 | 159 | 94 | 83 | 83 |
| medium_014 | 1165 | 40 | . | 180 | 161 | 180 | 161 | 151 | 73 |
| algae_004 | 1113 | 50 | . | 139 | 160 | 191 | 121 | 84 | 79 |
| buffalo_004 | 1404 | 60 | . | 136 | 164 | 195 | 120 | 104 | 104 |
| large_003 | 1906 | 100 | . | 172 | 224 | 250 | 154 | 154 | 154 |
| large_004 | 2034 | 100 | . | 431 | 391 | 381 | . | 381 | 185 |
| large_005 | 3223 | 75 | . | 398 | 310 | 317 | . | 299 | 141 |
| universe_bg_007 | 3820 | 100 | . | 224 | 289 | 323 | 202 | 184 | 184 |
| large_007 | 4706 | 100 | . | 753 | 497 | 491 | 497 | 471 | 215 |
| microbes_008 | 5643 | 100 | . | 329 | 359 | 425 | 322 | 279 | 188 |
| algae_009 | 7311 | 100 | . | 500 | 439 | 441 | . | 421 | 176 |
| large_009 | 8595 | 100 | . | 398 | 387 | 566 | . | 352 | 176 |

References

- 1 IBM ILOG Cplex. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.
- 2 Sándor P. Demaine, Erik D. and Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.
- 3 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The cg:shop challenge 2021, 2021. arXiv:2103.15381.
- 4 Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*, 2017.
- 5 Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015. doi:10.1016/j.artint.2014.11.006.
- 6 Jack Spalding-Jamieson, Paul Liu, Brandon Zhang, and Da Wei Zheng. Coordinated motion through randomized k-opt. In *37th International Symposium on Computational Geometry, SoCG 2021*, volume 189 of *LIPICs*, pages 64:1–64:8, 2021. doi:10.4230/LIPICs.SoCG.2021.64.
- 7 Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. *arXiv preprint*, 2019. arXiv:1906.08291.
- 8 Pavel Surynek. Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In *International Joint Conferences on Artificial Intelligence*, pages 1177–1183, 2019.
- 9 Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *22nd European Conference on Artificial Intelligence, ECAI 2016*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 810–818, 2016. doi:10.3233/978-1-61499-672-9-810.
- 10 Hyeyun Yang and Antoine Vigneron. A simulated annealing approach to coordinated motion planning. In *37th International Symposium on Computational Geometry, SoCG 2021*, volume 189 of *LIPICs*, pages 65:1–65:9, 2021. doi:10.4230/LIPICs.SoCG.2021.65.