A Simulated Annealing Approach to Coordinated Motion Planning

Hyeyun Yang
Ulsan National Institute of Science and Technology, South Korea

Antoine Vigneron[1]
Ulsan National Institute of Science and Technology, South Korea

Abstract

The third computational geometry challenge was on a coordinated motion planning problem in which a collection of square robots need to move on the integer grid, from their given starting points to their target points, and without collision between robots, or between robots and a set of input obstacles. We designed and implemented an algorithm for this problem, which consists of three parts. First, we computed a feasible solution by placing middle-points outside of the minimum bounding box of the input positions of the robots and the obstacles, and moving each robot from its starting point to its target point through a middle-point. Second, we applied a simple local search approach where we repeatedly delete and insert again a random robot through an optimal path. It improves the quality of the solution, as the robots no longer need to go through the middle-points. Finally, we used simulated annealing to further improve this feasible solution. We used two different types of moves: We either tightened the whole trajectory of a robot, or we stretched it between two points by making the robot move through a third intermediate point generated at random.

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

Keywords and phrases Path planning, simulated annealing, local search

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.65

Category CG Challenge

Funding This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2017R1D1A1B04036529).

1 Introduction

In this paper, we present our solution to the third computational geometry challenge, which was on a coordinated motion planning problem [2, 3]. We first briefly describe this problem. A set of N robots, modeled as unit squares, need to move on the integer grid $\mathbb{Z}^2$, from their starting positions $s_1, \dots, s_N \in \mathbb{Z}^2$ to their target positions $d_1, \dots, d_N \in \mathbb{Z}^2$. A (possibly empty) set $\mathcal{O} \subset \mathbb{Z}^2$ of obstacles is also given. We denote by $p_i(t) = (x_i(t), y_i(t)) \in \mathbb{Z}^2$ the position of robot i at time $t \in \mathbb{N}$. At each time $t \in \mathbb{N}$, the robot may either stay at the same position, or move to one of the four neighboring squares, hence we have $p_i(t+1) - p_i(t) \in \{(0,0), (-1,0), (0,-1), (0,1), (1,0)\}$.

While moving, each robot must avoid collision with obstacles or other robots. In particular, for any robot i and any time $t \in \mathbb{N}$, we must have $p_i(t) \notin \mathcal{O}$ and $p_i(t) \neq p_j(t)$ for all $j \neq i$. In addition, a constraint is enforced in order to model coordinated movement: a robot i can only move to the position previously occupied by another robot j if they move in the same direction. More precisely, if $p_i(t+1) = p_j(t)$, then we must have $p_i(t+1) - p_i(t) = p_j(t+1) - p_j(t)$.

[1] Corresponding author.

The *length* of the trajectory of robot i is the number of times robot i moves, in other words it is $|\{t \in \mathbb{N} \mid p_i(t+1) \neq p_i(t)\}|$. Its *completion time* $t_C(i)$ is the time when robot i ceases to move; in other words, it is the minimum time $t_C(i)$ such that for all $t \geq t_C(i)$, we have $p_i(t) = d_i$. The *makespan* is the maximum completion time.

The solutions to this coordinated motion planning problem were scored according to two criteria: we should either minimize the makespan (MAX), or minimize the sum of the lengths of the paths (SUM). Our team (UNIST) ranked second according to both criteria, out of 17 teams participating in the contest. Team Shadoks [1] ranked first according to MAX and third according to SUM, while team gitastrophe [4] ranked first according to SUM and third according to MAX. More information on the contest can be found in the survey by Fekete et al. [3]

2 Data structure

Our data structure consists of two 3-dimensional arrays G and H. The array G is a 3D-array of 8-bit integers, where $G[x, y, t] = -1$ if (x, y) is an obstacle, $G[x, y, t] = 0$ if the cell (x, y) is empty at time t, and if $(x, y) = p_i(t)$ for some i, then $G[x, y, t]$ records $p_i(t+1) - p_i(t)$. In other words, if robot i is located at (x, y) at time t, then $G[x, y, t]$ records the direction taken by robot i. We encode this direction as an integer in $\{1, \dots, 5\}$, where 5 means that robot i does not move. (In particular, G and H do not record robot numbers.) The array H is a 3D-array of 16-bit integers, which is only needed in the SUM version.

We chose this data structure in order to minimize memory usage, as we feared that large instances would not fit in RAM otherwise. It turns out that we only used a small percentage of our RAM even for the largest instances (less than 7%), so we could have afforded a larger data structure.

For the sake of analyzing our algorithms, we use w to denote the size of G in either dimension. The reason is that, for the problem instances given in the contest, the makespan of the solutions we constructed was not much larger than the length and width of the grid we used – less than a factor 10. So we may assume that G is a $w \times w \times w$-array. Our data structure allows us to do the following.

Deletion. We can delete the trajectory of robot i from G in $O(w)$ time. It suffices to follow the direction given by $G[x, y, t]$ from the starting point s_i.

Insertion. Assuming robot i is not yet recorded in G, and there is a feasible path from s_i to d_i, we can insert in $O(w^3)$ time a feasible trajectory of robot i that either minimizes the completion time $C_p(i)$, or minimizes its length. It can be done by a simple sweep of G by increasing values of t, and storing in $G[x, y, t]$ the direction of the move from the parent (if any) of (x, y, t) as an integer in $\{11, \dots, 15\}$. In the SUM version, we also record in $H[x, y, t]$ the length of the shortest feasible trajectory to (x, y, t). At the end of the sweep, we reconstruct the path backwards, and we remove all values larger than 10 from G.

3 Computing a feasible solution

In order to compute a feasible solution, we construct a middle-point m_i for each robot i, such that there is a feasible path from s_i to d_i through m_i. (See Figure 1a.) Let B be the minimum bounding box of the starting points, target points and obstacles. The middle-points are chosen from the set of grid points outside B, not adjacent to B, and whose x and y-coordinates are even.

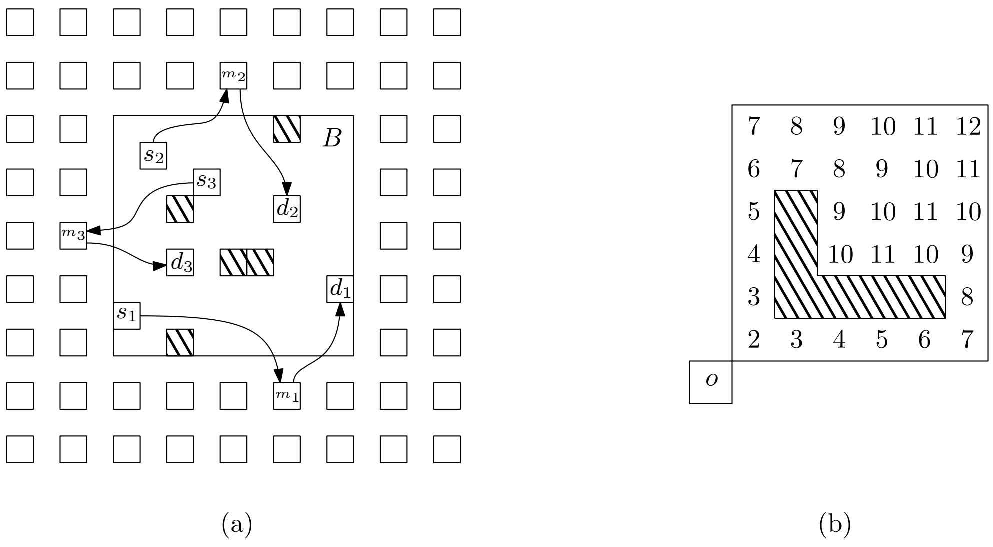

Figure 1 (a) Robot i goes from s_i to d_i through m_i. (b) The geodesic distance from o.

In a first stage, we move all the robots to their middle-points, and in a second stage we move them from their middle-points to their target points. In order to do this, we insert the robots one by one in our 3D array G, by applying the insertion procedure from Section 2.

One difficulty here is that a robot can be blocked by other robots that are still at their starting position, so we need to move the robots in an appropriate order. To this end, we compute the geodesic distances from an arbitrary point o outside B. (See Figure 1b.) We sort the robots by *increasing* geodesic distance, and insert them in this order. This ensures that there is always a feasible path from s_i to m_i for each i. In the second stage, we proceed in the same way, except that we proceed by *decreasing* geodesic distance from o to the target points. This approach finds a feasible solution to all the input instances from the contest, as all robots are in the unbounded face of $\mathbb{Z}^2 \setminus \mathcal{O}$.

For each robot i, we have several choices for the middle-point m_i. We tried a few possibilities. The one that most often gives the best results was to choose the available middle-point that minimizes the sum of the Manhattan distances to s_i and d_i.

As we run the deletion and insertion procedures $2w$ times, we obtain a feasible solution in $O(w^4)$ time. We were able to compute a feasible solution for each instance in less than 2 hours.

4 Simple local search

In order to improve the feasible solution from Section 3, we can simply pick a robot i, delete it from G and insert it again using the procedures from Section 2. It may give a quite large improvement for this robot, as it no longer has to go through its middle-point m_i. We call this operation a *tightening* move. (See Figure 2.)

A first approach is to repeatedly tighten the path of a robot chosen at random. We implemented it in a slightly different way, which produced better results. We first compute a random permutation of the robots. Then we go through this random list, and perform a tightening operation on the current robot. If, at the end of the list, no improvement was made, we return the current solution. Otherwise, we compute a new random permutation and repeat the process with the new permutation. The procedure above can be restarted from the original feasible solution several times, as it does not always produce the same solution.

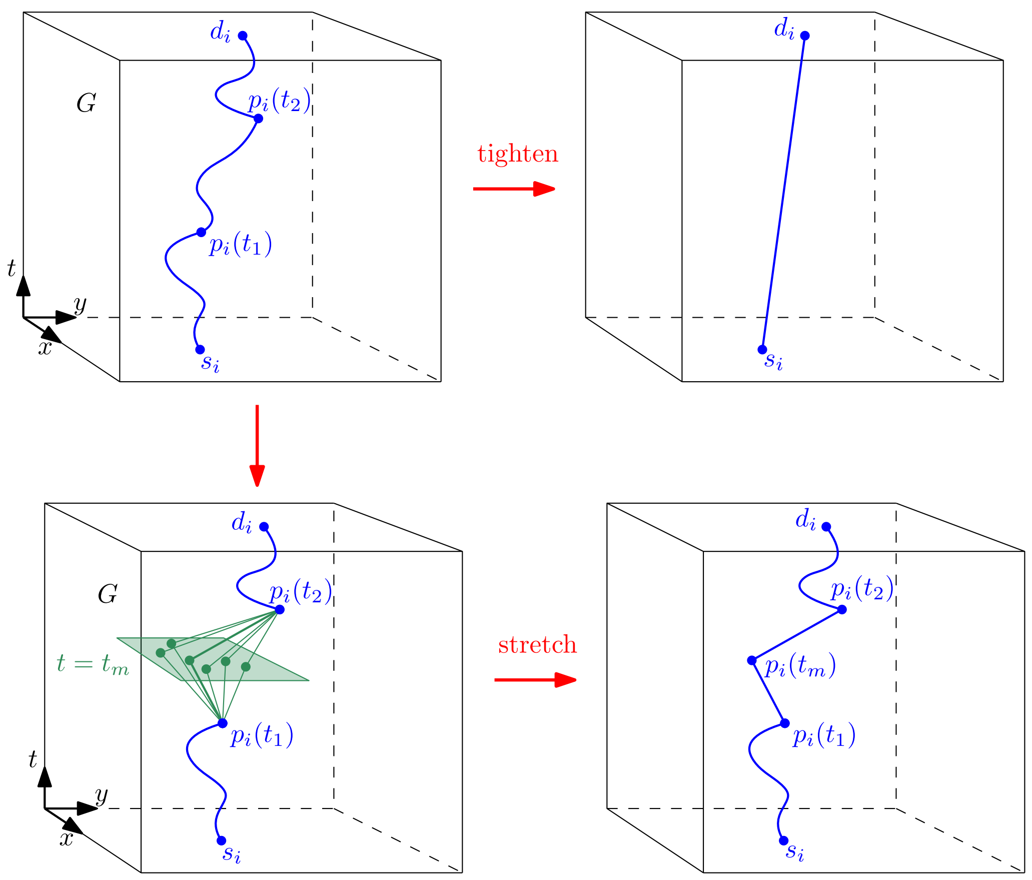

Figure 2 Tightening and stretching moves.

5 Simulated annealing

One drawback of the local search approach described in Section 4 is that the only type of moves that is allowed consists in shortening the whole trajectory of one robot, and thus it can easily be trapped in a local minimum. To remedy this, we used a simulated annealing approach [5] that uses two types of moves: tightening moves as described in Section 4, and *stretching* moves, which can make a trajectory longer.

We now describe stretching moves. (See Figure 2.) As mentioned above, we assume that G is a $w \times w \times w$ array. Let i be a robot, and let t_1, t_2 be integers such that $0 \leq t_1 \leq t_2 < n$. We will see later how we generate t_1 and t_2 at random. We first delete the trajectory of i between $p_i(t_1)$ and $p_i(t_2)$. Let t_m be chosen uniformly at random between t_1 and t_2. We compute all of the cells (x, y, t_m) of G that are reachable from $p_i(t_1)$, and that are reachable backwards from $p_i(t_2)$. We pick one of these points uniformly at random, which we denote by q. Then we connect q to $p_i(t_1)$ and $p_i(t_2)$ through shortest paths, computed in the same way as we did in the insertion operation from Section 2.

Let $\delta = t_2 - t_1$. As we only need to sweep a sub-array of size at most $2\delta \times 2\delta \times \delta$, and we need linear time to find $p_i(t_1)$ and $p_i(t_2)$, the stretching operation takes time $O(w + \delta^3)$.

We now explain how we generate t_1 and t_2. We first set $\delta = \min(w - 1, \lfloor \alpha\sqrt{1/x} \rfloor)$, where α is a constant larger than 2, and x is a random floating point number in $(0, 1]$. Then we generate t_1 as a random number chosen uniformly between 0 and $n - 1 - \delta$, and we set $t_2 = t_1 + \delta$. This approach ensures that, for $k < n - 1$, we have $\Pr(\delta = k) = \Theta(1/k^3)$, and thus $E[\delta^3] = \Theta(w)$. It follows that our stretching operation takes $O(w)$ expected time.

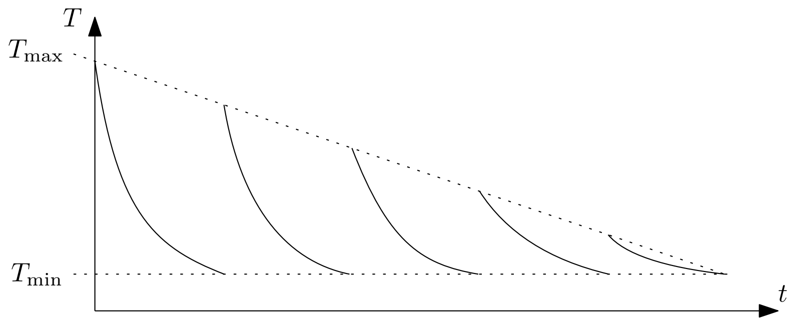

Figure 3 A cooling schedule with $n_{\text{cycle}} = 5$.

We use as the objective function a score that is either the makespan, or the sum of the lengths of the paths divided by the number of robots. A move that improves the score is always accepted by the algorithm, but a move that increases it is accepted with probability $\exp(-\Delta/T)$, where T is the current temperature and Δ is the score increase. We used a cooling schedule consisting of n_{cycle} cycles of n_{iter} iterations each, such that the temperature decreases exponentially within each cycle, and the maximum temperature decreases linearly. (See Figure 3.) We did not try other cooling schedules, but we tried different number of cycles and iterations per cycle.

Choice of parameters. We determined the values of the various parameters of the algorithm by trying it on several instances, varying the parameters and comparing the results.

We found that $\alpha = 5$ did better than other values on most instances. Regarding the choice of the moves, we generated a tightening move or a stretching move with the same probability 1/2, which implies that the algorithm is spending much more time on tightening. For the SUM version, in some cases, we obtained better results by generating a tightening move with probability 1/1 000, which means that the algorithm spends roughly the same amount of time on stretching and tightening.

We chose the minimum temperature to be $T_{\min} = 0.0001$. The maximum $T_{\max}$ was between 0.03 and 1 for the MAX version, and between 0.001 and 0.02 for the SUM version. The number of cycles was between 500 and 5 000. The number of iterations per cycle (typically 10 000-1 000 000) was adjusted depending on the time we wanted the computation to run for.

We tried different sets of moves; for instance, we tried to do tightening moves in a time window $[t_1, t_2]$ instead of $[0, n]$. We also tried to use a smoothed objective function, using a linear combination of the makespan and the sum of the lengths. Unfortunately, it did not seem to help.

6 Experimental results

We implemented the algorithms above in C++ and ran them on a server (4 Intel Xeon E5-2695 V4, 2.1 GHz, 72 cores in total). We denote by F, LS and SA the algorithm that generates the initial feasible solution (Section 3), the simple local search algorithm (Section 4) and the simulated annealing algorithm (Section 5), respectively.

We computed a feasible solution for each instance using F, which has two versions: one for SUM and one for MAX. Then we ran LS on each of these feasible solutions. After this, we ran SA on the solution given by LS, which is the approach denoted by LS+SA. Alternatively, we ran SA directly from the solution given by F.

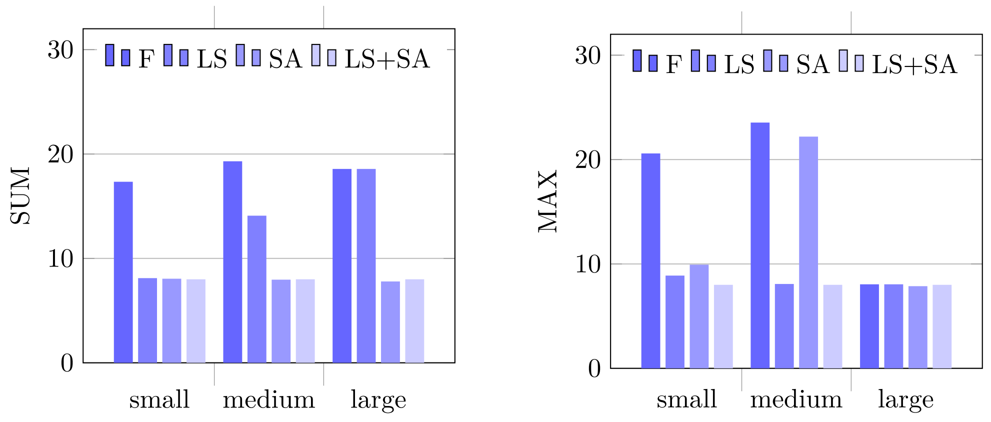

Figure 4 Performance of our algorithms on 8 small, 8 medium and 8 large-size instances. Results are normalized according to the score for LS+SA.

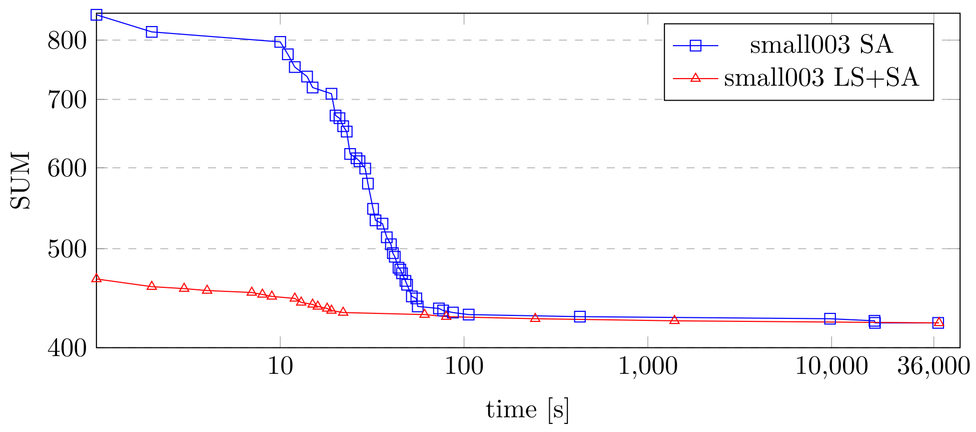

Figure 5 Best sum of lengths (SUM) until time t for the instance small_003_10x10_90_46.

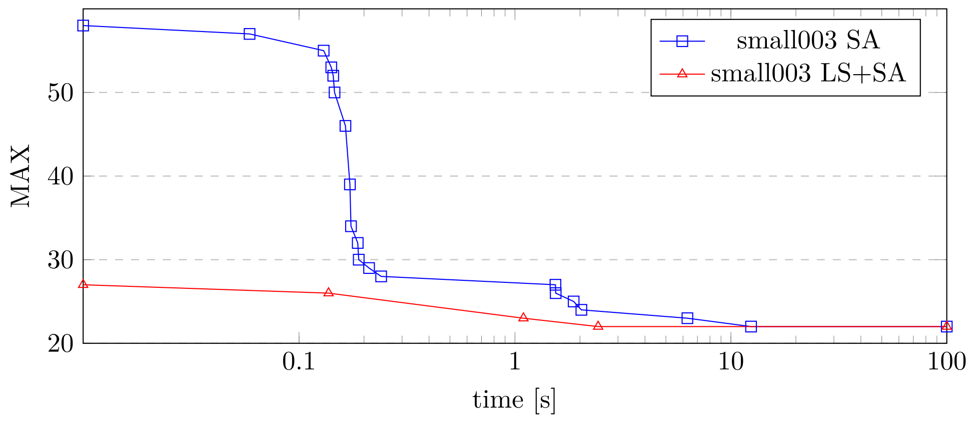

Figure 6 Best makespan (MAX) until time t for the instance small_003_10x10_90_46.

Figure 4 shows the results on 24 instances grouped into small, medium and large size, after running either SA for 8 days, or LS+SA for a total of 8 days (4 days LS, 4 days SA). Table 1 and Table 2 record the related data. Figure 5 and Figure 6 show the score improvement over time for one instance.

These results suggest that we could run SA directly after F, instead of running LS in between. One point of using LS during the contest was that it does not use any parameter, so we could run it until completion, or for large instances, stop it after a few day. Figure 4 shows that SA only does clearly worse than LS+SA for the MAX version of medium-size instances, which suggest that we may not yet have found the best parameters in this case.

Table 1 A few instances with the scores obtained after applying our algorithms (SUM).

instance name	nb. of robots	normalized score (SUM) F	LS	SA	LS+SA	score (SUM) LS+SA
small instances						
medium_free_004	480	2.86	1.03	1.02	1	11300
redblue_00002	669	1.99	1.01	1	1	25083
galaxy_c_00005	750	1.97	1.01	1	1	27107
galaxy_c2_00005	860	1.96	1.01	1	1	37413
microbes_00002	958	2.22	1.01	1	1	34754
large_001	1563	2.04	1	1	1	79793
medium_018	1993	2.32	1	1	1	98984
microbes_00006	2500	1.95	1	1	1	169912
sum		17.31	8.07	8.02	8	
medium-size instances						
universe_bg_00006	3000	2.08	1	1	1	217456
sun_00006	3796	2.38	1	0.99	1	268569
algae_00007	4000	2.33	1	1.00	1	275157
redblue_00009	4500	2.40	1	0.99	1	322346
galaxy_c_00008	5000	2.54	2.54	0.99	1	356037
london_night_00008	5648	2.44	2.44	0.98	1	444974
london_night_00009	6000	2.58	2.58	0.98	1	462928
microbes_00009	6000	2.50	2.50	0.99	1	468081
sum		19.25	14.06	7.92	8	
large instances						
clouds_00009	7229	2.33	2.33	0.97	1	641904
algae_00009	7311	2.39	2.39	0.97	1	648026
sun_00009	7500	2.42	2.42	0.97	1	675357
galaxy_c2_00009	7555	2.32	2.32	0.97	1	694166
galaxy_c_00009	7838	2.31	2.31	0.97	1	740212
universe_bg_00009	8000	2.29	2.29	0.97	1	755800
large_009	8595	2.22	2.22	0.96	1	876814
large_free_009	9000	2.25	2.25	0.96	1	911623
sum		18.53	18.53	7.74	8	

Table 2 A few instances with the scores obtained after applying our algorithms (MAX).

instance name	nb. of robots	normalized score (MAX)				score (MAX)
		F	LS	SA	LS+SA	LS+SA
small instances						
medium_free_004	480	3	1.2	0.98	1	60
redblue_00002	669	2.35	1.05	0.95	1	91
galaxy_c_00005	750	2.22	1.1	1.09	1	87
galaxy_c2_00005	860	2.67	1.19	1	1	105
microbes_00002	958	3.01	1.11	1.02	1	90
large_001	1563	2.29	1.01	1.94	1	137
medium_018	1993	2.87	1.2	0.98	1	190
microbes_00006	2500	2.14	1	1.94	1	179
sum		20.58	8.88	9.93	8	
medium-size instances						
universe_bg_00006	3000	2.25	1.02	2.17	1	198
sun_00006	3796	2.69	1	2.54	1	214
algae_00007	4000	2.92	1	2.78	1	206
redblue_00009	4500	2.74	1.02	2.62	1	221
galaxy_c_00008	5000	2.62	1	2.55	1	265
london_night_00008	5648	3.47	1.01	2.88	1	318
london_night_00009	6000	3.52	1	3.45	1	292
microbes_00009	6000	3.3	1.01	3.17	1	287
sum		23.54	8.08	22.19	8	
large instances						
clouds_00009	7229	1	1	0.98	1	1099
algae_00009	7311	1	1	0.98	1	1195
sun_00009	7500	1	1	0.94	1	1094
galaxy_c2_00009	7555	1.01	1.01	0.99	1	1095
galaxy_c_00009	7838	1	1	0.98	1	1261
universe_bg_00009	8000	1	1	0.99	1	1132
large_009	8595	1	1	0.99	1	1348
large_free_009	9000	1	1	0.98	1	1334
sum		8.05	8.05	7.87	8	

References

1 Loïc Crombez, Guilherme D. da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso. Shadoks approach to low-makespan coordinated motion planning. In *37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *LIPIcs*, pages 61:1–61:9, 2021. doi:10.4230/LIPIcs.SoCG.2021.61.

2 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM J. Comput.*, 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 Jack Spalding-Jamieson, Paul Liu, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k-opt. In *proc. 37th International Symposium on Computational Geometry*, volume 189 of *LIPIcs*, pages 64:1–64:8, 2021. `doi:10.4230/LIPIcs.SoCG.2021.64`.

5 Peter van Laarhoven and Emile Aarts. *Simulated Annealing: Theory and Applications*. Springer Netherlands, 1987.