Coordinated Motion Planning Through Randomized k-Opt

PAUL LIU, Stanford University, USA
JACK SPALDING-JAMIESON, University of Waterloo, Canada
BRANDON ZHANG, No institution, Canada
DA WEI ZHENG, University of Illinois at Urbana-Champaign, USA

This paper examines the approach taken by team `gitastrophe` in the CG:SHOP 2021 challenge. The challenge was to find a sequence of simultaneous moves of square robots between two given configurations that minimized either total distance travelled or makespan (total time). Our winning approach has two main components: an initialization phase that finds a good initial solution, and a k-opt local search phase that seeks to optimize this solution. This led to a first place finish in the distance category and a third place finish in the makespan category.

CCS Concepts: • **Computing methodologies** → **Motion path planning**; • **Theory of computation** → **Computational geometry**.

Additional Key Words and Phrases: motion planning, randomized local search, path finding

1 INTRODUCTION

For a set of unit square robots $\mathcal{R}$ each with start and target locations $\{\text{source}_r\}_{r\in\mathcal{R}}$, $\{\text{target}_r\}_{r\in\mathcal{R}} \subset \mathbb{Z}^2$, and a set of unit square obstacles $O \subset \mathbb{Z}^2$, the *coordinated motion planning* problem asks for an optimal sequence of simultaneous "moves" for each robot that brings the robots from their start location to their target location. At each timestep, robots can move to an adjacent grid cell or stand still (a no-op). These moves are subject to the following constraints at all timesteps:

- No robot shares a location with another robot or an obstacle.
- No robot moves into a location previously occupied by another robot, unless the other robot is moving in the same direction at that timestep.

Manuscript submitted to ACM

An *optimal* sequence of moves can refer to two different criteria:

- **MAX:** The minimum **makespan**, i.e. the total number of timesteps needed.
- **SUM:** The minimum **total distance**, i.e. the total number of position-changing moves each robot takes.

We, team `gitastrophe`, explored this problem as a part of the 2021 Computational Geometry Challenge (CG:SHOP 2021). The challenge ranked teams according to each of the two different optimality criteria. Our team ranked first among all junior teams, first according to the total distance criterion, and third according to the makespan criterion. Team Shadoks [1] ranked first according to makespan and third according to total distance, while Team UNIST [7] ranked second in both categories.

One important property of the test instances of the challenge is that feasibility is always guaranteed — no obstacles enclose the starting or target positions of any robots. In the following sections, we assume this property. We refer the reader to the survey of the challenge [2] for more details on the instances.

2 PRIOR WORK

The problem examined in this challenge is a specific type of *multi-agent path finding*, an area which has had a plethora of work in the past few decades [6]. Despite this, we found that existing methods were ineffective for this challenge. The main impetus seems to be the large number of agents involved in the challenge and the high spatial density of the agents on the grid. Since these methods do not exploit the feasibility guarantees of the challenge (i.e. the robots are not enclosed within any boundaries), the methods we tried were typically unable to find any feasible solution at all, as we describe below.

For the general problem of multi-agent path finding, variants of conflict-based search (CBS) have achieved state-of-the-art results across many domains [3–5]. For this challenge, our implementation of CBS was able to find optimal solutions for instances with at most a few dozen robots. Beyond that however, CBS was unable to find any solutions in a reasonable amount of time. Notably, this has also been observed by other teams participating in the challenge [1].

For the more constrained problem of moving of square-shaped robots on a grid, related works have examined generalizations of the popular “15-puzzle" problem. Using an ILP-based approach Yu and LaValle [8, 9] were able to solve instances on the order of a few hundred robots. We attempted similar ILP-based models, modelling the path of each robot by binary variables $x_{r,\ell,t,d}$ indicating if robot r moved through grid location ℓ at time t in direction d. The constraints of the problem can then be encoded as linear inequalities. For the problem of minimizing distance, the objective seeks to minimize the sum of the $x_{r,\ell,t,d}$ variables. For the problem of minimizing makespan, we enforce

a bound on the maximum t and check if the ILP is feasible. We also attempted to use the original code of [9] to check if their methods scale to the instances of this challenge. Unfortunately, we were unable to solve dense instances of more than 100 robots in either case.

3 OUR APPROACH

As with other teams who took part in the challenge, our approach consisted of two phases: initialization (Section 3.1) and solution optimization (Section 3.2).

The initialization phase consisted of moving the robots out of the initial problem grid, into an intermediate state that can easily be routed to their end goals. This strategy was remarkably similar to the feasibility approaches used by both Team Shadoks [1] and Team UNIST [7]. The solution optimization phase used a novel local search strategy, which consisted of locally reordering the movements of k sampled robots while keeping all others fixed.

We note that our approach can be used in tandem with the approaches of the other teams for better results. To illustrate this, we combine our approach with the conflict optimizer of Team Shadoks [1] in Section 4.

3.1 Initialization strategy

Our initialization strategy is simple and relies on the notion of *depth* values (see Definition 3.1). The depth values partition the robots into subsets that can be routed in parallel. Intuitively, once a robot is routed, it will not interfere with robots of smaller depth value.

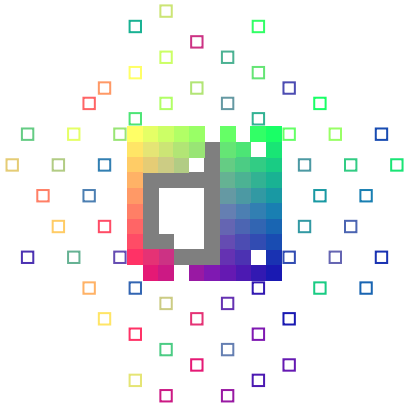
Definition 3.1. Given a set of locations $H \subset \mathbb{Z}^2$, the depth value $D(v)$ for $v \in \mathbb{Z}^2$ is computed recursively:

- If $v \in H$, then $D(v) = 0$.
- Otherwise, $D(v) = 1 + \min_u D(u)$ where the min is over all non-obstacle squares adjacent to v.

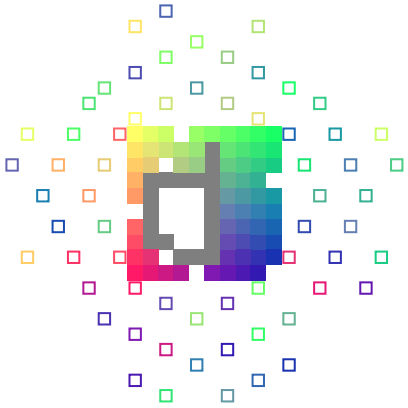
Such values are uniquely defined, can easily be found via breadth-first search, and can be interpreted as the ℓ_1 obstacle-avoiding distance to H in $\mathbb{Z}^2$.

The initialization operates in a few phases:

(1) Compute a set of intermediate positions H, derived from a superset of candidate positions which we call *filler* shapes (see Figure 2). These filler shapes are located outside of the bounding box of the robots' start and target positions, and arranged in a way such that no intermediate positions are adjacent.
(2) Compute a min-cost matching of the robots to their intermediate positions. The cost of a matching is the sum of shortest path lengths from the robots' start and target positions to



(a) `small_005` start and intermediate positions



(b) `small_005` target and intermediate positions

(c) `medium_005` start and intermediate positions

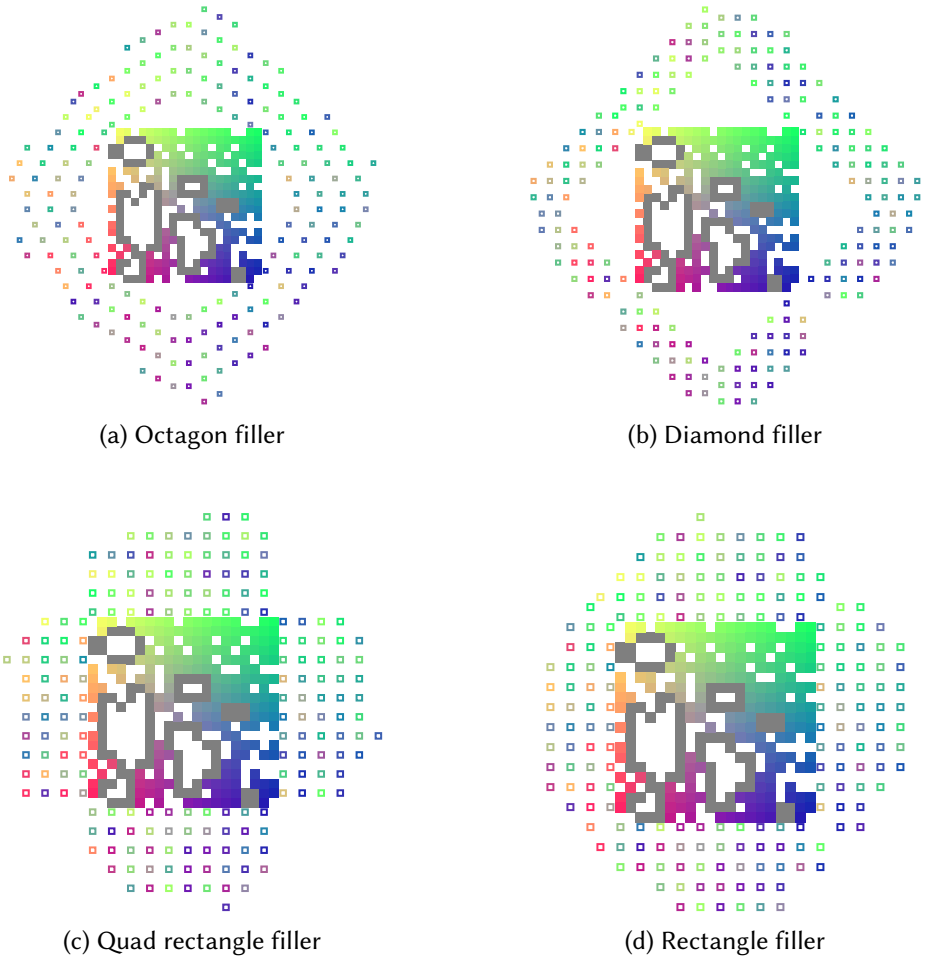
(d) `medium_005` target and intermediate positions

Fig. 1. A visualization of our initializations for the instances `small_005` and `medium_005`. The grey squares are obstacles. On the left (Figures 1a and 1c), the colours of the filled boxes match the instances start positions to the intermediate positions. On the right, (Figures 1b and 1d), the colours match the target positions and intermediate positions.

their matched intermediate position. To improve the initialization, we generate many more intermediate positions than robots. Figure 1 shows an example of the robots' start and target positions and the corresponding matched intermediate positions.

(3) For each location within the bounding box, compute the depth of that location using H (see Definition 3.1). Route the robots from their start position to their corresponding intermediate positions one at a time, in order according to increasing depth values of their start positions. The routes here are constructed to avoid the paths of the robots routed before it.

(4) Re-route the robots *from their start positions* to their target positions' in decreasing order of the *target depth values*. We start with the paths computed in the previous phase, and



(a) Octagon filler (b) Diamond filler

(c) Quad rectangle filler (d) Rectangle filler

Fig. 2. Visualizations of the different methods we used to generate intermediate positions for the instance `small_012`. This visualization only shows the matched potential intermediate positions, which is why the displayed intermediate positions in Figure 2d do not resemble a rectangle.

reroute each robot individually. We do this by removing their path and inserting a new path respecting the current paths of all robots.

Although a natural alternative to step 4 is to simply route the robots from the intermediate locations to their target locations, it is instead much more efficient to re-route from the robots' starting locations. This can be done in a way such that the robots do not block each other in the rerouting, as we describe below.

Lemma 3.2. *Given the paths produced in the third step, a feasible set of paths is guaranteed to exist in the fourth step.*

Proof. As previously stated, robots located at higher depth values do not block robots at lower depth values. This is because a given depth value represents the length of a shortest path from a point in H to a given location. If the depth value of a location is d, then no locations with depth values greater or equal to d will be used by the shortest path (as it would imply a smaller depth value for that location). Thus, there exists a set of paths from each robots' intermediate position to their target positions by routing the robots in order from higher depth values to lower. By re-routing in the fourth step in decreasing order of the target depth, the same depth value properties guarantee that a re-routed robot's path will never block robots of greater or equal depth values. □

Initialization variations. Local search strategies are typically very sensitive to the initialization. We used a number of variations to generate a family of different initializations:

- Using different filler shapes in step 1, or different costs for matching in step 3.
- Finding random shortest paths, or approximate shortest paths for steps 3 and 4.
- Swapping the robots' start and target locations.

The optimization procedure was run on the initializations that scored the highest.

3.2 Solution optimization

The basic paradigm of our local optimization is to choose k robots, and improve their paths while keeping the paths of all others fixed. This type of approach is often referred to as k-opt. This approach has two distinct components: (1) the choice of the k robots and (2) the improvements of their paths.

Choosing the k robots. We experimented with a few types of weighted sampling to choose the k robots:

- *Sampling by completion time.* Sample k robots without replacement, where the sampling probability for each robot is proportional to their completion time (the time at which they make their last move).
- *Sampling by closeness.* Sample the initial robot proportional to completion time, then sample $k-1$ other robots based on proximity to the initial robot's path. The proximity of two paths is the number of time steps for which the two paths are adjacent to each other.
- *Sampling by constraints.* Sample the initial robot proportional to completion time, then find a minimum completion time path from the start to the target for the sampled robot with the relaxation that this robot is allowed to move through $k-1$ other robots. The robots that the path moves through forms the $k-1$ other robots in the sample.

In our experiments, sampling by completion time and by closeness seems to produce the best results. However, we note that Team Shadoks was able to exploit a variant of sampling by constraints to win the MAX category.

Path optimization. Given a sample of k robots, one would like to jointly optimize the k robots simultaneously. However, this approach causes the state space to grow exponentially with the number of robots. Furthermore, due to the size of the grid, the state space for path finding is already quite large to begin with. Like other teams, our path finding was done in the grid-time graph where states are characterized by the positions of the robots at each timestep. For the largest instance in the data set, the size of this state space is already on the order of 10^5. For these reasons, we were only able to scale to $k = 2$ in our code with the approach of joint optimization.

Instead, our main insight was to *approximate* the joint optimization by k individual single path optimizations. Our inspiration was the analogy of sorting: given the k robots, we route the robots one-by-one to their targets, where the j-th robot routed ($j \leq k$) respects the path of robots $1 \ldots j-1$. Crucially, this avoids the exponential blow-up of the state space as we're routing each robot from start to target in succession. The problem then reduces to finding a good ordering of the k robots to reroute. We had the most success by simply ordering the robots by decreasing completion time. If an ordering was infeasible (e.g. due to some subset of the k robots completely blocking off the path from the remaining robots) or not an improvement, it was discarded.

During the competition we found that there were advantages to relatively smaller values of k for faster computation. For our approximate joint optimization, larger k helped to get out of bad local optima. We iteratively used all values of k between 1 and 7 during the challenge.

Since the main component of our algorithm is path-finding, we used a number of practical techniques to reduce its cost. We use A* with a bucket priority queue, where the heuristic function was taken to be either the Manhattan distance or the shortest obstacle-avoiding path distance. To further speed up the search when we have an initial feasible solution, we limit the path-finding algorithm to search locally around the robot's original path, by enforcing that no robot may deviate more than R steps away from the set of positions forming their initial path for a fixed parameter R. Since the search is centred around the initial solution, the solution space is guaranteed to have a feasible solution for any value of R.

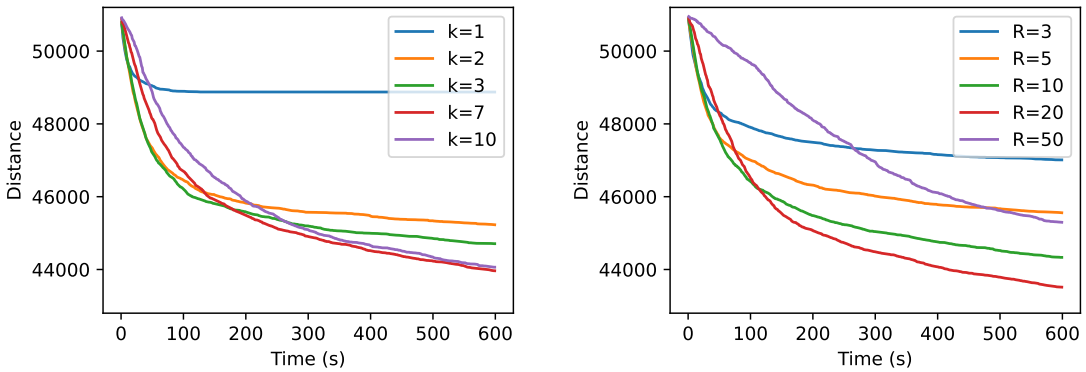
The optimizations above are agnostic to the optimization objective and allows for large gains following the initialization phase for both SUM and MAX.

4 RESULTS

4.1 Computational environment

For the most part, our experiments were done on a desktop with a Core i7-2600. In the last three weeks of the competition we used Stanford's Sherlock High-Performance Computing Cluster for SUM optimization. The University of Waterloo's Multicore lab also generously donated some night-time compute during the last week of the competition in the form of two EPYC 7662s. All code and references to datasets can be found at https://github.com/jacketsj/cgshop2021-gitastrophe.

4.2 Experimental results



(a) SUM optimization varying k where $R = 20$.

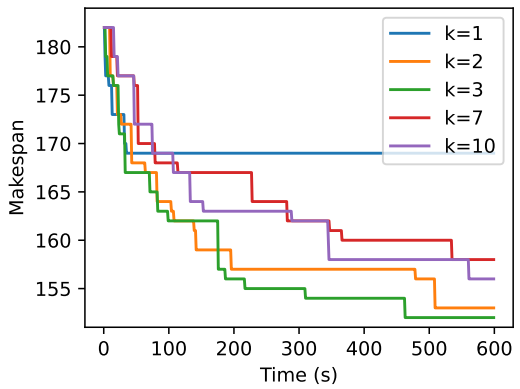
(b) SUM optimization varying R where $k = 7$.

Fig. 3. Varying k and R for SUM optimization on `microbes_004`. Final SUM: 43437.

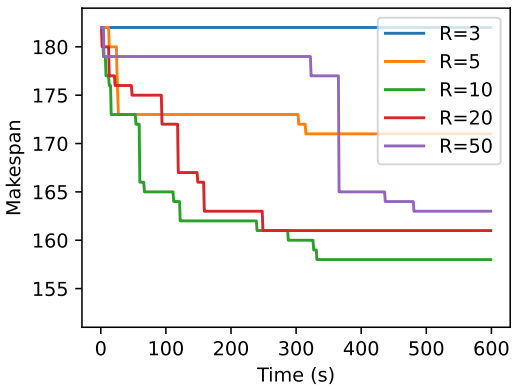
To empirically demonstrate the effect that k and R had on the optimization, we run our algorithm on the instance `microbes_00004`. As seen in Figure 3a, different values of k had an effect on the rate of convergence and the value of the local minimum found by our strategy. Experimentally, values of k greater than 10 did not improve our scores and also run slower. For MAX (in Figure 4a), improvements were much more discrete. During the competition, we continually varied the value of k, since it was unclear which value of k would ultimately give the best score.

As we vary R, we have a similar trade-off between performance and runtime for each optimization step of our algorithm (Figures 3b and 4b). Empirically, we found that choosing R to be around 20 achieved the best balance for most of the instances, as it significantly improved runtime while still giving steady improvements to the solutions.

Figure 5 compares the solutions we computed during the competition to the trivial lower bound, which is given by the maximum shortest-path distance for MAX and the sum of all shortest-path



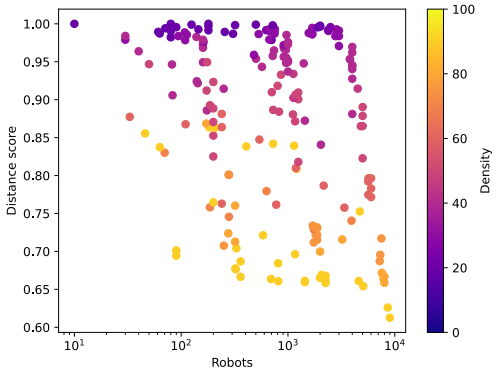
(a) MAX optimization varying k where $R = 20$.



(b) MAX optimization varying R where $k = 7$.

Fig. 4. Varying k and R for MAX optimization on `microbes_004`. Final MAX: 126.

distances for SUM. There is a clear trend that score decreases as density increases (and as n increases, to a lesser extent). This could be due to two factors: first, that the trivial lower bound becomes a worse approximation of the optimal score as these values increase; second, that our algorithm performs poorer on these instances, because more computational time is required and local changes to k robots at a time are insufficient to traverse the solution space.



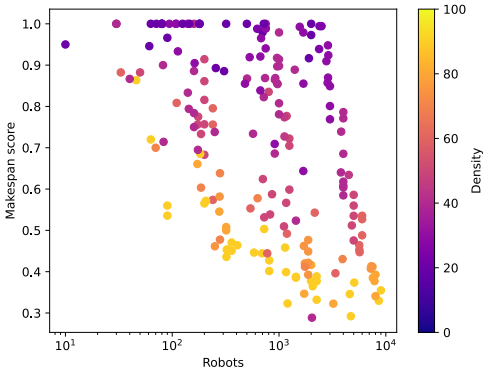


Fig. 5. Plots of $\frac{\text{score}}{\text{lower bound}}$ versus n for both SUM and MAX. Each point corresponds to one instance. The density of the instance, as defined by the contest organizers, is indicated by colour.

4.3 Randomized k-Opt with a conflict optimizer

4.3.1 Implementing the conflict optimizer. After the competition, we implemented Team Shadoks' [1] *conflict optimizer* to see if our k-opt framework could improve on their results. We adapted some optimizations from our k-opt implementation, including the bucket priority queue and the limiting of the pathfinding algorithm to a radius R from the original path. One major difference between our implementation and Team Shadoks' is that we restrict the maximum number of conflicting robots to a small constant c during pathfinding (we used $c = 15$). This allows us to store sets of conflicting robots as small arrays. We also optimize the data structures used for collision detection. Instead of mapping a position and time to a list of robots at that location in a hashtable, we use an array and stored only the unique robot at the location that is not currently in the queue of conflicting robots. All data structures of our algorithm are implemented with plain arrays.

With these optimizations, we obtain optimal results equal to the lower bound (the longest distance between a source and target pair) on some instances in significantly shorter time than Team Shadoks. For example, on `buffalo_004`, our solver obtains the optimal solution from initialization in 10 minutes while Team Shadoks' solver [1] took over 50 minutes, and on `clouds_00001` our solver obtains the optimal solution in 5 minutes while Team Shadoks' solver took over 40 minutes.

4.3.2 Using the conflict optimizer in tandem with k-opt. As previously remarked, randomized k-opt can be used in conjunction with other approaches for better results. We are able to combine k-opt with the conflict optimizer for better and more consistent results. The runtime and probability of success of the conflict optimizer depend heavily on the initial size of the queue (which contains all *span robots*: robots that move at the last time step). We find that randomized k-opt was excellent at taking solutions output by the conflict optimizer and greatly decreasing the number of span robots, which leads to improved runtimes in denser instances where the conflict optimizer is slower.

To exploit this further, we modify the conflict optimizer to only add a subset of span robots into the queue initially. The other span robots not initially in the queue are allowed to have completion times equal to the original makespan, to make it easier for the conflict optimizer to find a solution. With this modification, each run of the conflict optimizer is shorter and more likely to be successful, which enables us to use the faster k-opt more often for improvements and helped in situations where the conflict optimizer gets stuck.

Using this technique, we are able to beat Team Shadoks' score of 421 on `algae_00009` with a makespan of 414 after about 1.5 days of computation time. On instances like `london_night_005`, we do not match Team Shadoks' scores, possibly because of worse initializations, less computation time, our conflict optimizer being slightly more restricted, and the fact that we made no effort to avoid stalls in the optimizer.

5 CONCLUSION

In this paper, we propose *randomized k-opt*, our approach for the coordinated motion planning challenge of CG:SHOP 2021. As implied by the name, our main insight is to break the difficult optimization problem of motion planning into smaller optimizations involving sets of k randomly sampled robots. Using this approach, our team was able to place first in the distance category of the challenge, third in the makespan category, and first in the junior category. We show that our approach is general enough to be used in conjunction with other methods, and in many cases converges quickly to the optimal solution. As the instances get denser however, the efficiency of our approach diminishes. The relative difficulty of optimizing high-density instances has been noted by other teams [1, 7], and remains an avenue for future work.

As the instances to the challenge were very large, we put a substantial amount of work into code optimization. To facilitate future work, our code and testing framework can be found at https://github.com/jacketsj/cgshop2021-gitastrophe.

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*, 2021.

[2] Sándor Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing Coordinated Motion Plans for Robot Swarms: The CG:SHOP Challenge 2021. *CoRR*, abs/2103.15381, 2021. URL: https://arxiv.org/abs/2103.15381, arXiv:2103.15381.

[3] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics Autom. Lett.*, 4(2):1125–1131, 2019. doi:10.1109/LRA.2019.2894217.

[4] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In Brian Magerko and Jonathan P. Rowe, editors, *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17), October 5-9, 2017, Snowbird, Little Cottonwood Canyon, Utah, USA*, pages 270–272. AAAI Press, 2017. URL: https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15845.

[5] Robert Morris, Mai Lee Chang, Ronald Archer, Ernest Vincent Cross II, Shelby Thompson, Jerry L. Franke, Robert Christopher Garrett, Waqar Malik, Kerry McGuire, and Garrett Hemann. Self-driving aircraft towing vehicles: A preliminary report. In Adi Botea and Sebastiaan A. Meijer, editors, *Artificial Intelligence for Transportation: Advice, Interactivity, and Actor Modeling, Papers from the 2015 AAAI Workshop, Austin, Texas, USA, January 26, 2015*, volume WS-15-05 of *AAAI Workshops*. AAAI Press, 2015. URL: http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10075.

[6] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[7] 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*, 2021.

[8] Jingjin Yu and Steven M. LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 3612–3617. IEEE, 2013. doi:10.1109/ICRA.2013.6631084.

[9] Jingjin Yu and Steven M LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.