Fully Decentralized Planner-Guided Robot Swarms

Michael Schader and Sean Luke

George Mason University, Fairfax VA 22030, USA
{mschader,sean}@gmu.edu

Abstract. Robot swarms hold great potential for accomplishing missions in a robust, scalable, and flexible manner. However, determining what low-level agent behavior to implement in order to meet high-level objectives is an unsolved inverse problem. Building on previous work on partially-centralized planner-guided robot swarms, we present an approach that achieves total decentralization of executive and deliberator functions, adds robustness and performance optimization through dynamic task switching, and employs agent-initiated superrational planning to coordinate agent activity while responding to changes in the environment. We demonstrate the effectiveness of the technique with three swarm robotics scenarios.

Keywords: Coordination and control models for multi-agent systems · Knowledge representation and reasoning in robotic systems · Swarm behavior

1 Introduction

Since Beni [3] first developed the idea of robot swarms in 2004, researchers have tried to control large groups of robots in ways that accomplish complex tasks while preserving swarm virtues such as redundancy, parallelism, and decentralization. Despite years of effort since then, Dorigo et al [10] observed in 2020, "[T]he deployment of large groups of robots, or robot swarms, that coordinate and cooperatively solve a problem or perform a task, remains a challenge". Most existing solutions to this challenge either rely on some degree of centralization, which introduces single points of failure and limits scalability, or address only basic missions such as area coverage and shape formation, which are far short of the complex tasks that swarm engineers aspire to perform.

Dorigo predicted that "Hybrid systems mixing model-free and model-based approaches will likely provide additional power". In previous work [24], we employed that philosophy in creating *planner-guided robot swarms*, a hybrid deliberative/reactive approach to swarm management. A central automated planner produced plans for each group of agents within the swarm. At runtime, an *orchestrator* existing outside the swarm issued the plans to the agents, collected success reports, monitored sensor data, determined when actions were complete, and instructed the agents when to advance to the next plan step.

That architecture enabled a human programmer to specify complex missions in a high-level planning language for a swarm to execute. However, the centralized deliberator and executive components were potential single points of runtime failure, reducing the benefits of swarm decentralization. Here we build on that work by modifying the architecture to push the deliberative and executive functions down into the swarm agents

themselves. This involves solving problems with action synchronization, task allocation, and replanning without resorting to outside entities or differentiated swarm members. Ultimately our *distributed executive* accomplishes the same missions that the centralized version can, preserving scalability and robustness without any single points of failure.

In this paper we first review the work done by other researchers on swarm control, showing that no one else has integrated classical planning into a swarm or induced a swarm to accomplish complex actions without central direction or an agent hierarchy. Next, we explain our approach with a formal definition of the system, descriptions of the components of the architecture, and background on the design philosophy behind it. Finally, we report the results of three experiments performed on different scenarios: decentralized shape formation, swarm recovery from loss of agents, and agent-initiated replanning in response to changes in the environment. We demonstrate the fully decentralized swarm's robustness and scalability, validating the effectiveness of our method.

2 Previous Work

Published research touching upon our work can be organized into three groups, based on the degree of decentralization and on whether or not there are separate layers specifying the mission goals and the individual agent behaviors:

Partially centralized These methods lead to hub and spoke or hierarchical architectures. Becker et al [2] explored how a single signal broadcast to all agents in a massive swarm could be used to guide them to collectively accomplish a task. Kominis et al [14] translated uncertain, multi-agent planning problems into classical, single-agent ones that could be centrally solved and then given to the agents. Corah et al [9], Nissim et al [19], and Torreño et al [27] implemented methods to break preliminary plans into parts and have agents refine them through multiple planning rounds. Choudhury et al [7] and Riyaz et al [23] created hybrid systems, with a centralized task planner to assign tasks to individual robots combined with an onboard control system enabling each robot to refine and execute its task. All these methods rely on some central component, which represents a single point of failure and a limiting factor on scalability.

Decentralized single-layer These approaches amount to control laws which must be developed prior to runtime. Atay et al [1], Li et al [15], and Sheth [25] created emergent task allocation methods in which each robot only used information from its neighbors to select tasks, then sent coordination messages to resolve conflicts, possibly including adversarial interactions. Chaimowicz et al [6], Ghassemi et al [11], and Michael et al [18] used combinations of bidding and recruitment to determine role assignments and when they should be changed. Each of these methods involves designing integrated high- and low-level activities, limiting flexibility when developing solutions matching swarm platforms to specific problems.

Decentralized multi-layer Such systems combine the elimination of single points of failure with the relative ease of separately addressing domain-level and behavior-level concerns. Birattari et al [4] and Bozhinoski et al [5] proposed "automatic offline design":

enumerating likely missions within a problem domain, using reinforcement learning or evolutionary computing to generate suitable low-level behaviors in simulation, and deploying the best solution available for the problem at hand when needed. Coppola [8] explored this approach extensively. Although promising, this family of solutions requires the development of a large library of pre-generated behaviors to match to missions when needed. Our method falls into the same decentralized multi-layer category, but does not depend on having prebuilt solutions to new mission requirements.

3 Method

In our earlier work, we introduced a novel approach to swarm control: framing the high-level domain and problem using Planning Domain Definition Language (PDDL) [17], generating a plan to achieve the goal state with an automated planner, and having a central executive orchestrate the agents' activities by adjusting their behavioral parameters and synchronizing their plan step advances. In this new work, we move the executive and deliberative functions into the swarm agents themselves, thus eliminating all single points of failure and enabling truly decentralized operations. We add dynamic task switching based on action completion information shared by neighbors, enhancing robustness. Finally, we incorporate agent-initiated replanning to allow the swarm to respond to changes in the environment.

In our revised formulation, a planner-guided swarm scenario definition can be represented by a tuple:

$$S_{def} = \langle A, domain, M_{act}, M_{pred} \rangle \tag{1}$$

where the agent class $A = \langle sensors, behaviors \rangle$ represents the capabilities of a swarm robot platform, the $domain = \langle predicates, actions \rangle$ is the PDDL representation of the planning domain, the action mapping $M_{act} : actions \rightarrow \langle behaviors, parameters, criteria \rangle$ translates each PDDL action to a specific parameterized agent behavior with success criteria, and the predicate mapping $M_{pred} : predicates \rightarrow \langle sensors, parameters, criteria \rangle$ ties predicates needed for replanning to observed states.

A specific run of a scenario starts with scenario definition S_{def} and adds three items:

$$S_{run} = \langle S_{def}, problem, n, g \rangle \tag{2}$$

which are the PDDL expression of the planning *problem*, the count of agents n, and the number of groups g. If g is set to zero, the group count decision is delegated to each agent's deliberator, which will attempt to generate plans with various numbers of groups, choosing the smallest g that produces a minimum-length sound plan.

3.1 Definitions

Domain and Problem The PDDL domain defines how the world works from the swarm's top-level perspective: what constants will always be present, what predicates can be true or false, and what actions can be performed in terms of their preconditions and effects. The PDDL problem specifies the objects to consider, the initial conditions of the situation, and the goal state to be achieved. The scenario designer creates these two files to control the swarm.

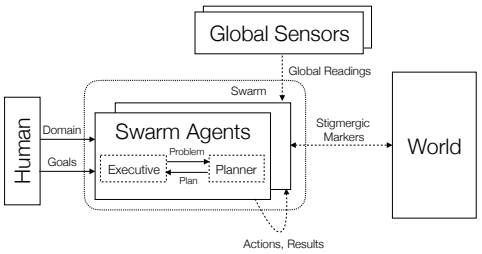


Fig. 1: Fully decentralized planner-guided robot swarm architecture with optional global sensors.

Agent Class The agent class defines the capabilities of the agents as they relate to other parts of the scenario in terms of sensors and behaviors. All agents of the swarm belong to this single class. The sensors include all the ways an agent can receive information from the world around it: its own local readings, data it receives from global sensors, and information exchanged with other agents in its neighborhood. The behaviors are the micro-behaviors that each swarm agent performs (e.g. "find item A", "discover site B", "deposit item A at site B"), which lead to emergent phenomena such as coverage and foraging. A given agent class can be used in multiple scenarios, with different domains and problems.

Each instantiated agent has its own planner, which takes PDDL domain and problem files as input and produces PDDL plans as output. The specific planner implementation can be changed at scenario start time, but it must be identical for all agents, be able to process the domain file constructs, and be deterministic (always producing the same output for given inputs). We employ parallel planning to generate plans with multiple simultaneous actions that are assigned to *virtual agents*, groups of real agents within the swarm. Note that this is not multi-agent planning in the sense of generating joint actions or violating classical planning assumptions; rather, we achieve parallelism by taking advantage of partially ordered plans in which the actions at each given plan step are independent.

An agent's executive manages its specific movements, manipulations, and communications. Our own software uses a state machine with states EXPLORING, CARRYING, and DONE, but this is just an implementation detail that is not demanded by the planner-guided swarm framework. The executive also determines when sensor inputs should drive replanning based on an updated set of initial conditions in the problem statement.

Action and Predicate Mappings The bridge between the high-level view of the world in the planning domain and the low-level configuration of the swarm agents' micro-behaviors is the mapping layer. When an agent class is paired with a domain file, the programmer builds two mappings. The first mapping translates domain actions (e.g. "pick up block A") into parameterized agent behaviors that will lead to the desired effect ("use the foraging behavior with the target item set to bricks of type A"). Bundled with this are the success criteria that must be met to infer that the action has been completed. The second mapping translates grounded domain predicates ("site D is full") into sensor conditions that will reveal its truth or falsehood ("check if the count of bricks deposited in site D equals the size of site D", or "determine if a sensor indicates site D is full").

The action mapping is critical in that it translates the abstract actions of a plan into the configuration of each agent's behaviors. The predicate mapping is not needed if the scenario designer specifies all initial conditions and there is no need for replanning; however, if the agents will need to assess conditions to plan or replan, then the relevant predicates do indeed need to be mapped to sensor readings and shared knowledge.

3.2 Decentralized plan monitoring

The agents keep track of their individual successes executing behaviors tied to plan actions, generating a *success token* each time they finish a granular activity (e.g. "remove item A from site B", "discover site C"). By exchanging these success tokens with each other in the course of their local interactions, sometimes along with factoring in data from their own sensors or global ones, the agents can determine when the plan action assigned to their group is complete. In addition, they keep track of other groups' progress toward task completion. When an agent learns that all groups have finished the current planglo step, it advances to the next plan step. As the same knowledge spreads, other agents will make the same decision, ending up with all agents on the same next step.

3.3 Dynamic task switching

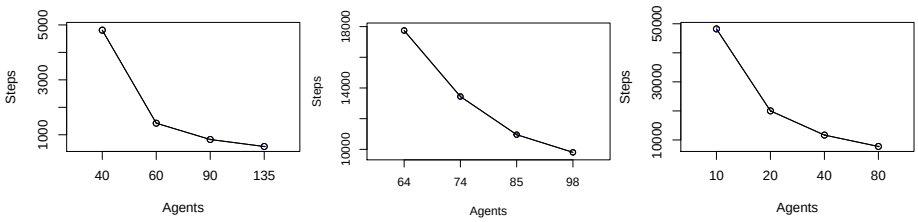
Since each agent knows the action completion status of its own group as well as that of the other groups, it can choose to temporarily switch groups when certain criteria are met. If an agent's current action is complete, and it is not in a state that precludes switching tasks (e.g. already carrying a certain item), it can identify other groups that have not finished their actions. If there is such a group, then the agent will switch to it based on a configured probability (0.1 in our experiments; the exact value has little effect on performance as long as it is positive). This task switching serves both to optimize the swarm's allocation of agents to actions, as well as to provide a fallback capability in case a portion of the swarm is destroyed or disabled.

3.4 Agent-initiated replanning

Hofstadter [12] named and refined the notion of *superrational groups* in 1983: when participants in an interaction know that all the others think the same way as they do, and that each recursively knows that the others know this, then they should independently arrive at identical conclusions aimed at maximizing joint benefit. In 2019, Tohmé et al [26] developed a formal analysis of the concept and determined it to be sound. In our situation of building homogeneous swarms in which all the agents have the same software and goals, the necessary conditions for superrationality do indeed hold, given enough time for the agents in the swarm to converge on the same knowledge of the state of the world. With a deterministic planner, we can be sure that subject to information propagation delay, the agents will produce the same updated plans as each other.

4 Experiments

We conducted three experiments designed to test the novel aspects of our fully decentralized planner-guided robot swarm implementation, seeking to verify that the new



(a) Letters. (b) Refineries. (c) Smart Sorting.

With 1000 runs of each treatment, the confidence intervals are too small to see.

Fig. 2: Mean steps to completion of scenario for various swarm sizes.

mechanisms succeeded reliably while scaling efficiently as agents were added to the swarm. First, we exercised basic operations with all centralized components eliminated. Second, we tested agent-initiated task switching to see if it led to robust recovery from agent failures. Third, we evaluated the effectiveness of decentralized replanning spurred by detected changes in the world state. All of our experiments were conducted in the MASON multiagent simulation toolkit [16], in which we modeled non-point robots, each exclusively occupying a nonzero area. Agents navigated using virtual pheromone trails, an established swarm mechanism [20] that was just one of several methods with which they could find their way.

4.1 Letters: Runtime-decentralized planning, coordination, and monitoring

The Letters scenario is a straightforward mission to have robots arrange themselves to spell out a three-letter sequence (Figure 3). The locations of the pixels of each letter are marked in advance, and the agents know when they have entered such a designated region. The purpose of the experiment is to show the effectiveness and scalability of a completely decentralized planner-guided swarm. Once the robots are launched, they have no special base to which to return to or overseer with which to communicate. They have only the domain and problem presented by the operator to the swarm.

This experiment used the PDDL4J sequential planner [21] with the fast-forward heuristic search option. We varied the number of agents from 40 up to 135 to observe the effect on the average number of steps needed to reach the goal state (Figure 2a). We performed 1000 runs of each treatment with 100% success and verified for statistical significance using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

A minimum of 39 agents was necessary to finish this mission, 13 for each of the three letters. The first treatment with 40 agents took an average of 4809 steps to complete. With 60 agents, that dropped dramatically to 1422, since there were more available to find and remain in the designated areas, especially for the later spaces to be filled in. With 90 and 135 agents, the steps needed were further reduced to 828 and 573; the speedup from more agents leveled off due to physical interference with each other as well as diminishing returns from having many potential fillers for each needed position.

(a) The swarm agents explore the area, stopping when they reach a location within the specified destination region.

(b) News that Step 1 is complete has propagated through the swarm, and the agents move onto their next action.

(c) The swarm finishes Step 2 and that word begins to spread.

(d) Ultimately, the agents finish Step 3 and the task is completed.

The green and blue dots represent agents on odd- and even-numbered steps. They turn black when plan execution is complete.

Fig. 3: Stages of the Letters scenario.

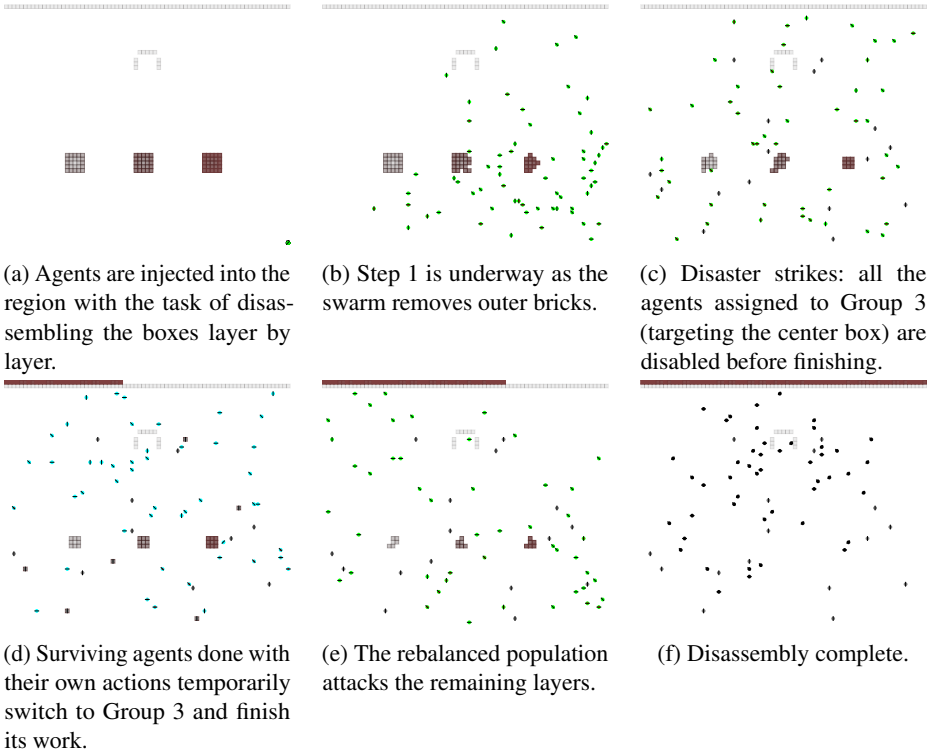
4.2 Refineries: Dynamic task switching in response to group failure

Refineries is a stress test of agent task-switching (Figure 4). There are three square piles of bricks, each consisting of three different layers. One group of agents is assigned to disassemble each pile. The agents need to bring the bricks to a refinery area, where they will be processed and removed from the environment. The outer bricks must all be processed first, then the middle ones, and finally the central bricks. Partway through the first step, however, all the agents initially assigned to one of the groups are disabled: rendered permanently immobile. The only way for the swarm to succeed is for the agents to determine via their short-range interactions that one task group is not succeeding, and to choose to switch into that group in order to accomplish the mission.

This experiment used the Blackbox parallel planner [13] with its Graphplan solver ensuring deterministic output. We varied the number of agents from 64 up to 98 to observe the effect on the average number of steps needed to reach the goal state (Figure 2b). We performed 1000 runs of each treatment with 100% success and verified for statistical significance using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

A minimum of 64 agents was needed to complete this assignment: 16 in each of three groups to gather the outermost layers, plus another 16 in the spare group. With the minimum number it took an average of 17,744 steps to finish. Using 80 agents reduced that to 13,444, and with 85 it took 10,957; the additional workers allowed the swarm to perform the discovery and moving jobs more quickly. 98 agents only improved the step count to 9807. The limited space around the pickup and dropoff sites placed an upper bound on the scalability of the swarm, as too many agents on the field blocked each other from moving around as needed.

The ability of the agents to temporarily switch task groups was critical to the swarm's recovery from the externally-imposed disaster, the disabling of all the agents in Group 3. Figure 5 shows the number of agents working in each group through one run of the simulation. Early on, members of the unassigned Group 4 switched to Group 2, which had the job of collecting bricks from the site farthest from the launch point and so needed the help. At step 3000, the Group 3 members were immobilized and their numbers disappeared from the graph. Soon after step 5000, some Group 2 members switched to Group 3 to make up for the lost effort. Around step 7000, the numbers in each group



(a) Agents are injected into the region with the task of disassembling the boxes layer by layer.

(b) Step 1 is underway as the swarm removes outer bricks.

(c) Disaster strikes: all the agents assigned to Group 3 (targeting the center box) are disabled before finishing.

(d) Surviving agents done with their own actions temporarily switch to Group 3 and finish its work.

(e) The rebalanced population attacks the remaining layers.

(f) Disassembly complete.

Green and blue dots are working agents. Gray dots are permanently disabled ones. The brown progress bar at the top shows how many bricks have been dropped off at the gray-walled refinery.

Fig. 4: Stages of the Refineries scenario.

equalized, then from step 11,000 onward the numbers fluctuated based on which groups had completed their assigned actions at the time. The low-level task switching behavior made the swarm robust and able to finish its job even when an entire task group was lost.

4.3 Smart Sorting: Self-initiated replanning to handle changed situation

The Smart Sorting scenario exercises the agents' coordinated replanning abilities (Figure 6). The swarm starts with the mission of gathering four different kinds of randomly scattered bricks and assembling them in order into blocks in a walled area. As soon as they finish the first two layers (A and B), though, the A block is teleported outside to a different location. The agents continue with their planned actions, but upon checking sensor readings, they determine that conditions have changed, so they replan and begin taking apart the outer blocks so as to reassemble the correct structure.

This experiment used the Madagascar parallel planner [22], specifically its MpC implementation. We varied the number of agents from 10 up to 80 to observe the effect on the average number of steps needed to reach the goal state (Figure 2c). We performed

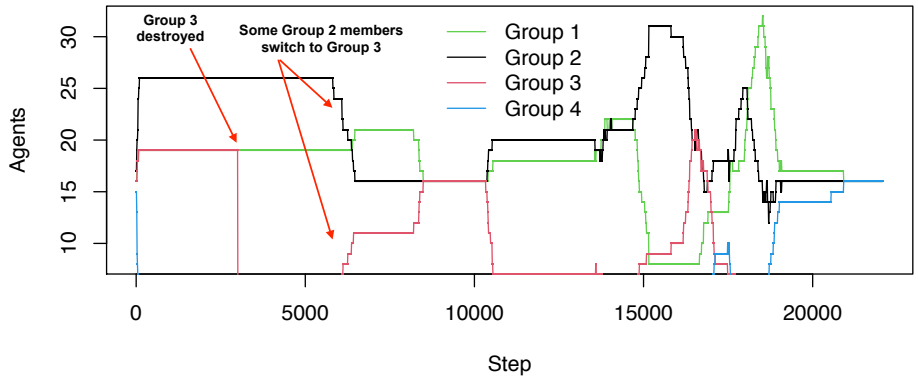


Fig. 5: Number of agents working in each task group as time advances in a single run of the Refineries scenario. At step 3000, all the agents in Group 3 were disabled; soon after, members of other groups switched in order to finish Group 3's assigned tasks.

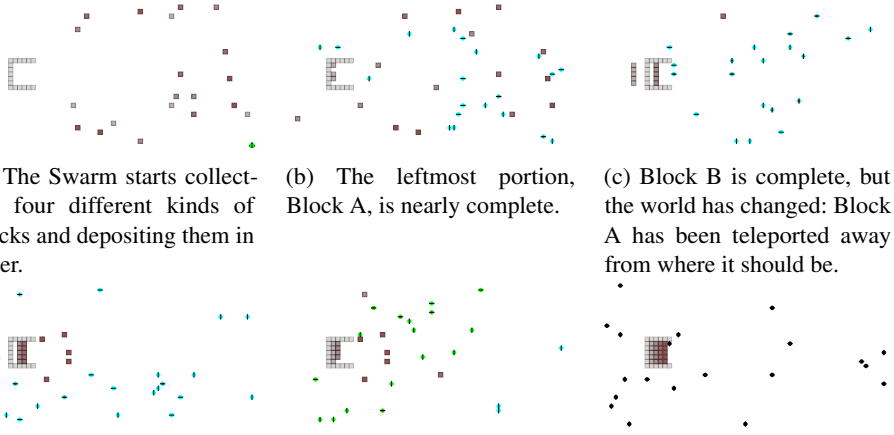
1000 runs of each treatment with 100% success and verified for statistical significance using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

The minimum number of agents needed to complete this scenario with two groups was ten, enough for each group to collect all five bricks of a single type. With that smallest possible population, the swarm took 48,230 steps on average to finish. With 20 agents, that was slashed to 20,018; with 40 it dropped to 11,686; and with 80 it was 7766. This excellent scalability was due to more agents being available to explore and move bricks around, along with faster information dissemination caused by increased agent density in the simulation work area.

5 Conclusions and Future Work

Modifying our previously published planner-guided robot swarm architecture to achieve complete decentralization was a success. Each scenario explored in our experiments showed a different area of improvement. Eliminating all central components ensured there were no single points of failure. Introducing dynamic task switching provided robustness against agent failure. Superrational planning enabled the swarm to incorporate flexibility into swarm behavior. We conducted all the experiments using the same agent code, further demonstrating the generality of our method.

In future work, we will attack the problem of retrograde behavior (agents getting out of sync with each other's plan steps), quantify aspects of the speed of communications in a swarm environment, and implement different agent classes with varying navigation and sensing mechanisms. This work will show for the first time a widely-applicable approach to building robot swarms that can collectively accomplish complex tasks.



(a) The Swarm starts collecting four different kinds of blocks and depositing them in order.

(b) The leftmost portion, Block A, is nearly complete.

(c) Block B is complete, but the world has changed: Block A has been teleported away from where it should be.

(d) The agents receive a sensor reading that Block A is out of position; they replan and start disassembling Blocks B, C, and D.

(e) Block A has been returned to its correct location, and the agents turn to reassembling Blocks B, C, and D.

(f) Mission success.

The green and blue dots represent agents on odd- and even-numbered steps. The gray/brown shaded squares make up the four different blocks. The gray lines are walls.

Fig. 6: Stages of the Smart Sorting scenario.

References

1. Atay, N., Bayazit, B.: Emergent task allocation for mobile robots. In: Proceedings of Robotics: Science and Systems. Atlanta, GA, USA (June 2007)
2. Becker, A., Demaine, E.D., Fekete, S.P., Habibi, G., McLurkin, J.: Reconfiguring massive particle swarms with limited, global control. In: International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics. pp. 51–66. Springer (2013)
3. Beni, G.: From swarm intelligence to swarm robotics. In: International Workshop on Swarm Robotics. pp. 1–9. Springer (2004)
4. Birattari, M., Ligot, A., Bozhinoski, D., Brambilla, M., Francesca, G., Garattoni, L., Garzón Ramos, D., Hasselmann, K., Kegeleirs, M., Kuckling, J., et al.: Automatic off-line design of robot swarms: a manifesto. Frontiers in Robotics and AI **6**, 59 (2019)
5. Bozhinoski, D., Birattari, M.: Designing control software for robot swarms: Software engineering for the development of automatic design methods. In: 2018 IEEE/ACM 1st International Workshop on Robotics Software Engineering (RoSE). pp. 33–35 (2018)
6. Chaimowicz, L., Campos, M.F.M., Kumar, V.: Dynamic role assignment for cooperative robots. In: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). vol. 1, pp. 293–298 vol.1 (2002)
7. Choudhury, S., Gupta, J., Kochenderfer, M., Sadigh, D., Bohg, J.: Dynamic Multi-Robot Task Allocation under Uncertainty and Temporal Constraints. In: Proceedings of Robotics: Science and Systems (July 2020)

8. Coppola, M.: Automatic Design of Verifiable Robot Swarms. Ph.D. thesis, Delft University of Technology (2021)
9. Corah, M., Michael, N.: Efficient online multi-robot exploration via distributed sequential greedy assignment. In: Proceedings of Robotics: Science and Systems (July 2017)
10. Dorigo, M., Theraulaz, G., Trianni, V.: Reflections on the future of swarm robotics. Science Robotics **5**(49) (2020)
11. Ghassemi, P., Chowdhury, S.: Decentralized task allocation in multi-robot systems via bipartite graph matching augmented with fuzzy clustering. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. vol. 51753, p. V02AT03A014. American Society of Mechanical Engineers (2018)
12. Hofstadter, D.R.: Dilemmas for superrational thinkers, leading up to a luring lottery. Scientific American **248**(6), 739–755 (1983)
13. Kautz, H., Selman, B.: Blackbox: A new approach to the application of theorem proving to problem solving. In: AIPS98 Workshop on Planning as Combinatorial Search. vol. 58260, pp. 58–60 (1998)
14. Kominis, F., Geffner, H.: Beliefs in multiagent planning: From one agent to many. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 25 (2015)
15. Li, J., Abbas, W., Shabbir, M., Koutsoukos, X.: Resilient Distributed Diffusion for Multi-Robot Systems Using Centerpoint. In: Proceedings of Robotics: Science and Systems (July 2020)
16. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multiagent simulation environment. Simulation **81**(7), 517–527 (2005)
17. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL: the planning domain definition language (1998)
18. Michael, N., Zavlanos, M., Kumar, V., Pappas, G.: Distributed multi-robot task assignment and formation control
19. Nissim, R., Brafman, R.I., Domshlak, C.: A general, fully distributed multi-agent planning algorithm. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 1323–1330 (2010)
20. Panait, L., Luke, S.: A pheromone-based utility model for collaborative foraging. In: Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on. pp. 36–43. IEEE (2004)
21. Pellier, D., Fiorino, H.: PDDL4J: a planning domain description library for Java. Journal of Experimental and Theoretical Artificial Intelligence **30**(1), 143–176 (2018)
22. Rintanen, J.: Madagascar: Scalable planning with sat. Proceedings of the 8th International Planning Competition (IPC-2014) **21** (2014)
23. Riyaz, S.H., Basir, O.: Intelligent planning and execution of tasks using hybrid agents. In: 2009 International Conference on Artificial Intelligence and Computational Intelligence. vol. 1, pp. 277–282 (2009)
24. Schader, M., Luke, S.: Planner-guided robot swarms. In: Demazeau, Y., Holvoet, T., Corchado, J.M., Costantini, S. (eds.) Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness. The PAAMS Collection 18th International Conference, PAAMS 2020, October 7-9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12092, pp. 224–237. Springer (2020)
25. Sheth, R.S.: A Decentralized Strategy for Swarm Robots to Manage Spatially Distributed Tasks. Ph.D. thesis, Worcester Polytechnic Institute (2017)
26. Tohmé, F.A., Viglizzo, I.D.: Superrational types. Logic Journal of the IGPL **27**(6), 847–864 (2019)
27. Torreño, A., Onaindia, E., Sapena, O.: An approach to multi-agent planning with incomplete information. arXiv preprint arXiv:1501.07256 (2015)

6 Appendix: PDDL Files

Letters domain, problem, and plan

```
(define (domain LETTERS) (:requirements :strips :typing) (:types group site)
  (:constants site-g site-m site-u - site) (:predicates (visited ?s - site) (dummy))
  (:action visit-g :parameters (?g - group)  :precondition  () :effect  (visited site-g))
  (:action visit-m :parameters (?g - group) :precondition (visited site-g) :effect (visited site-m))
  (:action visit-u :parameters (?g - group)  :precondition  (visited site-m) :effect (visited site-u)))
(define (problem GMU) (:domain LETTERS) (:objects group1 - group) (:init (dummy))
  (:goal (and (visited site-g) (visited site-m) (visited site-u))))

;;;; Plan for one group
1 ( (visit-g group1) )     2 ( (visit-m group1) )     3 ( (visit-u group1) )
```

Refineries domain, problem, and plan

```
(define (domain REFINERIES) (:requirements :strips :typing) (:types group item site)
  (:constants site-1a site-1b site-1c site-2a site-2b site-2c site-3a site-3b site-3c refinery - site
    item-1a item-1b item-1c item-2a item-2b item-2c item-3a item-3b item-3c - item)
  (:predicates (empty ?s - site) (all-at ?i - item ?s - site) (uncarrying ?g - group) (carrying ?g - group ?i - item))
  (:action collect-from-1a :parameters (?g - group) :precondition (and (uncarrying ?g) (all-at item-1a site-1a)
    (empty site-1b) (empty site-1c)) :effect (and (not (uncarrying ?g)) (carrying ?g item-1a) (empty site-1a)))
  (:action deposit-at :parameters (?g - group ?i - item ?s - site) :precondition (and (carrying ?g ?i))
    :effect (and (uncarrying ?g) (not (carrying ?g ?i)) (all-at ?i ?s))))  ;; actions repeated for all sites and items
(define (problem DISPOSE) (:domain REFINERIES) (:objects group1 group2 - group group3 - group)
  (:init (uncarrying group1) (uncarrying group2) (uncarrying group3)
    (all-at item-1a site-1a) (all-at item-1b site-1b) (all-at item-1c site-1c)) ;; predicates repeated for all
  (:goal (and (all-at item-1a refinery) (all-at item-1b refinery) (all-at item-1c refinery)))) ;; predicates repeated for all

;;;; Plan for four groups (three groups would be optimal, one is added for redundancy)
1 ( (collect-from-3c group1 ) (collect-from-1c group2 ) (collect-from-2c group3 ) nil )
2 ( (deposit-at group1 item-3c refinery) (deposit-at group2 item-1c refinery) (deposit-at group3 item-2c refinery) nil )
3 ( (collect-from-3b group1 ) (collect-from-1b group2 ) (collect-from-2b group3 ) nil )
4 ( (deposit-at group1 item-3b refinery) (deposit-at group2 item-1b refinery) (deposit-at group3 item-2b refinery) nil )
5 ( (collect-from-3a group1 ) (collect-from-1a group2 ) (collect-from-2a group3 ) nil )
6 ( (deposit-at group1 item-3a refinery) (deposit-at group2 item-1a refinery) (deposit-at group3 item-2a refinery) nil )
```

Smart Sorting domain, problem, initial plan, and revised plan

```
(define (domain SMART-SORTING) (:requirements :strips :typing :equality :disjunctive-preconditions)
  (:types group item site) (:constants site-a site-b site-c site-d - site item-a item-b item-c item-d - item)
  (:predicates (empty ?s - site) (all-at ?i - item ?s - site) (some-at ?i - item ?s - site)
    (uncarrying ?g - group) (carrying ?g - group ?i - item))
  (:action collect :parameters (?g - group ?i - item) :precondition (and (uncarrying ?g))
    :effect (and (not (uncarrying ?g)) (carrying ?g ?i)))
  (:action clear-out :parameters (?g - group ?s - site) :precondition (and (uncarrying ?g)
      (or (= ?s site-d) (and (= ?s site-c) (empty site-d)) (and (= ?s site-b) (empty site-d) (empty site-c))
        (and (= ?s site-a) (empty site-d) (empty site-c) (empty site-b))))
    :effect (and (not (all-at item-a ?s)) (not (all-at item-b ?s)) (not (all-at item-c ?s)) (not (all-at item-d ?s))
      (not (some-at item-a ?s)) (not (some-at item-b ?s)) (not (some-at item-c ?s)) (not (some-at item-d ?s))
        (empty ?s) (uncarrying ?g)))
  (:action deposit :parameters (?g - group ?i - item ?s - site) :precondition (and (carrying ?g ?i) (empty ?s)
      (or (= ?s site-d) (and (= ?s site-c) (empty site-d)) (and (= ?s site-b) (empty site-d) (empty site-c))
        (and (= ?s site-a) (empty site-d) (empty site-c) (empty site-b))))
    :effect (and (uncarrying ?g) (not (carrying ?g ?i)) (all-at ?i ?s) (some-at ?i ?s) (not (empty ?s)))))
(define (problem REPLAN) (:domain SMART-SORTING) (:objects group1 group2 - group)
  (:init (uncarrying group1) (uncarrying group2) (empty site-a) (empty site-b) (empty site-c) (empty site-d))
  (:goal (and (all-at item-a site-a) (all-at item-b site-b) (all-at item-c site-c) (all-at item-d site-d))))

;;;; Initial plan for two groups
1 ( (collect group1 item-a) (collect group2 item-b) )     2 ( (deposit group1 item-a site-a) nil )
3 ( nil (deposit group2 item-b site-b) )     4 ( (collect group1 item-d) (collect group2 item-c) )
5 ( nil (deposit group2 item-c site-c) )     6 ( (deposit group1 item-d site-d) nil )
;;;; Revised plan for two groups (after item-a moved out of correct position)
1 ( (clear-out group1 site-d) nil )     2 ( (collect group1 item-a) (clear-out group2 site-c) )
3 ( nil (clear-out group2 site-b) )     4 ( (deposit group1 item-a site-a) (collect group2 item-b) )
5 ( (collect group1 item-c) (deposit group2 item-b site-b) )     6 ( (deposit group1 item-c site-c) (collect group2 item-d) )
7 ( nil (deposit group2 item-d site-d) )
```