Randomization as a Robust Exploration and Coordination Tool in Situated Multi-Agent Systems

Andreas Awouters, Alexandre Baré, Mathijs Dom, Aloïs Van de Voorde
Computer Science
KULeuven
Leuven, Belgium
{andreas.awouters, alexandre.bare, mathijs.dom, alois.vandevoorde}@student.kuleuven.be

***Abstract*—In this paper we discuss a multi-agent system approach to solve multiple challenges in an automated guided vehicle abstraction: The Packet World. The implementation is aimed at efficiently delivering multiple packets to specific destinations. Agents operate in a partially observable environment where they explore and make use of their individual memory to progressively build up a reliable belief of the world. This knowledge of the world is used to find effective paths that avoid all obstacles using the A* algorithm. They autonomously recharge their battery at energy stations which they find by following the gradient fields these emit. Collaborative behaviours allow agents to solve ever more complex environments. Our study focuses in particular on exploration, coordination and optimal decision-making with limited information about the environment. We argue that randomization can be a robust tool to reach these objectives.**

Index Terms*—Situated Multi-Agent System, Automated Guided Vehicle, Packet-World, Randomization, Exploration, Coordination, A

I. Introduction

Multi-agent systems are becoming more prevalent in today's world. This stems from their ability to perform complex tasks that are impossible or time-consuming for a single agent or a more traditional system with a central server. Multi-agent systems consist of multiple autonomous and independent agents with a limited knowledge of the world they operate in. These agents have to communicate with one another to complete some tasks as efficiently as possible. In essence a multi-agent system is a system that is structured as a set of autonomous agents that are able to flexibly adapt their behavior to changing operating conditions. [1]

A centralized control architecture can often make better decisions since it has a global knowledge of the current situation. On the other hand, if this central server were to experience a failure, the entire system would be out of order. With a multi-agent system, the knowledge is distributed and agents have to make decisions based on limited information. This may lead to some inefficiencies but it also means agents are more flexible and can adapt to a variable stream of tasks. It makes the system more scalable. Furthermore if one of the agents were to fail, it could be removed from the environment and repaired without affecting the overall functioning of the system.

Since these systems are most effective for large, complex problems and such problems exist in varying disciplines, it can be difficult to do research using real-world problems. Therefore an abstraction of the applications is often made for research purposes. In this paper, the Packet-World [2] is used as a test bed. It can be seen as an abstraction of many different real-world problems.

A first example of such a real-world application is the one of large autonomous warehouses such as in [3] and [4]. Some examples of such warehouses are those of package delivery services. Those have a lot of throughput and could benefit from autonomous, battery powered transportation vehicles to perform the transport of packages. In this application the transport of a package from one location to another can be mapped onto the packets that have to be delivered to their destination in the packet world. The vehicles performing these movements can be mapped onto the agents.

While the automated warehouse example is an obvious application of the packet world a more obscure one is that of mapping an unknown environment to find shortest paths within a maze. This application can be used in emergency situations as detailed in [5]. For example, victims could be trapped inside of a collapsed building after an explosion or earthquake and emergency personnel might not be able to reach them immediately. In this case small autonomous robots could be deployed to map the damaged environment and possibly find a path that could lead emergency personnel to the victims or possibly these robots could extricate the victims themselves. Once again the robots can be mapped onto the agents in the packet-world while the victims could be mapped onto the packets and the emergency personnel could be seen as the destinations. This is a more abstract mapping but nonetheless a possible application of multi-agent systems in the real world.

II. Packet-World Description

The Packet-World is a test bed developed by Weyns et al. [2] in java to experiment with situated multi-agent systems. A situated multi-agent system is a decentralized system of agents whose individual states are characterized among other by their position in an environment. Each agent can perceive its environment locally and is given a behaviour to follow based on its current internal state. In a given behavioural graph, each behaviour corresponds to a node and an agent transitions from one behaviour to another when it meets the changing conditions specified as vertices in the graph. The inherent

patterns, built-in the Packet-World, of virtual environment, situated agents, roles and situated commitments are described further in [13].

The Packet-World is composed of a two-dimensional rectangular grid in which packets (represented by squares) of different colors are scattered. Agents in this simulated world have to collect the packets and bring them to the destination (represented by a circle) of the same color. An environment is considered solved once all packets are delivered to their destinations. Agents can move one step at a time in any of the eight directions within the environment, pick or drop a packet or skip their turn.

Besides acting in the environment, agents can also communicate with each other to exchange information, either by message-passing (direct communication) or through environment-mediated communication (indirect communication).

Agents only have a localized perception of the world due to a limited view-range. They can see walls, packets, destinations, energy stations and indirect communication items of different sorts. To achieve optimal performance, they are thus required to explore and memorise the environment and communicate and cooperate with other agents.

III. Problem Description

At the start of the simulations, agents are not given any information about the environment outside of their field of vision. The environments is said to be unexplored. In addition, environments are also dynamic: agents move around, packets can appear (at a packet generator or when dropped by an agent), move (when they are picked up by an agent) and disappear (when they are brought to a destination of the same color). This results in agents lacking information about the environment when changes happen outside of their field of vision. This information is needed in order for an agent to make optimal decisions such as finding the shortest or most energy efficient path between itself, packets and their destinations. Furthermore some environments deal with agents that have a limited energy supply. Managing their battery and charging it in time becomes also a requirement.

The challenge is thus to explore the environment to improve the agents knowledge of the world and to exploit the limited information at hand to effectively solve complex charging/pickup/delivery tasks while minimising energy consumption.

The goal of this research paper is to answer the question: "How can randomization be used to align multiple agents' exploration and exploitation objectives in a partially-observable environment?"

We will introduce seven test environments of increasing complexity. Among those, the main optimal decision-making problems can be split into three parts: autonomous behaviour of an agent, managing a limited energy supply, and collaboration with other agents.

A. Autonomous Behavioural Agents

The first environment *basic-1-tutorial* consists in a fully-observable world of same color packets with only one destination. This environment requires a simple baseline model that can pickup and deliver packets without worrying about exploration, energy requirements or communication. (see IV-A1)

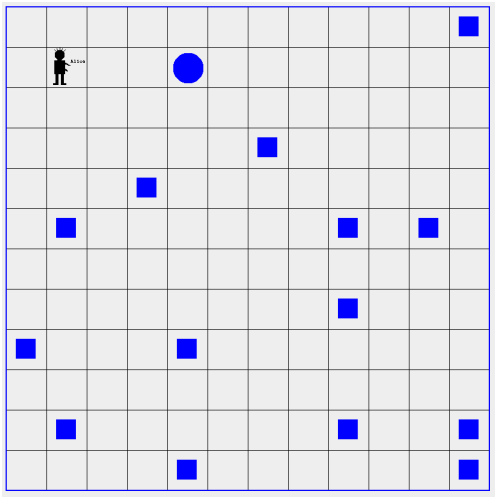

Fig. 1. *basic-1-tutorial*

The second environment *basic-2-vision* differs from *basic-1-tutorial* by three characteristics: it is a multi-agent environment (two agents) with a partially-observable world and with two destinations. By restricting the agents' view-range (depicted by a blue rectangle), they are required to explore the environment. (see IV-A2)

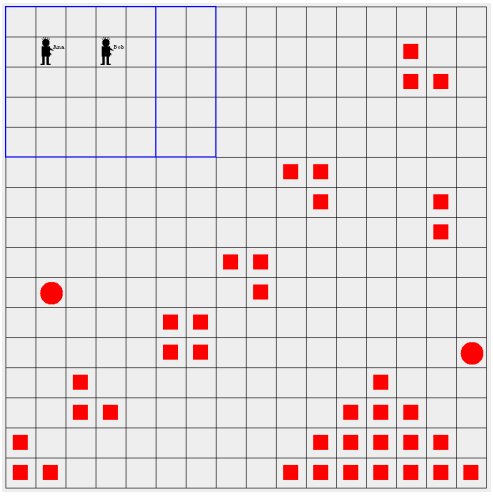

Fig. 2. *basic-2-vision*

The third environment *basic-3-color* differs from *basic-2-vision* by the presence of packets and destinations of four different colors. It therefore demands that the agents perceive (and memorize) the color of these items such that they pick up and deliver the right item to the right location. (see IV-A2)

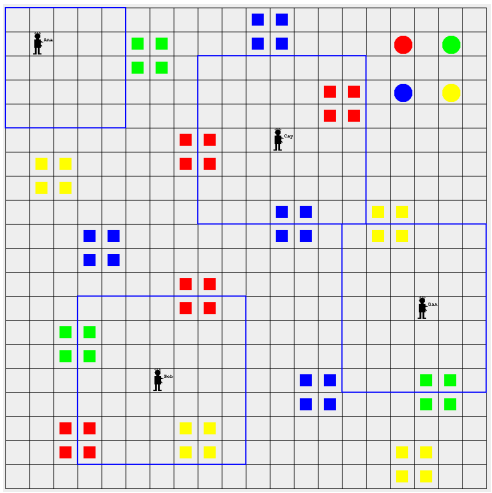

Fig. 3. *basic-3-color*

The fourth environment *basic-4-complexity* differs from *basic-3-color* by the presence of packet generators and solid walls. Packet generators require effective exploration as a packet could appear even after the local region has been emptied out from any packets. The solid walls block the agents' moves and restrict their view-range. Another requirement therefore is the ability to effectively find short paths to pick up and deliver packets despite the presence of walls. (see IV-A3)

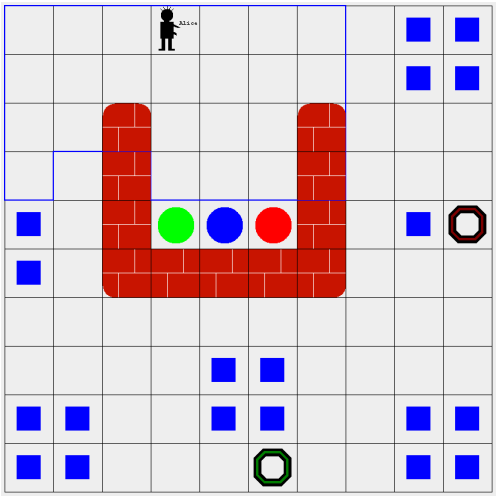

Fig. 4. *basic-4-complexity*

B. Energy Management

In the environments *energy-1* and *energy-2*, an energy cost is now associated to each action as can be seen in table I.

TABLE I
ENERGY COST OF AGENTS' ACTIONS

Step	10
Step with carried packet	20
Other action (pick packet, skip, ...)	5

Each agent can only perform a limited number of actions before it needs a battery refill at one of the energy stations. An agent is recharged by an additional 100 energy units each turn spent at an energy station up to a battery limit of 1000 units. Only one agent at a time per energy station can fill up its battery. The efficiency of the system is measured in the amount of global energy consumed to deliver all packets.

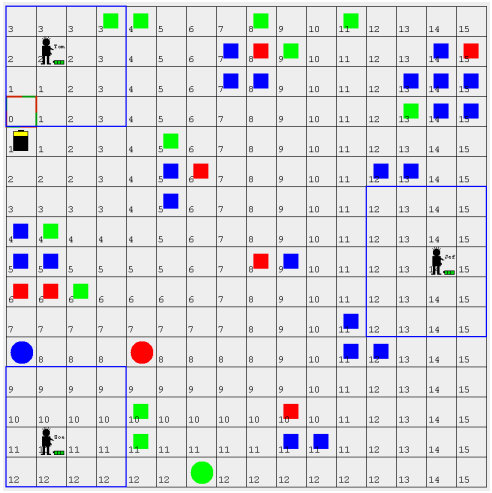

Fig. 5. *energy-1*

The main challenge when moving from *energy-1* to *energy-2* lies in the increasing amount of agents (3 vs 6), energy stations (1 vs 2), colors (3 vs 4) and the size of the world.

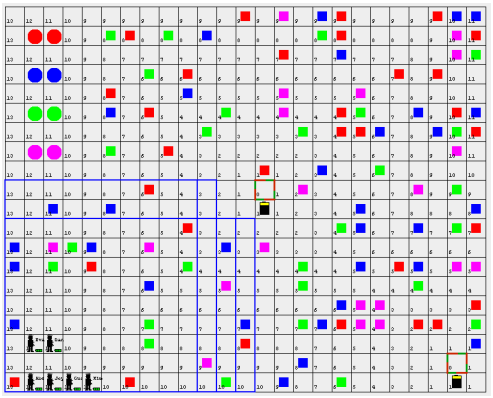

Fig. 6. *energy-2*

The focus should be to ensure effective coordination when recharging. The number of energy stations is limited compared to the amount of agents and each agent's battery has a limited capacity. Despite their narrow direct communication range (i.e. restricted to their view-range), agents should coordinate their arrival to the chargers so that they do not run out of battery. (see IV-B1)

C. Task Delegation

The final challenge involves collaboratively solving an environment which is impossible to handle in a single-agent

paradigm. Whereas before there were only black agents, agents can now be assigned a color which restricts the packages they can pick up to those of the same color. Glass walls are also introduced. They block agents' moves but not their view-range.

In the environment *task-delegation*, there are three different colors, with one agent and one destination for each of them. The paths to these destinations are blocked by packages of different colors. Agents can only get to their destination if they collaboratively free the path to it. (see IV-C1)

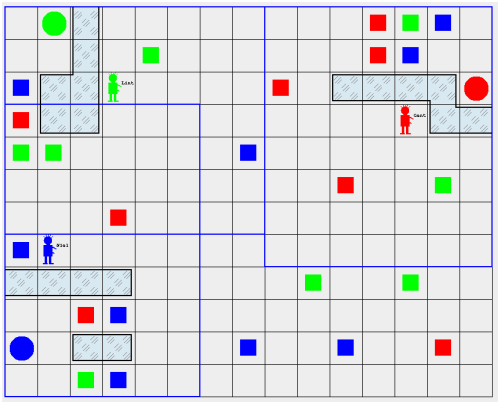

Fig. 7. *task-delegation*

IV. Solution

We will now describe our solutions to the test environments presented in the previous section. Referring to the map of design patterns in multi-agent systems discussed in [12], our solutions can be placed under the dimension *Focus*, the category *Behavioural* and the subcategory *Interactions*.

A. Autonomous Behavioural Agents

1) Moving Randomly: In the case of battery-free agents, an easy solution to solve most environments is to make agents move around randomly. Picking up packets and delivering them as they come across a matching destination. However, the ease of implementation of this solution comes at the cost of inefficiency and inconsistency from one run to the other. The algorithm (which we will be calling *basic* form now on) can serve the purpose of baseline to benchmark other implementations.

Exploration via full randomness is here at the core of the algorithm. Optimal decision-making is simply ignored. Despite that, it provides a very robust solution in any environment where efficiency and collaboration are not a strict condition for success. We started our work from this interesting perspective with the aim to transfer this property to more efficient algorithms.

basic can solve the following environments: *basic-1-tutorial*, *basic-2-vision*, *basic-3-color*.

2) Explore, Memorise and Greedy Pick-Up and Deliver: The implementation *distance_greedy* demands that each agent memorizes the parts of the environment that it sees in its individual memory and keeps it up to date as it moves.

To explore an environment, agents pick a direction and continue along it until they hit an obstacle. They then pick a new direction as close as possible to their previous one in Manhattan distance. This behaviour ensures that agents maximise their chances to explore the four corners of the environment as early as possible if no packets and matching destinations can be seen on the way. It is however a deterministic procedure that might fail under some configurations where obstacles would force an agent to cycle through the same list of locations. We can alleviate this shortcoming by adding some randomness in the procedure (eg: choose a random move 20% of the time). Note that exploration time is greatly minimized compared to a full random exploration and yet semi-randomness preserves the robustness property mentioned in IV-A1.

If an agent has some packets in memory and at least a destination of the same color is known, it will use this stored data and move to the closest packet and then to the matching destination which is the closest. As agents can move diagonally, the distance measure is chosen to be the Chebyshev distance.

$$d_{chebyshev}(x, y) = max_i(||x_i - y_i||)$$

for $i \in \{0, 1\}$ in a two-dimensional space.

Despite fostering a selfish behaviour, *distance_greedy* pushes each agent to focus on solving the local part of the environment it is in before moving to unexplored parts where other agents are potentially already active.

This solution solves the following environments much more efficiently: *basic-1-tutorial*, *basic-2-vision*, *basic-3-color*. This implementation achieves indeed the lowest number of cycles and energy costs out of all our implementations on these three environments as reported in table II and III

As depicted in figure 8, the whole process can be divided into two behaviours: Pickup (explore if no packet-destination pair in memory. Otherwise, move to a packet for which a destination is known) and Deliver (carry a packet until dropped to its destination).

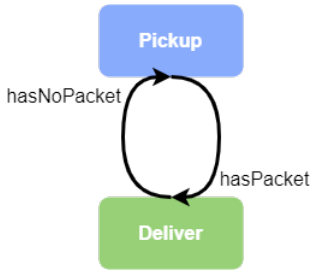

Fig. 8. Pickup-Deliver behavioural diagram

While moving around, each agent update its belief with the perception of its surroundings. This is done in exploration, pickup and delivering phase. Strict exploration occurs only when no packet-destination pair exists in an agent's memory.

3) Multi-Targets A:* In this solution, each agent stores the walls it encounters on top of the packet and destination representations.

If an agent has some packets in memory and at least a destination of the same color is known, this implementation will use the cells stored in memory to find the shortest path between the current agent and the closest packet and then between this packet and the closest matching destination. Efficient path search is performed with the algorithm A* [15] which was modified to find the shortest path to the closest target among multiple targets, hence the name of the implementation: *a_star_multi_target*.

A* works by implementing a priority queue of states to visit. The algorithm starts from the initial state the agent is in and progressively adds the legal states available from the current state. In our case, a state is defined as the coordinates of each cell, e.g. $n = (x_0; y_0)$. The $(width \times height)$-rectangular grid of the environment has thus in total $width \times height$ states. The priority function $f(n) = g(n) + h(n)$ that orders the states in the queue is the sum of two score functions: the cost $g(n)$ and heuristic $h(n)$. On the one hand, the cost function computes the backward cost of the path from the current state to the initial state. In our case, this corresponds to the length of the current path.

$$g(n) = length(path(n))$$

On the other hand, the heuristic function estimates the forward cost, i.e. the cost left to arrive at a terminal state. To ensure that A* finds one of the least-cost paths, the heuristic function should be admissible. In other words, it should never overestimate the cost left to get to the goal. As agents can move diagonally, our heuristic is based on the Chebyshev distance between the current position and the terminal position (i.e. a packet or a destination location). It will never be greater than the actual distance that the agent has to travel. To be more exhaustive, the heuristic is the minimum Chebyshev distance among multiple possible targets.

$$h(n) = min_{t \in T}(d_{Chebyshev}(n, n_t))$$

where T is the set of indices of terminal states. The set of terminal states correspond in pickup phase to the position of all memorised packets for which a destination is known and in delivering phase to the position of all memorised destinations of the same color as the carried packet.

We could even prove that this heuristic is consistent in which case it would ensure that A* is guaranteed to find an optimal path without processing any state more than once. We would thus have a time complexity of $O(width \times height)$. To be consistent, the heuristic should satisfy the additional condition

$$h(n) \leq c(n, a, n') + h(n')$$

for $n' \in N'(n)$ and where a is an action/move ($a \in \{(1, 1), (-1, 1), (1, -1), (-1, -1), (0, 1), (1, 0), (-1, 0), (0, -1)\}$); $c(n, a, n')$ is the true cost for the agent to move from state n to the adjacent state n' following the relative move a; $N'(n)$ is the set of neighboring legal states to n. A legal state is defined as the coordinates of a free cell that remain within the world's borders.

To move from state n to n', the true cost $c(n, a, n')$ will always be 1 as they are adjacent states. If we evaluate $h(n) - h(n')$, three scenarios are possible: either we move closer to the terminal state by 1, or by 0 (e.g.: if the optimal move is going North-West and we go either North or West instead) or by -1 (we actually moved further from the terminal state). We thus have that $max_{n' \in N'(n)}(h(n) - h(n')) = 1$ and $c(n, a, n')$ is always 1. The consistency condition is thus met.

As reported in table II and III, *a_star_multi_target* achieves almost similar results to *distance_greedy* in the following environments *basic-1-tutorial*, *basic-2-vision*, *basic-3-color* and manages to solve the additional *basic-4-complexity* much more effectively than *basic*. Fundamentally, we have not changed the behavioural diagram and the exploration approach from *distance_greedy*. But we have optimized the exploitation process by making use of more memorised information to compute short paths: cells with walls, packets and destinations that could block the greedy path.

B. Energy Management

1) Semi-Random Greedy Charging & Multi-Targets A:* We now move to environments where the energy cost of each action is described as in table I. The implementation *charging_greedy* allows to handle that thanks to two criteria: agents arrive with a safe margin of energy left at the energy stations and they do not come all together at once.

The first criterion can be encouraged by forcing the agents to only move to an energy station when they are not carrying any packet. The detour to a battery charger is then less costly in energy. An agent should thus estimate if he has enough energy for the whole trip to a packet, then to its destination and then to the energy station or if he should instead directly go to the energy station.

It is also chosen in our case to rely on an indirect communication middle: gradient fields emitted by energy stations. Each cell of the environment is thus assigned a gradient value (see figure 5 and 6) that gives the number of steps required to move to the closest energy station. Gradient fields are a nature-inspired approach that is further described in [14].

To encourage the second criterion, we could have chosen to rely on direct communication between the agents but it is limited by their view-range. We introduced instead a random term in the condition to start a trip to a battery station. This term chooses an energy value at random within a predefined range of values. The range is environment-specific. It aims at forcing agents to initiate their trip to an energy station at different random moments such that battery stations are rarely overcrowded and when it happens, the agents still have enough energy to wait for their turn.

We always let the agents recharge to the fullest to ensure they do not need to come back too often to the station.

The behavioural diagram is extended with two new behaviours: FindCharger (move to an energy station), Charge (skip turns while charging).

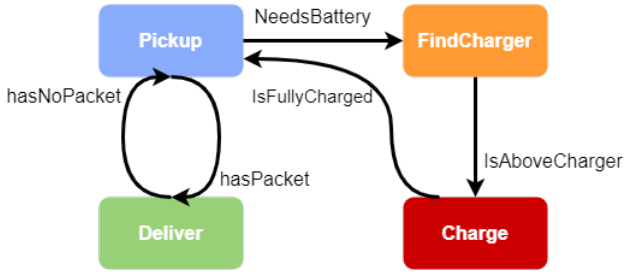

Fig. 9. Pickup-Deliver-FindCharger-Charge behavioural diagram

As shown in table II and III, this implementation solves the following environments efficiently though not as much as *distance_greedy* and *a_star_multi_targets*: *basic-1-tutorial*, *basic-2-vision*, *basic-3-color*, *basic-4-complexity*. And it manages to tackle the additional environment *energy-1*, *energy-2*. It is furthermore the least costly implementation in terms of cycles and energy on the environment *energy-2*.

C. Task Delegation

1) Gather Packets & Multiple-Consecutive-Targets A:* In the environment *task-delegation*, the main challenge is to unblock the path to the destinations. This has to be done in collaboration as each agent can only take hold of the packets of its color. The solution implemented in *task_coordination* consists in gathering the same color packets together while exploring the environment, as long as a path to a destination for these packets is not free.

Concretely, if the agents can not find a path to the packet and from the packet to its destination with A*, they are asked to group the packets in a gathering zone. This zone is a circle where the center is the average of the position of all known packets of the same color. The center will thus change as we localize new packets. We call that a moving center. And the radius, defined in Chebyshev distance, is a parameter to choose depending on the number of packets of the same color. We typically set the radius to 2 or 3. If all agents try to gather their packets together in this way, they will naturally unblock all paths.

Note that an unfortunate situation could happen where gathering the packets together builds up a wall that blocks the paths to the destination. We strongly minimize this risk by choosing a radius not too small that leaves some gap cells between the packets. The moving center definitely helps as well. Indeed, the gathering zone moves as time passes and as packets are discovered, thus unblocking the way. This procedure relies on some form of randomness as packets can be discovered during the exploration phase which is semi-random. Randomization ensures once again the robustness of our solution.

This gathering solution relies also on an improvement of the algorithm A* as now the shortest path to two consecutive targets among multiple targets is computed: from the agent's coordinates to the optimal packet and from this packet to the optimal destination such that the total distance is the shortest among all possible combinations of consecutive targets. The idea comes from the work of Grenouilleau et al. [7] that extends A* precisely such that paths with multiple ordered goals can be computed, noticeably to allow an optimal pickup and delivery of packets at the same time. Although the actual implementation is quite complex, the essential improvement lies in the formulation of the heuristic function.

$$h(n) = min_{t_1 \in T_1, t_2 \in T_{1,2}} \big(d_{Chebyshev}(n, n_{t_1}) + d_{Chebyshev}(n_{t_1}, n_{t_2}) \big)$$

where T_1 is the set of indices of first terminal states. The set of first terminal states contains the position of many packets for which a destination is known and $T_{1,2}$ is its counterpart for second terminal states. The set of second terminal states $T_{1,2}$ contains the position of all memorized destinations of the same color as the packet chosen in T_1. A* finds a path when a first and second terminal states are reached consecutively. Note that admissibility and consistency are preserved and can be proved following the same process as in IV-A3.

The behavioural diagram is extended with one new behaviour: Gather (move a carried packet to the gathering zone).

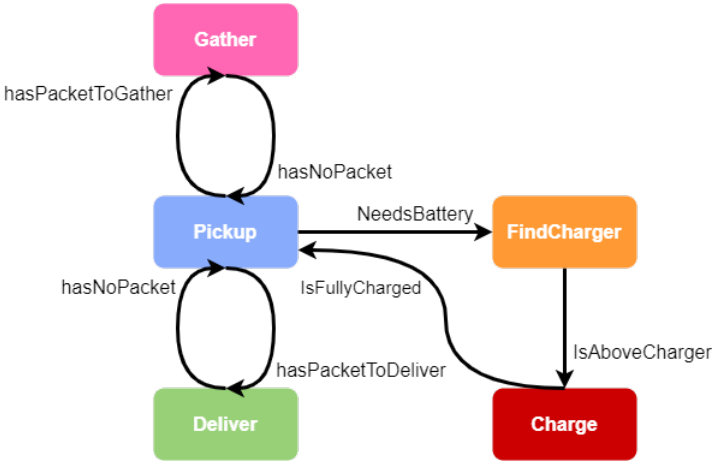

Fig. 10. Pickup-Gather-Deliver-FindCharger-Charge behavioural diagram

task_coordination can effectively solve all environments. When looking at table II and III, it is on a par with *distance_greedy* for the environment *basic-1-tutorial*. It achieves a lesser performance in especially the environment *basic-2-vision* and *energy-2* but obtains the lowest energy consumed and number of cycles in *basic-4-complexity* and *energy-1*. It additionally solves our last test environment *task-delegation*.

V. Discussion

In *basic*, an agent wanders around randomly until it finds the correct destination. We first noted that full random moves allowed us to systematically solve the four first environments.

There is an intrinsic property to randomization that allows us to handle even difficult problems such as the presence of walls, we call that robustness.

We systematically transferred this property to the exploration phase in all further implementations by making it semi-random. Randomness is introduced to break never-ending cycles as mentioned in IV-A2 and ensures thereby that all packets will be eventually seen. In *distance_greedy*, memory also plays an important role in minimizing the exploration time, the least effective phase as the agents are not able to make optimal decisions due to a lack of information. This approach has drastically improved energy consumption compared to the algorithm *basic*.

The main contribution of A* is to allow the system to more efficiently solve path finding for the pickup and delivery of packets, where a straight line path is not possible. This algorithm is thus very useful when the environment contains many obstacles, e.g.: walls. A* computes the shortest path, in the parts of the environment that were previously explored, between the agent and the closest packet and then between this packet and the closest matching destination. This behaviour ensures that each agent focuses on solving the local part of the environment it is in, before moving to unexplored parts where other agents are potentially already active. It lacks however some form of cooperation and delegation to spare some more cycles.

In the energy environments, an interesting concern came from the limited view range of the agents making coordination by direct communication challenging especially in large environments. Given that there are very few energy stations, the main problem is to prevent the agents from waiting too long near the stations. They would otherwise lose time and potentially run out of battery in the queue. We concentrated our efforts on preventing that agents arrive together at the station. Part of the solution is achieved by precisely estimating the energy needed to go to the station and to undertake a pickup and delivery task such that agents foresee the need for a refill before venturing into a packet delivery. But despite that, agents tend to arrive at the stations together. To trigger the time at which an agent should move to an energy station, *charging_greedy* adds a random term that can be tuned for environment specific needs. It lowers the probability of overcrowded energy stations. Although a random move is not optimal at the individual level, it naturally allows a better coordination of the agents at the energy stations and a better overall solution at the multi-agent level. Nevertheless, charging stations can still end up being overcrowded, especially if the random term is not properly tuned. Prioritising low-battery agents in the queue over others via direct communication could prove to be a beneficial complement to our solution.

In the environment *task-delegation*, the last implementation ensures that each agent gather their packets together when the corresponding destination is not reachable yet. Gathering packets by color is a fast solution where we fully benefit from task delegation between multiple agents. This approach naturally unblocks the path to the destinations but as a side effect, it pushes all packets towards the center of the world. It can therefore make it difficult for agents to travel. Fortunately, by asking the agents to explore in a semi-random way, they discover new packets at random times during the execution. As explained in IV-C1, it causes the center of the gathering zone to move. Consequently, the agent responsible for the packets of this gathering zone has to move them again. And despite being a less efficient process at the agent level, it allows to unblock other agents that are stuck between packets. It also frees different paths during the execution that may be more optimal to transport certain packets. And it ensures overall that the environment is solvable. Once again, the benefits at the multi-agent level outweighs the drawbacks at the agent level. To move efficiently in this dense environment, A* computes now the shortest path to consecutive targets. We test thereby every combination of matching packet and destination.

The additional patterns designed in each of these implementations help to solve more and more complex environments. From table II and III, it is however interesting to note that the final algorithm that we developed, *task_coordination*, is not a one-size-fits-all.

In particular, we found out that the greedy option *distance_greedy* was particularly efficient on the three first environments. When we face more complex tasks, we nevertheless see the need for more advanced behaviours and agent collaboration.

We can also point out that *task_coordination* is more effective on the smallest and less complex energy environment *energy-1* while *charging_greedy* dominates when the problem scales up in *energy-2*. We argue that this is mostly due to the fact that one major drawback of relying on A* is that each agent plans ahead its path and if the dynamic environment evolves a lot, the path might not remain optimal. In particular, *task_coordination* relies even more on reliably foreseeing the future as our last A*-variant plans ahead a full path from the agent's coordinates to a packet and destination. Although it is flexible enough to recompute the plan if the path gets blocked, it still benefits from a less dynamic environment with less agents.

VI. Related work

In the field of MAS a lot of papers have been written about solutions to similar problems. These papers provide an interesting background to this one.

"Multi-Agent Coverage Search in Unknown Environments with Obstacles: A Survey" [6] is a paper comparing multiple algorithms used in multi-agent systems both for static as well as dynamic environments. Most of these algorithms also include elements of randomness, such as the wall follower agent which randomly leaves the wall to make sure that the whole environment is explored.

"Agent-based approaches for exploration and pathfinding in unknown environments" [5] uses an approach where multiple agents use different exploration algorithms to cope with the diverse nature of possible environments as well as to deal with different, sometimes contradictory, goals.

The work of Nissim et al. [8] implements A* in such a way that, as it scans through the states in the queue, a form of cooperation through messages between agents allows relevant states to be shared.

Apart from the work on other problems there are also a lot of papers about solutions to the Packet-World itself.

"Implementing Multi-Agent System behaviours for overcoming energy constraints and obstacles in The Packet-World" [10] is an interesting paper because it uses the same environments as this paper (more precisely environments *basic-3*, *energy-2* and *task-delegation*) which allows for direct comparison with our work. The paper noticeably mentions a lot of randomization in the exploration phase. A key difference is that it does not rely on the algorithm A*. This seems to cost them a couple 100 cycles on every environment.

The paper *"A* Based Approach to a Situated Multi-Agent System in the Packet-World"* [11] proposes a very similar approach to the problem of this paper. However, the paper uses a different strategy when it comes to the random exploration. Their agents move randomly when exploring but will try to keep going on average in the same direction.

VII. Conclusion

Overall our research has focused on three main aspects of multi-agent systems: autonomous behaviour of agents, managing a limited energy supply and collaboration with other agents. For each of these domains, one or more test environments depicting different problem settings were used to validate the proposed implementations. All our implementations include some form of randomization to robustly solve these challenges.

The first domain is the automated movement of agents. Here our first idea was to test pure random movements. This is improved upon by the semi-random exploration to stop agents from walking back and forth. The final improvement occurs through the use of A* as a path finder algorithm when the start and end points are known.

The second domain focuses on energy management. All agents know exactly how far the nearest charging station is thanks to the emitted gradient fields. They can always predict how much energy is needed to get there. They can also predict from A* how much energy is needed to perform a packet pickup and delivery. But to avoid that all agents arrive at the same time to charge, their trip to the station is triggered semi-randomly within a certain interval of the battery level.

The last domain is the one of task delegation. Here, agents gather their packets together as long as the path to their destination is not free. The center of the gathering zone is calculated based on the known packet locations. This way, as the agent semi-randomly explores the environment, this center moves forcing agents to shift their gathered packets. Even though this is a less optimal solution at the agent level, it does free up paths that might be accidentally blocked by gathered packets. The benefits at the multi-agent level outweigh the drawbacks at the agent level.

Let us now come back to our research question: how can randomization be used to align multiple agents' exploration and exploitation objectives in a partially-observable environment? Randomization can be used to reduce the complexity of the individual agent behaviours and the interactions between them. It enforces robustness which we define as the property of consistently being able to solve even complex environments. It does have to be complemented by other algorithms and efficient optimizations, such as A* to find the optimal path between two points. Randomization can be used at different phases: for exploration, coordination and task delegation.

Some of the solutions for the tested environments include parameters like the radius of the gathering zone. These are environment-specific and could have a large impact on the performance of the system in differing environments. Furthermore, the tuning of these parameters manually to every specific environment could hamper the versatility of such systems in real-world applications. Therefore, future work would be of interest to include some form of automatic tuning through learning algorithms so that agents would be operational in different and dynamic environments without losing in efficiency.

Another challenge that remains with semi-random movements is that agents could block each other, especially in environments with narrow walkways. Future research to incorporate some sort of communication between agents to alleviate this problem would also be beneficial.

References

[1] Weyns. (2010). Architecture-Based Design of Multi-Agent Systems (1. Aufl.). Springer-Verlag. https://doi.org/10.1007/978-3-642-01064-4

[2] Weyns, Danny & Helleboogh, Alexander & Holvoet, Tom. (1970). The Packet-World: A Test Bed for Investigating Situated Multi-Agent Systems. 10.1007/3-7643-7348-2_16.

[3] Cossentino, Lodato, C., Lopes, S., & Ribino, P. (2011). Multi agent simulation for decision making in warehouse management. 2011 Federated Conference on Computer Science and Information Systems (FedCSIS), 611–618.

[4] Kato, Takumi & Kamoshida, Ryota. (2020). Multi-Agent Simulation Environment for Logistics Warehouse Design Based on Self-Contained Agents. Applied Sciences. 10.3390/app10217552.

[5] Becker, Matthias & Blatt, Florian & Szczerbicka, Helena. (2012). Agent-based approaches for exploration and pathfinding in unknown environments. 1-4. 10.1109/ETFA.2012.6489771.

[6] Xu, X., Yang, L., Meng, W., Cai, Q., & Fu, M. (2019). Multi-Agent Coverage Search in Unknown Environments with Obstacles: A Survey. 2019 Chinese Control Conference (CCC), 2317–2322. 2019, pp. 2317-2322, doi: 10.23919/ChiCC.2019.8865126.

[7] Grenouilleau, F., Hoeve, W.-J. van, & Hooker, J. N. (2021). A Multi-Label A* Algorithm for Multi-Agent Pathfinding. Proceedings of the International Conference on Automated Planning and Scheduling, 29(1), 181-185.

[8] Nissim, Raz & Brafman, Ronen. (2012). Multi-agent A* for parallel and distributed systems. 1265-1266.

[9] Zafar, & Baig, A. R. (2010). Optimization of route planning and exploration using multi agent system. Multimedia Tools and Applications, 56(2), 245–265.

[10] Geens J., Ignoul J., Lenaerts W., Goossens E. Implementing Multi-Agent System behaviours. for overcoming energy constraints and obstacles in The Packet-World

[11] Celis T., Decoster T., Fridrichova P., Schroijen J. A* Based Approach to a Situated Multi-Agent System in the Packet-World.

[12] Juziuk, J., Weyns, D., Holvoet, T. (02 2014). Design Patterns for Multi-agent Systems: A Systematic Literature Review. Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks, 9783642544323, 79–99. doi:10.1007/978-3-642-54432-3_5

[13] Weyns, D. (2009). A pattern language for multi-agent systems. 2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture, 191–200. doi:10.1109/WICSA.2009.5290805

[14] De Wolf, T., Holvoet, T. (2006). Design Patterns for Decentralised Coordination in Self-Organising Emergent Systems. Proceedings of the 4th International Conference on Engineering Self-Organising Systems, 28–49. Hakodate, Japan. Berlin, Heidelberg: Springer-Verlag.

[15] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107. doi:10.1109/TSSC.1968.300136

TABLE II
THE TOTAL ENERGY CONSUMED FOR DIFFERENT IMPLEMENTATIONS AND ENVIRONMENTS CALCULATED OVER A BATCH OF 10 RUNS

	basic	*distance_greedy*	*a_star_multi_targets*	*charging_greedy*	*task_coordination*
basic-1-tutorial	55 590	**2 154**	2 280	2 620	**2 160**
basic-2-vision	83 888	**5 504**	5 738	5 950.5	6 644.5
basic-3-color	1 228 557	**21 541.5**	21 919	23 966.5	21 893.5
basic-4-complexity	551 894.0	/	19 169	19 668.5	**18 113.0**
energy-1	/	/	/	19 879.5	**17 395.0**
energy-2	/	/	/	**67 172.5**	87 581.5
task-delegation	/	/	/	/	**15 242.5**

TABLE III
THE TOTAL NUMBER OF CYCLES FOR DIFFERENT IMPLEMENTATIONS AND ENVIRONMENTS CALCULATED OVER A BATCH OF 10 RUNS

	basic	*distance_greedy*	*a_star_multi_targets*	*charging_greedy*	*task_coordination*
basic-1-tutorial	3 154	**163.4**	170	187	**164**
basic-2-vision	2 697.2	**220.8**	227.1	234.0	260.5
basic-3-color	16 685.1	**409.2**	414.4	443.5	422.2
basic-4-complexity	27 550	/	1050	1 102	**1 007.9**
energy-1	/	/	/	590.9	**514.4**
energy-2	/	/	/	**991.77**	1452.2
task-delegation	/	/	/	/	**420.6**