EasyChair Preprint

№ 5578

Collaborative Situated Agents for Baseline Logistics Problems on the Packet-World Testbed

Georgios Kouros, Gregoris Malekos, Bolat Tleubayev and Victor Van Wymeersch

May 20, 2021

Collaborative Situated Agents for Baseline Logistics Problems on the Packet-World Testbed

Georgios Kouros, Gregoris Malekos, Bolat Tleubayev, Victor Van Wymeersch
Faculty of Engineering Science, KU Leuven
Leuven, Belgium
georgios.kouros@student.kuleuven.be, gregoris.malekos@student.kuleuven.be,
bolat.tleubayev@student.kuleuven.be, victor.vanwymeersch@kuleuven.be

***Abstract*—Multi-Agent Systems (MAS) enable the efficient solution of complex real world problems with flexibility and robustness to system perturbations. However, the implementation of these systems is nontrivial and requires careful consideration into software design principles, in support of the attractive properties of MAS. To this end, three key problems within MAS have been solved in this paper, namely autonomous navigation, energy management, and multi-agent coordination and cooperation. Grid search, gradient methods, and protocol-based communication are proposed as a means to enable autonomous situated agents to tackle the aforementioned tasks. The solutions are implemented on *Packet-World (PW)*, a testbed for experimentation with situated MAS, meant to be used to investigate and evaluate different perception, decision making, and communication aspects among agents that are situated in a virtual environment. The proposed solutions are evaluated with regard to total energy consumption and/or number of cycles for completing a run averaged across batch runs.**

***Index Terms*—multi-agent systems, Packet-World, situated agents, grid search, gradients, protocol-based communication**

I. Introduction

Increasingly complex industrial systems require flexible, robust, decentralized dynamic control solutions. Multi Agent Systems (MAS) are software systems that provide a solution fulfilling all of these requirements [1]. These systems are composed of multiple singular agents that work together to solve intricate problems beyond the scope of a single agent's capabilities. To develop these system solutions it is pertinent to be able to test MAS research and developments before they are applied. To this end a open-source development platform *Packet World* is used for the development of these MAS. Here the agents are required to move around in a grid-based world, communicate and coordinate to pick up and deliver various packages to defined locations [2].

This paper focuses on the development of techniques to control agent behaviour, coordination of an agent energy management systems, and communication between multiple agents.

Section II gives an introduction to PW, its components and general architecture. Section III contains a description of the problems that this assignment is aimed to solve, while Section IV presents the proposed solutions. The paper concludes with a listing of related work and a final discussion in Sections V and VII, respectively. For clarity discussions on future work or possible implementation improvements are located within the individual parts of Section IV.

II. Packet World

Packet-World [2] with regard to its pattern language on the domain of *situated* multi-agent systems and the various aspects of this pattern language are presented in the following section.

A. Packet World Concepts and Characteristics

PW is based on the Situated Agents pattern that promotes module reuseability and decoupling of independent processes. *Agents*, *ongoing-activities*, and the *environment* are the three main abstractions of this system. The environment is observable by the agents and contains processes that regulate states and interactions. The agents can act independently based on their local perception of the environment and goals. They can communicate with each other through messages to form collaborations to jointly complete tasks when needed.

The environment of PW is a multi-layered grid, where each layer is assigned to a different entity (eg. agent, packet, flags) and has its own constraints (eg. one packet per cell). The state of the world consists of the combined grid layers and agents perceive only the layers that are relevant to them.

The perception module maps the local state of the environment to a percept for the agent. Percepts change the internal knowledge held by the agent of it's environment. Percepts are provided by three subsystems, sensing, interpreting and filtering. Sensing allows the agent to focus on a specific perception ability to gain information of the world, for example through sight or smell. Sensory information is translated into percepts through internal knowledge descriptions of the world. Finally, percepts can then be filtered to exclude unnecessary information not relevant to the agents' intended search.

From local knowledge of the world, agents make decisions on which actions to perform to achieve goals with minimal energy requirements. This decision making process is represented using a free-flow tree. Role and situated commitment extensions enable social behavior. The tree nodes receive information internally, through the agents' knowledge and externally from perceptions. Nodal activity is passed down the tree to leaves where an action is selected. In the free-flow trees agents have sub-trees dedicated to cooperation-, individual-,

and battery recharging-based roles. Commitments define the relationships between roles.

Communication happens with the help of protocol steps, which are tuples (conditions, effects). Some conditions must be satisfied for an agent to send a message and some effects result from the protocol step being applied. There is a finite number of different messages which the agents can send to one another. Via those messages they can request the help of other agents to solve tasks cooperatively and answer other agent's requests for help. Information relevant to the tasks can also be shared in this way.

Advanced agent collaboration has been built into PW. Agents using explicit and implicit messaging can collaborate in many ways, such as letting other agents know where packets/goals are or setting up chains to move packets along faster. Some tasks, such as moving large packets, specifically require agent cooperation. For these tasks, messaging allows agents to collaborate in a timely manner.

B. Real-world applications

In this subsection some potential future industry applications of MAS are laid out in order to outline the relevance of MAS research to real-world problems.

In a *MAS of autonomous taxis* serving a city, each taxi is represented as an agent that is tasked with picking up and dropping off passengers. Taxis have to work together to transport large groups of passengers that sometimes might not fit into one taxi. The goal is to transport as many passengers as possible within the day while minimising total travel distance and fuel consumption. Gradient fields can be used to attract agents to areas with higher demand, or for refueling purposes.

A MAS of waiters serving people in a restaurant. The layout of the restaurant can be modelled within the PW grid with waiters acting as agents. Agent tasks may include taking orders, serving food, cleaning tables, represented as differently colored packets, or even cooperation for service of a big table of customers.

III. Problem Descriptions

A. Autonomous behaviour of single agents

Within PW individual agents are, among several others, required to be able to autonomously perceive the environment, navigate the terrain, pick up and drop off packages, move to recharge stations when their battery level is low, coordinate and collaborate with other agents etc. The most fundamental of these responsibilities is the ability to independently, as a single agent, perceive and navigate the environment to deliver packages to the drop-off points/destinations. The software architecture defining the agents' autonomous behaviour needs to achieve this goal in the most effective way while maintaining the benefits of multi-agent systems in general, such as flexibility and adaptability. To this end, the aim is to program optimised agent behaviours using the well-established architectural software pattern of situated agents.

B. Coordination of agents to manage energy

Creating agents that eventually are able to perform all the tasks required of them in PW does not pose a great challenge. A more challenging goal is to introduce energy constraints and program agents that are able to perform the tasks while ensuring no agent runs out of energy. To do this, the agents' behaviour when tackling their tasks must be optimised to avoid unnecessary energy wasting. Additionally, it is required to set up a system for agents to coordinate and manage energy consumption/charging. In MAS, agents coordinate and communicate directly with each other, avoiding the large overhead costs of being coordinated by a central authority which would keep track of everything and introduce a single point of failure in the system. There are two main methods for agents to coordinate [2]. The first, which is explicit, is direct messaging with requests/answers between agents. The second, which is implicit, is the use of gradient fields within the PW environment which inform agents of the existence of goals/charging points to them. In both cases the aim is for the agents to act as a coordinated unit rather than a sum of individuals competing to complete the same tasks.

C. Collaboration of agents with each other

The assignment of tasks in a MAS can be complex and difficult to manage. Often tasks cannot be completed by individual agents without the help of other agents. Moreover, agents often operate in large dynamic environments where the number of agents able to do work changes along with their goals. Due to this complexity and scale, centralised task assignment lacks the flexibility needed in MAS where the ability to rapidly switch task assignments between agents for optimal coordination and execution on a system level is crucial. Various methods exist to tackle this problem of task assignment, namely environment-based, and protocol-based. Environment-based task assignment involves populating agent environments with various gradient fields to attract them to goals and repulse them from each other. Protocol-based task assignment is more focused on messaging, where tasks send out requests for work towards all agents in the system and the agent response with the best proposal is selected for the task. Both methods allow for task reallocation, either by the update of gradient fields, or through new task requests allowing the methods to cope with variations in the system.

IV. Problem Solutions

A. Autonomous behaviour of single agents

Agent behaviours were tackled in three distinct steps, building up in complexity. Starting from the aimless random wandering behaviour, agents were programmed to pick up and drop off packets should they reach the appropriate tiles. These are tiles adjacent to either packets to pick-up or drop-off destinations for packet delivery. In the second phase we refactor the wander behaviour according to the situated agent architectural pattern as shown in figure 1. In particular, the refactoring included the distribution of behaviours to distinct classes and the development of behaviour change classes that

switch from one behaviour to the other in a cyclical fashion as seen in figure 2. This transitioning between behaviours is described in more detail in section IV-A1. In the third and final phase, the wandering behaviour of the agents is optimised to improve packet delivery time and energy consumption through goal-oriented navigation. Specifics of this improvement are covered in section IV-A2 below.

1) Design: The design selection process involved analysis of two types of MAS architectures, namely Belief-Desire-Intention (BDI) by Rocha et al. [1] and Situated Agents [2]. We compared how would the BDI fit to our problem against the Situated Agent architectural pattern. One advantage of the BDI pattern was its focus on distributed autonomous behaviour. However, in the end, we settled on the Situated Agent architectural pattern [3] due to its innate suitability to Packet-World and its capabilities for agent-to-agent communication and coordination and agent independence(agents maintain their own behaviour and state). Also, its distributed architecture comprised of knowledge, perception, decision making, and communication components in each agent, enabled us to easily create, test, and assess increasingly more complex behaviours.

The design of the software architecture for the agent behaviour follows the form as presented in Figure 1. In this free-flow tree of a single there are two distinct behaviours, *GetPacket* and *DeliverPacket*.

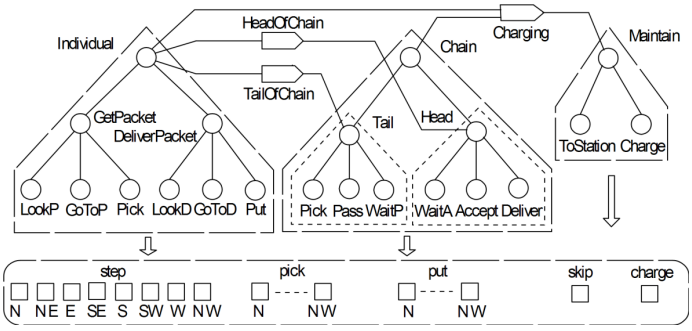

Fig. 1. Action selection free-flow tree of situated agents. Original diagram from Weyns et al. [2]

Concretely the behaviour change pattern consists of two distinct classes *BChangeGet2Del* and *BChangeDel2Get* to change agent behaviours from getting packets to delivering packets and visa-versa, based on the conditions presented in Table I. This software architecture pattern improves agent behavioural flexibility, allowing for additional behaviours such as "Recharge" to easily be incorporated into the existing code structure in the future. Furthermore, according to the situated agent action selection free-flow tree in Fig. 1, the two implemented behaviours can be hierarchically subdivided into smaller behaviours. For example, split GetPacket could be divided to LookForPacket, GoToPacket, PickUpPacket with corresponding behaviour changes.

2) Implementation: Implementation of the code follows directly the design as described above. However, in the final implementation an optimisation has been incorporated to

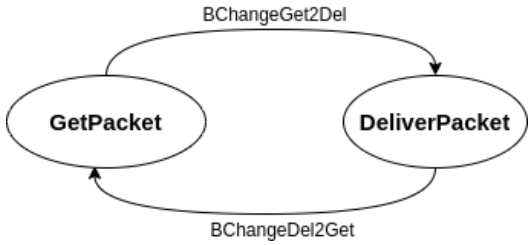

Fig. 2. Agent behaviour change classes.

improve the efficiency and performance of the agent packet collection and delivery. Instead of wandering to boxes and drop off points randomly, agents directly move to collect closest packets and drop them off at the closest drop off points with the same color. The agents always select the packets that are closest to them, provided that they are within their perception field. If the agents cannot find any packets within their perception field they perform random movements until they find one. The same approach is used for destinations, although, when a destination is found, it is stored in the agent's memory in order to avoid redundant searches. One potential future improvement would be to replace this random exploration with a targeted one that takes into account which parts of the environment the agents have already visited. This idea could be implemented quite using flags [2] placed suitably to prevent agents to re-explore already cleared areas.

To establish the best direction to walk in, agents calculate the distance from their current position to their target locations, such as packets or drop-off points. Two distance metrics were utilized namely, Manhattan and Euclidean distances. From the distance measurement agents take the best step, closest to the target destination, if possible. To prevent agents getting stuck when facing obstacles in the path to their desired locations, possible moves are ranked according to the shortest distance to the target location. Should the shortest move be blocked, the agent will take the second best step towards it's goal. If the second best step is infeasible, it will take the third, etc. The results of this optimisation can be seen in the following section IV-A3.

3) Test results: Figure 3 shows the improvement in the speed it takes the agents to collect 60 packets of different colours in the PW environment. The optimised walking behaviours based on Manhattan and Euclidean distance measures are able to collect all 60 packets in 668, and 333 cycles, respectively. The Euclidean distance measure provides more than 41x improvement in speed when compared to the random walk. Moreover, since Euclidean distance measures the shortest path agents exploit diagonal movements, improving efficiency over the Manhattan distance measure. The total energy consumption to collect the 60 packets before and after behaviour optimisation is 983440 and 18065 units, respectively, a 54x decrease in total energy consumption. This improvement is illustrated in figure 3 in which the number of delivered packets are plotted against the Packet-World cycles for a typical run in each of the three walking behaviours.

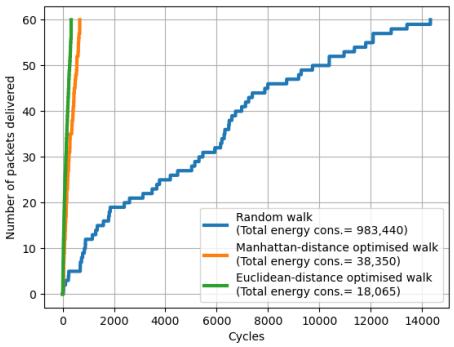

Fig. 3. Performance increase of optimised agent behaviour.

B. Coordination of agents to manage energy

Agent coordination to tackle energy management requires agents to be able to communicate either implicitly or explicitly, as well as agents to be aware of the location of charging stations in their vicinity. For agents to locate charging stations, gradient fields were implemented to guide them to the stations. The gradient value of each tile in the Grid-World was calculated using the Manhattan distance from the agent to the closest charging station, taking into consideration any obstacles like walls.

A messaging system allowing direct communication between agents was set up to manage priority in charging, which aimed to avoid agents running out of energy while waiting for another agent to finish charging. Locating and navigating to packages and goals was done in the same fashion as in section IV-A with some minor improvements detailed bellow.

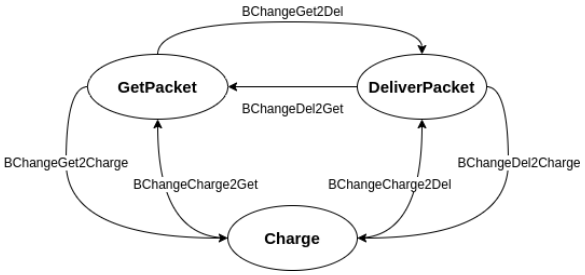

Fig. 4. Agent behaviour change classes with the added Charge behaviour.

1) Design: The design decisions made were based on the most intuitive ways to tackle each problem. As proposed by Weyns et Al. [2], a system of gradients was used to lead agents to the battery charging points. Once agents were below a certain battery threshold they would switch to charging mode and use this gradient field to navigate towards the closest charging station. The threshold used was dynamic depending on the distance of the agent from charging points as well as on whether or not the agent was carrying a packet. This allowed agents to be aware of whether they have enough charge remaining to complete their current task first, or if they needed to abort it in order to go recharge straight away. Navigation was based on the Euclidean distance measure as it provided a 2x decrease in energy consumption when compared to the Manhattan distance measure as highlighted in figure 3.

Finally direct messaging was used when an agent was in urgent need of recharging. If an agents battery level was below a critical threshold they would send out a request to take priority in charging. In case another agent was using the charger, they would check whether their own battery level is above a certain yield-threshold. If both of these criteria were fulfilled the agent currently using the charging station would vacate it and give priority to the agent with the critically low battery level, while remaining in standby mode until the charging station was freed again.

2) Implementation: To implement the behaviours described above a new behaviour was added named Charge as presented in figure 4. This behaviour was activated when agents needed to move towards the closest charging point to recharge their batteries. In addition, four behaviour change classes were added. These classes allowed the agents to transition between getting packages, delivering packages and charging. The activation of these behaviour changes occurs based on three conditions as presented in Table II.

Priority of charging in cases where an agent needed to use the charging port urgently was implemented using agent messaging, as detailed in section IV-A1.

The dynamic threshold used by agents for switching from their current task, say delivering a packet, to moving to recharge meant that agents can more accurately judge how much power is required to reach a charging station and avoid running out of power whilst getting there. Multiple threshold checks are done at each timestep by each agent to avoid running out of battery while remaining efficient. For example, agents attempting to deliver a package compute (1) how much energy is required to reach the charging station with and without carrying their current package (based on the agents gradient field value) and (2) how much energy is required to deliver the package (based on euclidean distance and energy consumption per step) and then return to the closest charging station without a package in hand. Thereby agents can decide on the best approach to take, which is often to deliver the package and then recharge without a package in hand. Agents were implemented to carry packages to charging stations if possible to avoid creating obstacles and impassable walls. However, when agents estimated that they would not reach a charge with a margin of battery remaining they would drop their packages and go to charge empty handed as this requires less energy to move.

To take into account the existence of gradient fields the walking behaviour of agents were extended. As in section IV-A2, agents always take the best step towards their goals given obstacles that may be in their way. However, for agents following gradient fields non-optimal movements were allowed when faced with obstacles allowing agents to avoid getting stuck in gradient field local minimums.

3) Test results: To assess the performance of our agents, a batch run of 100 runs was made on the energy-1 and

energy-2 environments. Figures 5 and 6 respectively show histograms of the total consumed energy by the agents at the end of each run. As can be seen the consumed energy is bounded within certain limits and does not show a big variation over the runs. This is especially true for the energy-1 environment where the energy consumption remained constant on the vast majority of runs. The agents are quite stable due to the removal of random elements in their behaviour, they always make informed choices and thus their behaviour does not alter significantly between runs. The average energy consumed for energy1 was 9529.35 units and the total cycles taken to complete the tasks were 228.21 over all 100 runs. For the energy 2 environment those values were of 54868.3 units and 886.82 cycles respectively. Future work should include a behaviour allowing agents to drop packages in clusters around other packages if they have critically low battery instead of in random locations. This would decrease the variance in energy consumption between runs, as in some cases agents can create walls of packages on the way to charging stations.

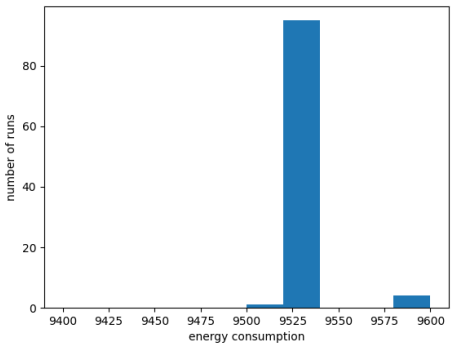

Fig. 5. Energy consumption on energy1 over 100 runs.

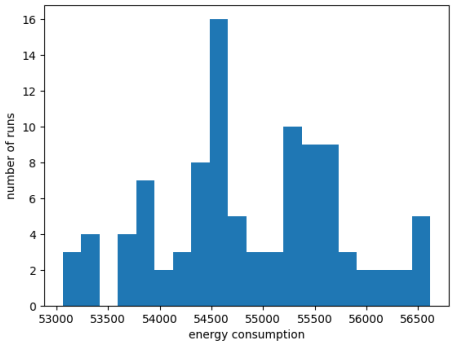

Fig. 6. Energy consumption on energy2 over 100 runs.

C. Collaboration of agent with each other

In the agent collaboration task three agents of three distinct colours are present, each able to pick up packages corresponding to their colour. Walls in the environment limit agent movements and packets are placed such that agents require other agents to help them in clearing the paths to their respective package drop off points. In this situation cooperation is required between agents to achieve the global goal of delivering all packets in the environment. The environment upon initialisation is illustrated in figure 9(a) below.

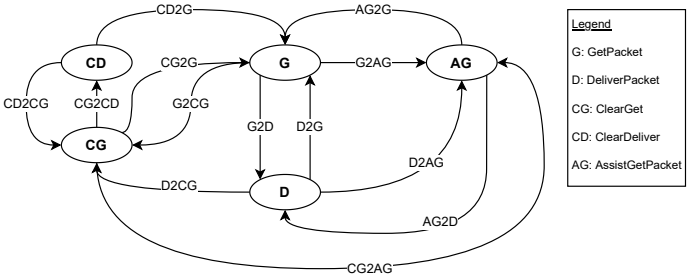

Fig. 7. Agent behaviours and behaviour changes for task III, enabling agents to clear their paths and assist other agents with theirs.

1) Design: Upon tackling the problem of agent coordination and task delegation, two basic approaches were considered. A field-based approach and a protocol-based approach.

Field based approaches rely on implicit communication through the adaption of environment. Environments can be adapted by agents by populating the grid-world with gradients, placing flags, or releasing pheromones as described by Weyns et al. [2]. These changes made to the environment are perceived by agents. From these environmental signals agents can then decide how to act. For example, when navigating down narrow corridors an agent might know to avoid paths with other agents pheromones as two agents going down the same corridor might lead them to getting stuck. Alternatively, agents may send out repulsive gradients signalling to other agents that they are in the vicinity as described by Weyns et al. [3].

Protocol-based approaches rely on explicit means of communication such as direct messaging. When an agent requires the help of another agent the agent sends a message with a request. The other agent can then either accept or decline the request based on their priorities.

Both approaches have advantages and disadvantages, however it was deemed more pragmatic for agents to directly communicate with each other instead of indirect communication by leaving markers on the environment. The large perception fields present within this task meant that agents did not struggle to find one another when in need of help. This greatly simplified the approach and made for more efficient, direct task solving.

The basic principles of this design approach is as follows. Initially agents wander around looking for drop-off locations and packets corresponding to their colour. When a goal or packet is found in their perception field, a breadth first search is used to find the shortest path to the target. If on this path to their drop-off point is blocked by a packet which the agent is unable to pick up, the agent would find an appropriate agent for help. When the correct agent is found (corresponding to the

colour of the packet blocking their path) the agent then directly messages the agent for help. The message sent requesting assistance contains the location of the packet to be moved. The process of searching for optimal paths to drop-off points and messaging agents for assistance is repeated until all the packages are delivered in the packet world.

2) Implementation: One of the vital components of the implemented system is Breadth First Search (BFS) inside an agents perceptive field, used for the establishment of the optimal path towards a package drop-off point and for retrieval of packets. Such a search scheme ensures that agents do not get trapped behind obstacles when using standard distance measures when walking. On Figure 8 you can see each step of the search depicted as different color. The optimal path is towards the agents goal is given by the green arrows. BFS was chosen over other search algorithm for its relative simplicity and optimality, as it always returns the shortest path to a desired position in the presence of obstacles or non-walkable cells such as walls and environment boundaries such that there are no collisions. The BFS is computed for each agent at each timestep of the simulation, which is computationally tractable since agent perceptive fields are not too large. If the perception fields of the agents were too large, the more complicated A* search could be implemented as it is both faster and has less memory requirements.

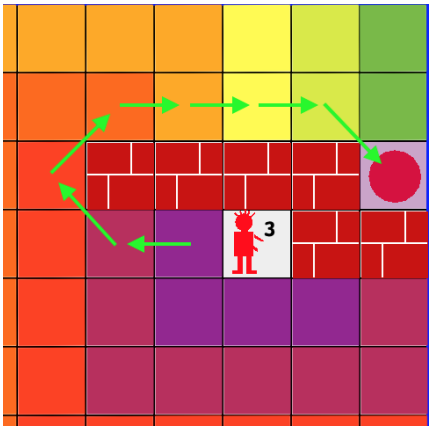

Fig. 8. Breadth First Search scheme.

Figure 7 shows all the behaviours of agents along with their corresponding behaviour change classes that are implemented. The conditions for each behaviour change is presented in Table III. The abbreviation of each behaviour is listed below along with a description of its functionality:

- ***G*** - GetPacket: Agents wander around the environment until they find and pick up a packet corresponding to their colour. The agents always choose the packets that are closest to them.
- ***D*** - DeliverPacket: Agents perform BFS to find the optimal path towards their destination and attempt to navigate towards it to drop off the packet they are holding.
- ***CG*** - ClearGet: When a package is blocking the agent they either get the packet to clear the path themselves if possible or if another colour packet blocks the path, the agent finds another agent of the corresponding colour through random wandering and messages them for help along with the location of the packet to be moved. For the messaging to occur, the two agents must be withing each other's perceptive fields.
- ***AG*** - AssistGet: Is the action taken by agents when they have agreed to assist another agent to clear their path. The request comes in the form of a message containing the location and color of the blocking packet and the ID of the agent requested to handle it so that an agent can recognize whether a request is intended for them or not. Then the agent goes to pick up that packet and deliver it to its destination. If the agent was already holding a packet, then they drop it out of they way before going to provide assistance.
- ***CD*** - ClearDeliver: An agent transitions to this behaviour when it has picked up a packet that was blocking the path to their destination. In this case the agent either delivers that packet if the destination is cleared or drops it out of the way and then attempts again to clear the path on their own or by asking for assistance.

3) Test results: From this behaviour change diagram and implementation agents are then able to solve the problem of task delegation in this complex environment, and effectively coordinate their actions to deliver all the packets. Figure 9 shows four stages during run-time with (a) corresponding to environment initialisation and (d) the termination of the program. In (b) one can see that the red and blue agents have successfully cleared the green agents path to it's goal. In (c) agents have worked together in clearing all the paths to all the agents' respective drop-off locations.

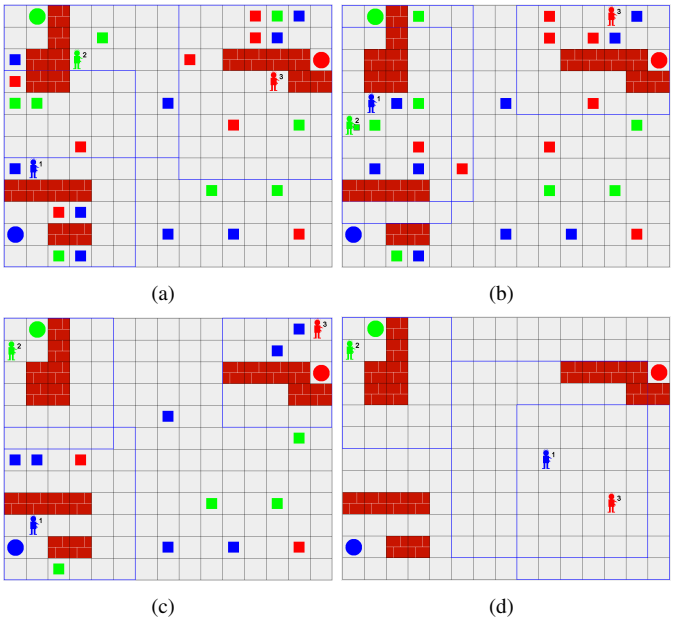

Fig. 9. Task delegation execution in packet world

Despite the implementation of the BFS, and protocol-based

task delegation, the global solution is still not always reliable. Figure 10 illustrates this over 50 runs on PW. Although most runs finished in under 1500 cycles, a considerable amount of runs took far longer, with some requiring even more than 5000 cycles. This inefficiency is mainly due to random agent wandering when searching for an agent to help and in agents waiting for their paths to be cleared by others. This inefficient behaviour could be improved by memorising the last seen locations of agents and various environmental markers. Information like flag locations would mean that agents could easily return to "memorised" locations and avoid wandering behaviour, allowing agents to indirectly communicate despite not seeing one another. Additionally, future work could also include the prioritisation of tasks among agents which would improve task efficiency.

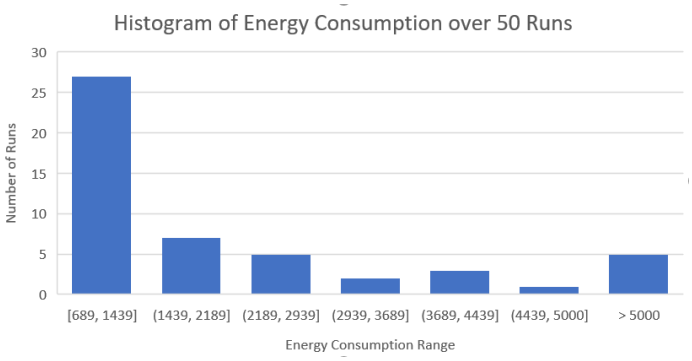

Fig. 10. Cycles taken histogram in batch run of collaboration task.

V. Related Work

The field of Multi-Agent Systems (MAS) has received a significant amount of attention from the scientific community over the course of the last decades. Numerous classification approaches and design patterns were presented by a multitude of researchers. So, to gain valuable insights about the field, we have studied several significant works.

"An introduction to multiagent systems" by Wooldridge [4] provides us with an important notion on the place of MAS in the scientific world. This is done through presentation of the history of trends in computational science, followed by an examination of various scenarios, purposes, and objections to MAS. This paper does a good job of explaining the relevance of MAS solutions such as the ones seen in this paper in the future of logistics and robotics.

Rocha, Boavida-Portugal, and Gomes [1] give a detailed introduction into MAS, by presenting a general overview, classification criteria, and key characteristics of the field.

A work done by Juziuk, Weyns, and Holvoet [5] on the literature review of MAS design patterns provided us with important ideas on how to approach the development process of the project. In their work, they stress the importance of a systematic approach to the design and present a detailed overview of existing design patterns in the field of MAS. The goal of the paper is to increase the accessibility of knowledge for practitioners in the field about design patterns. Following a systemic approach such as the one recommended allowed us to reuse large parts of our code on all three tasks solved. This also allows for the agent be further developed with more behaviours being added to the current patterns.

The main focus of Rao and Georgeff's [6] work was to explore a particular type of agents, Belief-Desire-Intention (BDI) agents. In their paper the authors laid a foundation for BDI agents from different perspectives and considered various implementations of such agents. Overall, the study presents a new paradigm for the development of MAS.

Weyns et Al. [7] provide a good interpretation of the environment as middleware in MAS. The middleware interpretation of the environment allows it to easily incorporate elements such as gradient fields and messaging which facilitate easy agent communication/coordination even between different types of agents.

VI. Future Work

Although all three presented problems were solved successfully, there is still much room for improvement to optimise these solutions in terms of turns it takes to complete the tasks and the resulting energy consumption. Furthermore, those two latter approaches failed to perform on the optional more challenging environments. In hindsight, integrating Breadth-First- Search-based navigation on the energy management task would have easily allowed the agents to navigate around walls in the energy-3 environment. The maze environment, on the other hand, presents an even bigger challenge and we hypothesized that it could have been solved using a stigmergic approach [2], such as flags or pheromones.

Approaches that include inter-agent information sharing about the explored world with the use of either direct messaging or flag placing for example could prove useful. Also, a predefined pattern through which to explore the environment in search of packets which would replace the random step exploration behaviour could greatly improve the overall performance of the solution. Some forms of indirect means of communication such as pheromones, flags or gradient placement could also be used to improve agent coordination when it came to the tasks requiring explicit cooperation.

The logical next step in this work would be to unify all the different behaviours presented in this paper in order to create a MAS able to tackle an environment that involves all the challenges faced in the three different parts combined. Such an agent would be one step closer to a solution which could eventually be implemented in real-life logistics situations.

VII. Discussion and Conclusion

In this paper we described three important topics or problems in MAS, namely the *autonomous behaviour of single agents*, the *coordination of agents to manage energy* and *collaboration between agents*. Within each of these areas designs are proposed and solutions implemented successfully solving the problems. From these respective solutions various key observations have been made.

TABLE I
BEHAVIOUR CHANGE CONDITIONS TRUTH TABLE FOR TASK I.

Condition	BChangeGet2Del	BChangeDel2Get
hasPacket	True	False

TABLE II
BEHAVIOUR CHANGE CONDITIONS TRUTH TABLE FOR TASK II.

Condition	BChangeCharge2Del	BChangeCharge2Get	BChangeDel2Charge	BChangeDel2Get	BChangeGet2Charge	BChangeGet2Del
hasPacket	True	False	True-	False	False	True
needsCharging	False	True	True	False	True	False

TABLE III
BEHAVIOUR CHANGE CONDITIONS TRUTH TABLE FOR TASK III.

Condition	AG2D	AG2G	CD2CG	CD2G	CG2AG	CG2CD	CG2G	D2AG	D2CG	D2G	G2AG	G2CG	G2D
hasPacket	True	False	-	False	True	True	True	-	-	False	-	-	True
pathBlocked	-	-	True	False	False	-	False	-	True	False	-	True	False
receivedRequest	False	False	False	False	False	False	False	True	True	False	True	False	False

When dealing with the autonomous behaviour of single agents, transitioning from a randomized grid traversal to a more sophisticated and targeted agent behaviour leads to a substantial improvement of the performance of the system. Total energy consumption and duration for completing the delivery of all packets is minimized significantly. However, this improvement is far from optimal. Employing selfish competitive agents results in the generation of conflicts and deadlocks between agent goals, among other problems. Two or more agents might target the same packet, but only one will pick it up, while the other will have gone towards that packet for nothing, resulting in a considerable waste of time and energy.

The problem of energy management while delivering packets requires several considerations such as battery gradient fields and communication. Gradient fields are an adequate way of guiding agents to charging points, although they provide no information about whether this charging port is currently being used or not. A basic system of messaging allows agents to never run out of battery by informing each other in case one urgently requires to charge so that the other agents may yield that agent priority. Messaging strategies such as these, the use of gradient fields, and optimised movement can help ensure that agents do not run out of power during simulations. Estimations of the energy required to complete tasks improve agent efficiency and help avoid the problem of agents running out of power on the way to charging stations.

While messaging can help manage battery demands and let critically low agents have preference to chargers, it by itself does not mean explicit agent coordination. True coordination and cooperation between agents is essential for completing tasks that cannot be completed by single agents. Agent coordination and task delegation highlight importance of integrating structured communication between the agents, enabling them to minimize their energy consumption and cooperate to solve complex tasks that cannot be solved individually. Indirect messaging between agents, through the placement of markers on the environment, can further improve cooperation and overall MAS efficiency as agents can then respond to agent requests despite not being in range to directly communicate.

REFERENCES

[1] Jorge Rocha, Inês Boavida-Portugal, and Eduardo Gomes. Introductory chapter: Multi-agent systems. In *Multi-Agent Systems*. IntechOpen, 2017.

[2] Danny Weyns, Alexander Helleboogh, and Tom Holvoet. The packet-world: a test bed for investigating situated multi-agent systems. In *Software Agent-Based Applications, Platforms and Development Kits:*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 383–408. 2005.

[3] Danny Weyns. Architecture-based design of multi-agent systems. 2010.

[4] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[5] Joanna Juziuk, Danny Weyns, and Tom Holvoet. Design patterns for multi-agent systems: A systematic literature review. In *Agent-Oriented Software Engineering*, pages 79–99. Springer, 2014.

[6] Anand S Rao, Michael P Georgeff, et al. Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[7] Tom Holvoet Danny Weyns, Alexander Helleboogh and Michael Schumacher. *The agent environment in multi-agent systems: A middleware perspective*. 2009.