Building collaboration in multi-agent systems using reinforcement learning

Mehmet Emin Aydin and Ryan Fellows

Department of Computer Science and Creative Technologies,
University of the West of England,
Frenchay Campus, Bristol, UK
{mehmet.aydin, ryan.fellows}@uwe.ac.uk

Abstract. This paper presents a proof-of concept study for demonstrating the viability of building collaboration among multiple agents through standard Q learning algorithm embedded in particle swarm optimisation. Collaboration is formulated to be achieved among the agents via competition, where the agents are expected to balance their action in such a way that none of them drifts away of the team and none intervene any fellow neighbours territory, either. Particles are devised with Q learning for self training to learn how to act as members of a swarm and how to produce collaborative/collective behaviours. The produced experimental results are supportive to the proposed idea suggesting that a substantive collaboration can be build via proposed learning algorithm.

Keywords: Agent collaboration · reinforcement learning · multi-agent systems · Q learning · disaster management

1 Introduction

Cutting-edge technologies facilitate the daily-life of individuals and societies with more opportunities to overcome challenging issues continuously introducing new smart gadgets day-in day-out. These astonishing technologies introduce changes with use of smart sensors in most of the time, which places a crucial role in our daily life as they are literally everywhere any more. Internet of Things (IoT) is one of key technologies to organise smart sensors in order to facilitate living environments with more and more services such as Smart homes and cities, highly-efficient engineering products, crews/swarms of robots etc. An other hot-topic attracts much attention nowadays is the swarms of unmanned aerial vehicles (UAVs). A particular example can be the teamed up UAVs to collect information from disaster areas for predicting/discovering the level of damage and human suffering and subsequently identifying humanitarian aid and rescue activities. This is due to the fact that information collection plays a very crucial role in disaster management, where the decisions are required to be done timely and based on correct and up-to-date information. Swarms of UAVs can be devised for this purposes, which are expected to remain inter-connected all the time to deliver the duties collaboratively [4]. Swarms of UAVs can also be

typical implementations area of IoT, where smart sensors and tiny devices, which are drones (UAVs) in this case, require efficient and robust settings and configuration. However, it is not easy to design and run an efficiently exploring swarm due to various practical issues such as energy limitations. This paper introduces a proof-of-concept learning algorithm to train multiple individual agents/devices for smart behaviors and collaboration.

Multi-agent systems (MAS) is an up-to-date artificial intelligence paradigm, which attracts much attention for modeling intelligent solutions following divide-and-conquer principle. It imposes dividing the complete functionality into modules and tasks, so that each task can be delivered by a proactive and smart entity, so-called agent, which are expected to act autonomously and accumulate experience to exploit ahead in fulfilling duties more efficiently. It is also expected that the agents would collaborate for a collective intelligence to deliver the entire functionality. This paradigm has proved success so many times in a wider problem solving horizon [5, 7, 20, 25]. It proves that developing IoT models and UAV implementations using MAS paradigm will produce a substantial benefit and efficiency. However, building a collaboration among multiple agents remains challenging since MAS studies have not reached to a sufficient level of maturity due to the difficulties constitute from the nature of the problem.

This paper introduces an approach for implementing a learning algorithm to build collaboration among multiple agents in order to create collective behaviour. It is a reinforcement learning algorithm, which best fits real-time learning cases within dynamically changing environments. The individual agents are expected to learn from past experiences for which how to stay interconnected and remain as a crew to collectively fulfill the duties without wasting resources. The latter purpose enforces the individual agents to compete in achieving higher rewards through out of the entire process, which makes the study further important since collaboration has to be achieved while competing. Previously, a competition-based collective learning algorithm has been attempted with learning classifier systems for modelling social behaviours [18]. Although there are many other studies conducted for collective learning of multi-agents with Q learning [15, 29], the proposed algorithm implements a competition-based collective learning approach extending Q learning with the notion of individuals and their positions in particle swarm optimisation (PSO) algorithm, which ends up as Q learning embedded in PSO.

The rest of the paper consists of the following structure; the background and literature review is presented in Section 2, the proposed reinforcement learning algorithm is introduced in Section 3, the implementation of the algorithm for scanning fields is elaborated in Section 4, experimental results and discussions are detailed in Section 5 and finally conclusions in Section 6.

2 Background

Swarm intelligence is a subfield of artificial intelligence (AI) in which the intelligence in behaviours emerges as the outcome of collective effort by self-organised simple entities, e.g. agents, organisms, individuals. Simple organisms that live

in colonies; such as ants, bees, bird flocks etc. have attracted the attention of researchers with their collective intelligence and emergent behaviours manifested in their activities. A population of such simple entities helps create emergent and intelligent behaviours through interaction among one another without using any set of instruction(s) to fulfill, and subsequently construct a swarm intelligence system [22].

Swarm intelligence approaches reveal the collective behaviour of social insects in performing specific duties, where modelling the behaviour of those social insects facilitates that such models can be used as the basis of varieties of artificial entities and emerging collective intelligence. This leads to paving problem solving for real world problems by exploiting the discovered problem solving capabilities of social insects in this way. The main aim is to understand and model the behaviours of simple individuals and the their local interactions with neighbouring peer individuals and the entire their environments, so as to attain more complex behaviours that might be useful for solving larger and more complicated problems, e.g. optimisation problems [12], [32].

Reinforcement learning (RL) is a machine learning approach in which unsupervised learning rules work alongside with a reinforcement mechanism to reward an agent based on its action selection activity to respond the stimulus from its own environment. It can be also called as semi-supervised learning since it receives a reinforcement point, either immediate or delayed, fed back from the environment. Let Λ be an agent works in environment E, which stimulates Λ with its state $s \in S$, where S is the finite set of states of E. The agent Λ will evaluate this perceived state and make a decision to select an action $a \in A$, where A is the finite set of actions that an agent can take. Meanwhile, the reinforcement mechanism, may also be called as reward function, assesses the action, a, taken by Λ in response to state s and produces reward r to feed back to Λ. Here, the ultimate aim of the agent Λ is to maximize its accumulated reward by the end of the learning period/process, with $\{\max \mathbf{R} = \sum_{i=1}^{\infty} r_i\}$, where ∞ is practically replaced with a finite number such as I to be the total number of learning iterations. Although an agent is theoretically expected to function forever, it usually works for a predefined time period as a matter of practicality. There are various reinforcement learning methods developed with various properties. Among these, Q Learning [37], [34], TD Learning [8],[33], learning classifier systems [9], [10] etc are well know reinforcement learning approaches.

Multi agent systems (MAS) are well-known and relatively mature distributed collective intelligence approaches with which a set of proactive agents act individually for solving the problems in collaboration [2]. The motivation is to team up intelligent autonomous entities for solving the problems in harmony and composing a certain level of coordination to help individual autonomous agents act pro-actively and efficiently to contribute and collaborate in problem solving process demonstrating individual intelligence capacity [5]. It is useful to note that the main properties of MAS (i.e. autonomy, responsiveness, redundancy, and distributed approach) facilitate success in MAS applications, which result in a good record in implementations within many research fields including pro-

duction planning, scheduling and control [28], engineering design, and process planning [2].

Researchers are conscious on that solving complex and large problems with distributed approaches remains as a challenging issue due to the fact that there is not a productive method to commonly use for organising distributed intelligence (agents in this case) for a high efficiency [24, 29, 35]. In fact, the performances of multi-agent systems including metaheuristic teams significantly depends on the quality of collaboration [7]. Swarm intelligence-based agent collaboration is suggested in [5], while the persistence of this challenging issue is reflected in a number of recent studies including [16] and [13], where [16] introduces auction-based consensus among the agents while [13] studies theoretical bases of agent collaboration through mathematical foundations.

3 Collaborating Agents with Q Learning

Managing an efficient collaboration among concurrently functioning multiple agents within a dynamic environment is a challenging problem as described above. The agents are expected to solve a problem in collaboration while acting individually. Particle swarm optimisation algorithm is known to be an effective, population-based, nature-inspired algorithm for continuous problem domains. It can be a good framework for learning multi-agents in 2D spaces and is expected to result better once embedded with an online learning algorithm such as Q learning.

3.1 Particle swarm optimisation (PSO)

The PSO approach has been invented inspiring of social behaviours of bird flocks to solve the optimization problems in which each single solution, called a particle, joins the other individuals to make up a swarm (population) for exploring within the search space. Each particle is set to move towards the optimum (ultimate food) with certain velocity updated each time and is evaluated with a fitness value calculated through a cost function. The entire swarm of the particles conduct search across the problem space following a leading particle, which found to be nearest to the optimum. A PSO algorithm makes a start with initialising the population of solutions (the swarm) and keeps updating it iteration-by-iteration, where a typical PSO algorithm uses particles (solutions) built based on, mainly, two key vectors; position vector, $\mathbf{x_i(t)} = \{x_{i,1}(t), ..., x_{i,n}(t)\}$, and velocity vector $\mathbf{v_i(t)} = \{v_{i,1}(t), ..., v_{i,n}(t)\}$, where $x_{i,k}(t)$, is the position value of the i^{th} particle with respect to the k^{th} dimension at iteration t, and $v_{i,k}(t)$ is the velocity value of the i^{th} particle with respect to the k^{th} dimension at iteration t. The initial values for each particular solution, $\mathbf{x_i(0)}$ and $\mathbf{v_i(0)}$, are randomly generated within a range of lower and upper pre-determined boundaries. Once a solution is obtained, the quality of that solution is measured with a cost function of f_i, defined as $f_i : \mathbf{x_i(t)} \longrightarrow \Re$.

For each particle in the swarm, a personal best, $\mathbf{y_i(t)} = \{y_{i,1}(t), ..., y_{i,n}(t)\}$, is defined, where $y_{i,k}(t)$ denotes the position of the i^{th} personal best with respect to the k^{th} dimension at iteration t. The personal bests are equal to the corresponding initial position vector at the beginning. Then, in every generation, they

are updated based on the solution quality. Regarding the objective function, f_i, the fitness values for the personal best of the i^{th} particle, $\mathbf{y_i(t)}$, is denoted by $f_i^y(t)$ and updated whenever $f_i^y(t+1) \prec f_i^y(t)$, where t stands for iteration and $\prec$ corresponds to the logical operator, which becomes $<$ or $>$ for minimization or maximization problems respectively.

In addition, a global best, which is the best particle within the whole swarm is defined and selected among the personal bests, $\mathbf{y(t)}$, and denoted with $\mathbf{g(t)} = \{g_1(t), ..., g_n(t)\}$. The fitness of the global best, $f_g(t)$, can be obtained using $f_g(t) = \mathbf{opt}_{i \in N}\{f_i^y(t)\}$, where **opt** becomes **min** or **max** depending on the type of optimization. Afterwards, the velocity of each particle is updated based on its personal best, $\mathbf{y_i(t)}$ and the global best, $\mathbf{g(t)}$ using the following updating rule:

$$\mathbf{v_i}(t+1) = \delta w_t(c_1 r_1(\mathbf{y_i}(t) - \mathbf{x_i}(t)) + c_2 r_2(\mathbf{g}(t) - \mathbf{x_i}(t))) \quad (1)$$

where w is a parameter known as the inertia weight, and used to regulate the impact of the previous values of the velocities upon the current ones, where it is updated by β, decrement factor, through $w_{t+1} = w_t \times \beta$, δ is known as constriction factor that is used to keep the effects of the randomized weight within the certain range. In addition, r_1 and r_2 are both random numbers generated within the range of [0,1] and c_1 and c_2 are the learning factors for regulating social and cognitive interactions. The next step is to update the positions with $\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + \mathbf{v_i}(t+1)$ for continues problem domains and with $\mathbf{x_i}(t+1) = \frac{1}{e^{\mathbf{v_i}(t+1)}}$ for discrete domains as suggested by Kennedy and Eberhart [14]. Once the position values of all particles are updated, the fitness value of each solution is calculated as the last instruction of an iteration. The algorithm proceeds to a new iteration if the predetermined stopping criterion is not satisfied. For further information, [21] and [32] can be seen.

3.2 Q learning

Q learning is a reinforcement learning algorithm developed based on temporal-differences handled with asynchronous dynamic programming. It provides rewards for agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build map of the respective domain [36]. The main idea behind Q learning is to use a single data structure called the utility function ($Q(x, a)$). That is the utility of performing action a in state x [37]. Throughout the whole learning process, this algorithm updates the value of $Q(x, a)$ using x, a, r, y tuples per step, where r represents the reinforcement signal (payoff) of the environment and y represents the new state which is obtained as the consequence of executing action a in state x. Both x and y are elements of the set of states (S) and a is an element of the set of actions (A). $Q(x, a) \in \mathbf{Q}$ is defined as $\mathbf{Q} : S \times A \longrightarrow \Re$ and determined via $Q(x, a) = E(r + \gamma e(y)|x, a)$, where γ is a discounted constant value within the interval of [0,1] as described according to the domain and $e(y)$ is the expected value of y defined to be $e(y) = \max\{Q(y, a)\} \ \ \forall a \in A$.

The learning procedure first initialises the Q values to 0 for each action. It then repeats the following procedure. The action with the maximum Q value is

selected and activated. Corresponding Q value of that action is then updated using $Q^{t+1}(x,a) = Q^t(x,a)+\beta(r+\gamma e(y)-Q^t(x,a))$, where $Q^t(x,a)$ and $Q^{t+1}(x,a)$ are the old and the new Q values of action a in state x, respectively. β is the learning coefficient changing in [0,1] interval. This iterative process ends when an acceptable level of learning is achieved or a stopping criterion is satisfied. For more information see Sutton and Barto [31].

3.3 Swarms of Learning Agents

PSO is one of very well know swarm intelligence algorithms used to develop collective behaviours and intelligence inspiring of bird flocks. Although it has a good record of success, furnishing each individual with learning capability remains an important aspect to be developed further for an improved collective intelligence. There are few studies investigating the hybridisation of reinforcement learning algorithms, especially with Q Learning algorithm for particular applications [11], [23], [27]. Likewise, Q Learning algorithm has been implemented by various studies to develop coordination of multi agent systems [19]. However, PSO has not been considered as the framework of learning agents furnished with Q Learning, where each particle within the swarm would be able to learn how to collaborate with other peer particles.

For the purpose of training the particles so that each is to behave in harmony within its neighbourhood, we propose use of Q Learning algorithm embedded in PSO in a way that the position vectors, $\mathbf{x_i}$, are updated through a well-designed implementation of Q learning to adaptively control the behaviour of the individuals towards collective behaviours, where all individual members of the swarm collectively and intelligently contribute. Hence, we revised PSO, first, with ignoring the use of velocity vector, $\mathbf{v_i}$, so as to save time and energy relaying on the fact that the position vector, $\mathbf{x_i}$, inherently contains $\mathbf{v_i}$, and does not necessitate its use as also discussed by [30], [6]. Secondly, the update rule of the position vectors, $\mathbf{x_i}$ is revised as follows:

$$\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + f(Q, x_i, a) \tag{2}$$

where $\hat{\mathbf{x}}_\mathbf{i}$ is a particular position vector obtained from $f(Q,x_i,a)$ defined as $\{\hat{x}_i | \max[Q(x_i,a)] \quad \text{for} \quad \forall a \in A\}$ in which action a is taken since it has the highest utility value, Q, returned. The main aim of each particle is to learn from the experiences gained through received rewards produced by reinforcement mechanism, where it credits rightly taken actions and punishes the wrongly taken ones. This learning property helps incrementally develop collaborative behaviours for each particle.

As clearly indicated before, reinforcement mechanism plays the crucial role in furnishing particles with learning capabilities. It remains as an independent monitoring mechanism to assess the actions taken by the particles and supply them with reinforcing payoff grades. It is usually implemented in a Reward Function, which is defined as $\mathbf{R} : S \times A \longrightarrow \mathbb{R}$. The reward function is implemented to consider the situation with a particular state, x, applied with action a, whether it is or not the correct action taken. A reward, r, is produced as the assessment

level for the situation. Thus, an efficient reward function is required to develop suitable to the problem domain.

4 Scanning disastrous area with swarm of learning agents

This problem case is adopted to illustrate the implementation of collective intelligence achieved using the multi agent learning algorithm proposed in this study, which is built up through embedding Q learning within particle swarm optimisation algorithm. Suppose that such an area subjected to some disasters is required to be scanned for information collection purposes. A flock of artificial birds (swarm of UAVs); each is identified as a particle and furnished with a list of actions to take while moving around the area in collaboration with other peer particles. Each particle is enabled to learn via the Q learning implemented for this purpose and being trained how to remain connected with the rest of the swarm.

Since the swarm intelligence framework preferred in this study is PSO, each individual to form up the swarm will be identified as a particle as is in particle swarm optimisation. Let M be the size of the swarm, where M particles are created to form up the swarm; each has a 2-dimensional position vector, $\mathbf{x_i} = \{x_{1,i}, x_{2,i} |\ i = 1, ..., M\}$, because the defined area is 2-dimensional and each particle will simply move forward and/or backward, vertically and/or horizontally. For simplification purposes, each particle is allowed to move with selecting one of predefined actions, where each action is defined as a step in which the particle can chose the size of the step only. Using the same notation as Q learning, the size of the set of actions is A, which includes forward and backward move with short, middle and long size steps. Hence, a particle can move forward and backward with selecting one of these six actions. Let $\mathbf{\Delta} = \{\delta_j | j = 1, ..., A\}$ be the set of steps including both forward and backward ones, which a particle is able to take as part of the action it wants to do. Once an action is decided and taken, the position of the particle will change as much as $f(Q, x_i, a) = \pi\delta_j$, where π is a probability calculated based on position and possible move of neighbouring particles. This function can substitute updating rule in PSO to calculate the new position of the particle under consideration. Here, the neighbourhood is considered as the other peer particles that has connectivity with the one under consideration, which is determined based on the distance in between. Let $N_i \in M$ be the set of neighbouring peer particles (agents) of ith particle, which is defined as $N_i = \{\mathbf{x_k} |\ \epsilon > d(\mathbf{x_i}, \mathbf{x_k})\}\ \forall k \in M$, where $d(\mathbf{x_i}, \mathbf{x_k})$ is calculated as a Euclidean distance and ϵ is the maximum distance, (the threshold), between two peer particles setup to remain connected. Once a particle moved as a result of the action taken, the reinforcement mechanism, the reward function in another name, assesses the decision made for this action considering the previous state of the particle before transition and the resulted position of neighbouring peer particles.

The reward is mainly calculated based on the total distance from the particle to its neighbouring particles, $\sum_{k=1}^{N_i} d(\mathbf{x_i}, \mathbf{x_k})$. If there is no neighbouring particle determined, which means the particle has lost connection, then it will

be punished with -100 negative reward. If there is still connection but is less then $N_i\epsilon$, then the negative reward will be as much as the difference between $N_i\epsilon$ and the total distance of $\sum_{k=1}^{N_i} d(\mathbf{x_i}, \mathbf{x_k})$. If the total distance from its position to all other neighbouring particles equals to $N_i\epsilon$, then that deserves the whole (highest) reward, which is 100.

5 Experimental Results

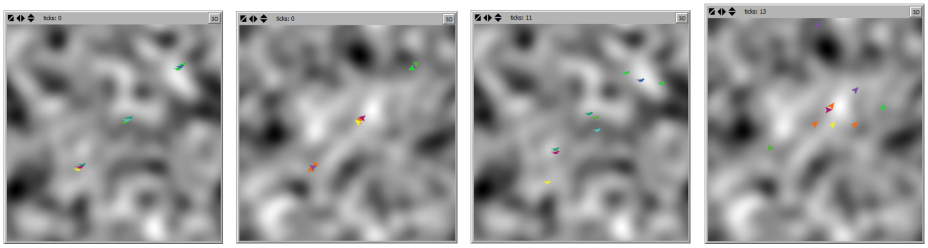
This section presents experimental results to demonstrate a proof-of-concept Q learning algorithm works to help particles (agents) self-train towards building a collaboration and behave as a swarm member. The aim is also to revise and analyse how the whole study turned out, judging whether the final implementation adhered to the expectations pre-set up. The algorithm has been implemented for a number of swarm sizes using an agent-based simulation tool called NetLogo [38].

5.1 Approximation and Evaluation

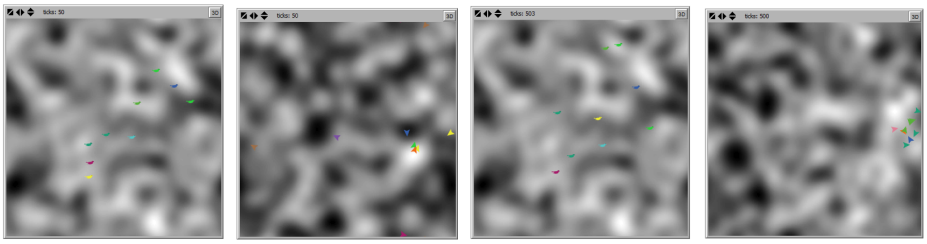
Throughout this study, Q learning is embedded in PSO as explained above to show that both algorithm work hand-by-hand to achieve a swarm of learning agents which collaborate for collective behavior/intelligence. Rather than testing the algorithm with speed, it is subject to in depth observation as to whether the particles are behaving correctly, which itself has intricacies that require close inspection.

Each particle was made to essentially be "reactive" to the environment similar to a real world environment. So if one particle moved, the others which are also moving simultaneously would need to take their fellow particles movement as well as their own into consideration and react accordingly so that they are always within proximity of their neighbours. This proximity prevents a particle from invading its neighbours space whilst also not allowing it to drift too far out of the pre-set radius, if it does either of these it will get punished whereas if it stays the "perfect" distance away, it will get the maximum reward. Corresponding details are elaborated in our technical report [3]. In order to go for comparative analysis, two swarms are created for which one was working with embedded Q learning (will be presented with the acronym of M-QL here-forth) and the other was run with a standard PSO to demonstrate the collective behaviour.

The experimentation is organised to start with the initial swarms as seen in Fig. 1a and Fig. 1b then the swarms are incremented through iterations as presented in later figures of Fig. 1, where the behaviors of both swarms, M-QL swarm and PSO swarm, after 10, 50 and 500 iterations are presented, respectively. As can be observed from Fig. 1, the particles of the swarm, learning with M-QL, can demonstrate connectivity among themselves via having a connecting distance from one another while the swarm running PSO approximates to a particular value, where all particles nearly come to overlapping positions. In fact, the behaviours of the particles in Fig. 1c, 1e, 1g clearly indicates that the individual particles keeping distance neither much falling apart nor remaining



(a) Initial positions for M-QL (b) Initial positions for PSO (c) 10 iterations with M-QL (d) 10 iterations with PSO



(e) 50 iterations with M-QL (f) 50 iterations with PSO (g) 500 iterations with m-QL (h) 500 iterations with PSO

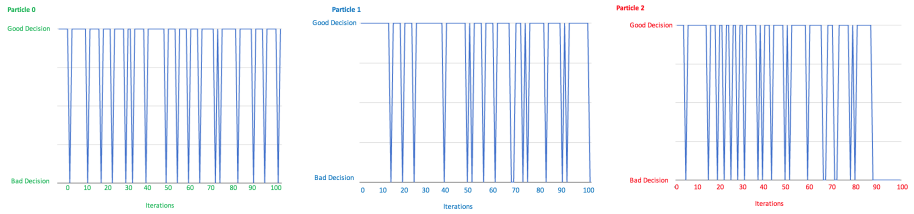
Fig. 1: A set of comparative results to demonstrate the behaviours of the learning algorithm versus PSO

too close to one another, while the number of iterations increases the distances become more fitting as Fig. 1c shows some particles are still too close to each other, but, Fig. 1g indicates a better positioning. On the other hand, Fig. 1d, 1f, 1h demonstrate how particles approximate to a targeted value without considering any having any distance among one another. More iterations help individual particles getting closer and taking overlapping positions more and more.

The results confirm that particles are at least capable of learning both in the individual sense and the subsequent group sense. Although this is a fundamentally basic example of learning, it acts as a basis which can be built on in various ways. From running various parameter configurations in earlier experimentations, it was observed that the particles choose the correct action to take in relation to the proximity as this showed they had learned which action would benefit them the most, which also showed the components of state and action were working correctly.

5.2 Individuals' learning behaviour

The performance of learning particles were also observed throughout of this research. For observing individual learning performance, three particles are taken under observation over 100 iterations. Due to the limitations of NetLogo, each simulation in this regard is physically observed from start to end, for each iteration, particles are individually judged whether each has made a good decision



(a) Particle 0 (b) Particle 1 (c) Particle 2

Fig. 2: Learning behaviours of the three particles.

or a bad decision, good decision means taking the correct action and receiving positive reward while bad decision indicates taking wrong actions and receiving negative rewards (penalty). Here, a good decision will be dictated by a particle moving in such a way it does not get too close to a fellow particle and does not drift outside of the radius either.

The results of these graphs show that for the most part, the correct decision is usually made, which shows the proposed algorithm does work. An issue occurs when two particles move at the same time because they simply cannot predict what their fellow particles will do which causes proximity problems. This could simply be rectified by extending the study to deal with simultaneous action and synchronisation. As can be seen in the graph for particle 2 (Fig. 2c), it started to make bad decisions for around 10 iterations which also continued after. This was because it drifted out of the radius of its topology and has not have a stronger penalty, which also needs to be tackled in the future. A parametric study should also be conducted for further benefits and limitations of this approach in the future.

6 Conclusions

This paper presents a proof-of concept study for demonstrating the viability of building collaboration among multiple agents through standard Q learning algorithm embedded in particle swarm optimisation. A number of particles furnished with Q learning has been subjected to self training to act as members of a swarm and produce collaborative/collective behaviours. Following introducing the algorithmic foundation and structures, an experimental study is conducted to demonstrate that the formulated algorithm produces results supporting the aimed behaviours of the algorithm. The results are produced with very simplistic assumptions, where further enhancements require further extensive theoretical and experimental studies.

References

1. N. Alechina, and B. Logan: Computationally grounded account of belief and awareness for AI agents, In Proc. of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010), Lyon, France, August 30 - Sept. 2, 2010, CEUR-WS 627.
2. M.B. Ayhan, M.E. Aydin, and E. Oztemel: A multi-agent based approach for change management in manufacturing enterprises. Journal of Intelligent Manufacturing, 26 (5), 2015, pp. 975-988.

3. M.E. Aydin, and R. Fellows: A reinforcement learning algorithm for building collaboration in multi-agent systems. arXiv preprint arXiv:1711.10574, 2017.
4. M.E. Aydin, N. Bessis, E. Asimakopoulou, F. Xhafa, and J. Wu: Scanning Environments with Swarms of Learning Birds: A Computational Intelligence Approach for Managing Disasters. In IEEE International Conference on Advanced Information Networking and Applications (AINA), 2011, pp. 332-339.
5. M. E. Aydin: Coordinating metaheuristic agents with swarm intelligence, Journal of Intelligent Manufacturing, 23(4), 2012, pp. 991-999.
6. M. E. Aydin, R. Kwan, C. Leung, and J. Zhang: Multiuser scheduling in HSDPA with particle swarm optimization, Lecture Notes In Computer Science, 5484, 2009, pp. 71-80.
7. M. Aydin: Metaheuristic agent teams for job shop scheduling problems. In Holonic and Multi-Agent Systems for Manufacturing, LNCS 4659, 2007, pp. 185-194.
8. J. Bradtke, and A. G. Barto: Linear least-squares algorithms for temporal difference learning, Machine Learning, 22(1-3), 1996, pp. 33-57.
9. L. Bull: Two Simple Learning Classier Systems, in Foundations of Learning Classier Systems, Studies in Fuzziness and Soft Computing, 183,Springer, 2005, pp. 63-90.
10. L. Bull and T. Kovacs: Foundations of Learning Classier Systems, Studies in Fuzziness and Soft Computing, 183, Springer, 2005.
11. C. Claus and C. Boutilier: The dynamics of reinforcement learning in cooperative multiagent systems, In Proc. of National Conf. on Artificial Intelligence (AAAI-98), 1998, pp. 746-752.
12. A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian: Ant system for job-shop scheduling. Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL), 34(1) 1994, pp. 39-53.
13. X. Dong: Consensus Control of Swarm Systems, In Formation and Containment Control for High-order Linear Swarm Systems, 2016, pp. 33-51. Springer.
14. R. Eberhart and J. Kennedy: A new optimizer using particle swarm theory, in Proc. of the 6th Int. Symposium on Micro-Machine and Human Science, 1995, pp. 39 - 43.
15. J. Foerster, Y.M.Assael, N. de Freitas, and S. Whiteson: Learning to communicate with deep multi-agent reinforcement learning, In Advances in Neural Information Processing Systems 2016, pp. 2137-2145.
16. M. Gath: Optimizing Transport Logistics Processes with Multiagent Planning and Control. PhD Thesis, 2015, Springer, 2016.
17. M. Hammami, and K. Ghediera: COSATS, X-COSATS: Two multi-agent systems cooperating simulated annealing, tabu search and X-over operator for the K-Graph Partitioning problem, Lecture Notes in Computer Science 3684, 2005, p. 647-653.
18. L. M. Hercog:Better manufacturing process organization using multi-agent self-organization and co-evolutionary classifier systems: The multibar problem. Appl. Soft Comput. 13(3),2013, pp. 1407-1418.
19. H. Iima and Y. Kuroe: Swarm reinforcement learning algorithm based on particle swarm optimization whose personal bests have lifespans, Lecture Notes in Computer Science, 5864, 2009, pp. 169-178.
20. A. Kazemi, M. F. Zarandi, and S. M. Husseini: A multi-agent system to solve the productiondistribution planning problem for a supply chain: a genetic algorithm approach. The Int. Jour. of Advanced Manufacturing Technology, 44(1-2), 2009, pp.180-193.
21. J. Kennedy and R. C. Eberhart: A discrete binary version of the particle swarm algorithm," 1997 IEEE Int. Conf. on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Orlando, FL, 1997, pp. 4104-4108.
22. J. Kennedy, R. Eberhart, and Y. Shi.: Swarm Intelligence, Morgan Kaufmann, San Mateo, CA, USA, 2001.
23. J. R. Kok and N. Vlassis: Sparse cooperative q-learning, in Proceedings of the International Conference on Machine Learning, ACM, 2004, pp. 481-488.
24. M. Kolp, P. Giorgini, and J. Mylopoulos: Multi-agent architectures as organizational structures, Autonomous Agents and Multi-Agent Systems, 13, 2006, pp. 3-25.
25. A. Kouider, and B. Bouzouia: Multi-agent job shop scheduling system based on co-operative approach of idle time minimisation. International Journal of Production Research, 50(2), 2012, pp.409-424.
26. P. Kouvaros and A. Lomuscio: Parameterised verification for multi-agent systems, Artificial Intelligence, 234, (May 2016), pp. 152-189.
27. Y. Meng: Recent Advances in Multi-Robot Systems, I-Tech Education and Publishing, 2008, ch. Q-Learning Adjusted Bio-Inspired Multi-Robot Coordination, pp. 139-152.
28. S. Mohebbi, and R. Shafaei: E-Supply network coordination: The design of intelligent agents for buyer-supplier dynamic negotiations. Journal of Intelligent Manufacturing 23, 2012, pp.375-391.
29. L. Panait and S. Luke: Cooperative multi-agent learning: The state of the art. Autonomous agents and multi-agent systems, 11(3),2005, pp.387-434.
30. R. Poli, J. Kennedy, and T. Blackwell: Particle swarm optimization. Swarm Intelligence, 1, 2007, pp. 33-57.
31. R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, USA, 1998.

32. M. Tasgetiren, Y. Liang, M. Sevkli, and G. Gencyilmaz: Particle swarm optimization algorithm for makespan and total flow-time minimization in permutation flow-shop sequencing problem. European Journal of Operational Research, 177(3) 2007, pp. 1930-1947.
33. G. Tesauro: Practical issues in temporal difference learning, Machine Learning, 1992, 8(3-4) pp. 257-277.
34. J. N. Tsitsiklis and R. Sutton: Asynchronous stochastic approximation and Q-learning, Machine Learning, 1994, 16(3), pp. 185-202.
35. J. Vazquez-Salceda, V. Dignum, and F. Dignum: Organizing multi-agent systems, Autonomous Agents and Multi-Agent Systems, 11, 2005, pp. 307-360.
36. C. Watkins: Learning from delayed rewards, PhD thesis, Cambridge University, 1989.
37. C. Watkins and P. Dayan: Technical note: Q-learning. Machine Learning, 8, 1992, pp. 279- 292.
38. U. Wilensky and W. Rand: An introduction to agent-based modeling: Modeling natural, social and engineered complex systems with NetLogo, MIT Press, Cambridge, 2015.