Reinforcement Learning of Supply Chain Control Policy using Closed Loop Multi-Agent Simulation

Souvik Barat[1], Prashant Kumar[1], Monika Gajrani[1], Harshad Khadilkar[1 [0000−0003−3601−778X]], Hardik Meisheri[1 [0000−0002−9014−1098]], Vinita Baniwal[1 [0000−0002−7329−3540]], and Vinay Kulkarni[1 [0000−0003−1570−1339]]

[1]TCS Research
souvik.barat@tcs.com

Abstract. Reinforcement Learning (RL) has achieved a degree of success in control applications such as online gameplay and autonomous driving, but has rarely been used to manage operations of business-critical systems such as supply chains. A key aspect of using RL in the real world is to train the agent before deployment by computing the effect of its exploratory actions on the environment. While this effect is easy to compute for online gameplay (where the rules of the game are well known) and autonomous driving (where the dynamics of the vehicle are predictable), it is much more difficult for complex systems due to associated complexities, such as uncertainty, adaptability and emergent behaviour. In this paper, we describe a framework for effective integration of a reinforcement learning controller with an actor-based multi-agent simulation of the supply chain network including the warehouse, transportation system, and stores, with the objective of maximizing product availability while minimising wastage under constraints.

1 Introduction

Business-critical systems such as supply chain networks require continual evaluation and adjustment to stay competitive and economically viable in a dynamic environment. Reinforcement Learning (RL) [35, 29] is a class of machine learning algorithms that can be used for controlling such complex systems in an adaptive and flexible manner. The goal of the system controller (also called *RL agent*) is to learn to take the best possible control actions in each possible state of the system, in order to maximise long-term system objectives. A crucial aspect of RL is the computation of next state and associated rewards for the chosen action(s), in a closed loop to enable learning. In compact systems with well-understood behaviour such as software-based games or vehicle dynamics, the action-driven state transition is simple to model, at least in terms of a probabilistic description. This is not the case for complex networked systems with a large number of entities that have their own micro-behaviour, and where the individual interactions build into emergent (and sometimes unpredictable) macro-behaviour. In such scenarios, top-down modelling allows for only a coarse approximation of the next states and rewards, hampering the training process of the RL agent.

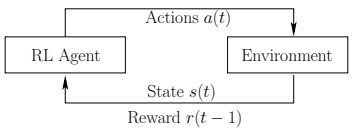


Fig. 1. Interaction of RL agent with an environment (actual system or simulation).

A more accurate representation of the next state and reward (see Fig. 1), and consequently a better estimate of the long term consequences of a series of actions for an *exceedingly complex* [33] business-critical system such as a supply should be possible using a bottom-up multi-agent simulation approach. Fundamentally, these systems are *open* as they exchange messages with their environment, and *complex* as they contain multiple non-linear feedback loops [3]. Moreover, these systems are not monolithic deterministic automatons, but are complex (scale-free) networks [5] or *systems of systems*, where the global behaviours emerge from the interactions of *autonomous*, *adaptable*, and *self-organising* sub-systems and constituent *agents* [15]. These characteristics pose obstacles to the application of alternative control approaches such as adaptive control and approximate dynamic programming. While the former requires an analytical representation of the control and adaptation laws, the latter requires at least a one-step rollout of a significant subset of actions, followed by a functional approximation of the subsequent value function.

We postulate that the use of analytical expressions for modelling (the method of choice for simpler RL applications [11, 19]), is infeasible for complex systems, and instead advocate an agent/actor based modelling abstraction [1]. The paper presents a framework that uses reinforcement learning for exploring policies and deciding control actions, and an actor-based modelling and simulation technique to perform accurate long-term rollouts of the policies, in order to optimise the operation of complex systems. The key attraction of RL is that online decision making is a one-shot forward pass through (typically) a set of neural networks, and does not require online search. We use the domain of supply chain replenishment as an illustrative example to demonstrate the proposed modelling abstraction and its impact on training RL agent prior to its deployment.

2 Problem formulation

Generic RL problem: A reinforcement learning problem is described by a Markov Decision Process (MDP) [35] represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$. Here, $\mathcal{S}$ is the set of states of the system, $\mathcal{A}$ is the set of control actions that the RL agent can choose from, $\mathcal{R}$ is the set of possible rewards, P is the (possibly stochastic) transition function from $\{\mathcal{S}, \mathcal{A}\} \rightarrow \mathcal{S}$, and γ is a discount factor for future rewards. In several cases, the agent is unable to observe the state space entirely, resulting in a partially-observable MDP or POMDP [35]. A set of observations $\mathcal{O}$ is derived from $\mathcal{S}$ to represent what the agent can sense. The goal of the RL agent is to compute a policy $\mathcal{O} \rightarrow \mathcal{A}$ that maximises the future discounted long-term reward. Clearly, an accurate representation of the transition function $P : \{\mathcal{O}, \mathcal{A}\} \rightarrow \mathcal{O}$ is a critical aspect of this effort.

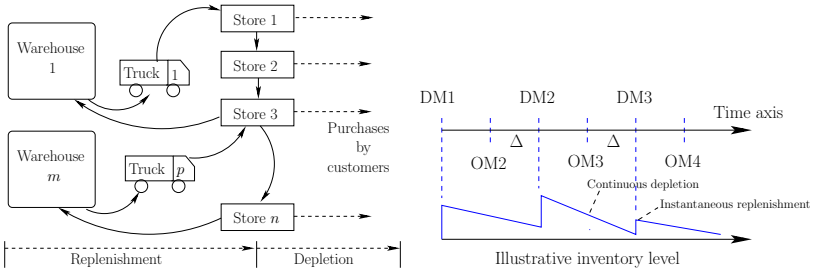


Fig. 2. [Left] Schematic of supply chain replenishment use case. [Right] Schematic of the periodic replenishment cycles. OM are ordering moments when the actions are computed, while DM are delivery moments when the inventory is actually delivered.

Specific instance: We illustrate the generic RL problem in the context of supply chain replenishment, which presents well-known difficulties for effective control [22, 30]. The scenario is that of a grocery retailer with a network of stores and warehouses served by a fleet of trucks for transporting products. The goal of replenishment is to regulate the availability of the entire product range in each store, subject to the spatio-temporal constraints imposed by (i) available stocks in the warehouses, (ii) labour capacity for picking and packaging products in the warehouses, (iii) the volume and weight carrying capacity of the trucks, (iv) the transportation times between warehouses and stores, (v) the product receiving capacity of each store, and (vi) available shelf space for each product in each store. A schematic of the product flow is shown in Fig. 2 [Left].

A temporal illustration of the replenishment process is shown in Fig. 2 [Right]. The replenishment of inventory is assumed to take place periodically (typically every 6 hours), at the time instants marked as DM (Delivery Moments). However, since it takes a non-zero amount of time to procure the new inventory within the warehouse, to transport it to the store, and to load it onto the shelves, the replenishment quantities of each product are computed at the time instants marked OM (Order Moments). There is a lead time Δ provided between each OM and the subsequent DM. The inventory of products is a monotonic non-increasing function between delivery moments, and there is a step increase at every DM when new inventory is provided to the stores.

Processes to be modelled: The warehouses stock a range of products and supply them to the stores as described in Fig. 2. This involves packing the products (using trolleys), loading packed products to the trucks/carriers and delivering them to respective stores on predefined routes. Each sub-process contains constraints such as the warehouse labour capacity, machine capacity, number of trucks, and the truck volume/weight capacities. The uncertainties that emerge due to the probabilistic behaviours of the individual elements are: unavailability and varying productivity of the labours, sub-optimal machine throughput and unavailability and unaccounted delays of the trucks. Trucks are constrained by the volume and weight capacities, often they are suited for specific types of products and each of them has probabilistic characteristics, such as: propensity for transportation delay and break down.

Let us assume that there are m warehouses, p trucks, and n stores in the system. From operational perspective, each store stocks $i = \{1, \ldots, k\}$ unique varieties of products, each with a maximum shelf capacity $c_{i,j}$ where $j \leq n$ is the index of the store. Further, let us denote by $x_{i,j}(t)$ the inventory of product i in store j at time t. The replenishment quantities (*actions*) for delivery moment d are denoted by $a_{i,j}(t_d)$, and are to be computed at time $(t_d - \Delta)$. The observation $O(t_d - \Delta)$ consists of the inventory of each product in each store at the time, the demand forecast for each product between the next two delivery moments, and meta-data such as the unit volume v_i and weight w_i, and its shelf life l_i.

Note that the *states* differ from the *observations* in this case because the actual inventory at the time of replenishment is $x_{i,j}(t_d)$, which must be estimated based on the current inventory $x_{i,j}(t_d - \Delta)$ and some forecast of the depletion in the remaining time Δ. The inventory $x_{i,j}(t)$ depletes between two delivery moments $(d-1)$ and d, and undergoes a step increase by amount $a_{i,j}(t_d)$ at time t_d. The actions are constrained by the various capacities in the system, including those within warehouses, in the transportation network, and in the store. The reward $r(t_{d-1})$ is a function of the previous actions $a_{i,j}(t_{d-1})$ and the evolution of inventory states $x_{i,j}(t)$ in $t \in [t_{d-1}, t_d)$. From a business perspective, of particular interest are: (i) the number of products that remain available throughout the time interval $[t_{d-1}, t_d)$, and (ii) the wastage of any products that remain unsold past their shelf lives. Mathematically, we define this as,

$$r(t_{d-1}) = 1 - \frac{\text{count}(x_{i,j} < \rho)}{k\,n} - \frac{\sum_{i=1}^{k}\sum_{j=1}^{n} w_{i,j}(t_{d-1})}{\sum_{i=1}^{k}\sum_{j=1}^{n} X_{i,j}}, \tag{1}$$

where $\text{count}(x_{i,j} < \rho)$ is the number of products that run out of inventory (drop below fraction ρ) at some time $t \in [t_{d-1}, t_d)$, $w_{i,j}(t_{d-1})$ is the number of units of product i in store j that had to be discarded in the time interval because they exceeded their shelf lives, and $X_{i,j}$ is the maximum shelf capacity for product i in store j. Since both negative terms in (1) fall in the range $[0, 1]$, we see that $-1 \leq r(t_{d-1}) \leq 1$. The goal of the control algorithm is to compute actions $a_{i,j}(t_{d-1})$ that maximise the discounted sum of these rewards from the present moment onwards, $\sum_{z=0}^{\infty} \gamma^z r(t_{d+z})$.

3 Related work

Control approaches: Supply chain control, including inventory management, has been a problem of interest for a long time [32]. Theoretically, such problems can be solved using mixed-integer linear programming [34, 7], but this is infeasible at real-world scales. Instead, actual implementations typically use approximate methods such as threshold policies [9]. Other traditional methods such as state feedback [6] and model predictive control [24] have similar scaling issues. Adaptive critics [31] and reinforcement learning [12, 17, 27] have also been used in literature, but primarily for single-product scenarios. However, these methods along with approximate dynamic programming (ADP) are likely to be the best suited for our problem, because they are known to work in related areas.

ADP has been used for task allocation problems in transportation networks [13, 37], but has the inherent restriction of requiring analytical descriptions of value functions and at least a one-step rollout of the policy. Imitation Learning has been used in robotics [10], but is only feasible where an expert policy is available to imitate. Reinforcement learning has been used in the computation of torque commands for robotic applications [28, 20]. In these cases as well as other complex systems [11, 19], the system models are analytically defined, thus simplifying the computation of step rewards and next state of the system. This is because RL is effective only when the (stochastic) transition functions closely approximate the real system to be controlled. In situations where the system cannot be described analytically, algebraic expressions cannot be used to compute rewards and transitions. Where RL has been used in supply chain management [12, 17, 38, 27], it tends to focus on single-product scenarios. An experimental approach can be used for training the RL agent when the system is non-physical (for example, is itself a piece of software as in the case of computer games) [26]. However, applying experimental approach on the actual system is not feasible in the case of business-critical systems. Therefore, the development of (and integration with) a high-fidelity simulation model is crucial for effective training of the RL agent and controlling complex systems.

Modelling approaches: Complex systems are typically modelled using two broad categories of approaches: *top-down* approach and *bottom-up* approach [36]. A *top-down* approach visualises a system from a higher scale and specifies aggregated macro-behaviour. This approach uses a range of models, such as *mathematical/analytical model* and *enterprise model* (EM), to represent and analyse the system as a whole. The *analytical models*, *e.g.*, Mixed Integer Linear Programming, represent a system using mathematical formulae and use rigorous mathematical and statistical problem solving techniques for system analysis. The *operational research techniques* are the specialised form of analytical models. The *enterprise models*, such as ArchiMate [16], BPMN [39] and System Dynamics [25], also serve a wide range of modelling and analysis needs by representing aggregated system behaviour. However, these approaches are found to be inadequate to represent systems (and their transition functions P) that contain multiple adaptable, self organising and uncertain entities (such as warehouses, trucks, products and store), individualistic behaviour (such as product expiry) and exhibit emergent behaviours (such as availability, unavailability and damages of products that are significantly influenced by several uncertain spatio-temporal aspects: transportation delay, inappropriate packaging with certain class of products, availability of other similar products, *etc.*).

The *bottom-up* approaches, such as *actor model of computation* [1] and multi-agent systems [23], capture the micro-behaviours of a system using a collection of interacting *actors* [1, 14] or *agents* [23] (henceforth *actor*) and help to observe emergent macro-behaviour at a higher level of abstraction. The agent and actor based technologies, such as Erlang [4] and Akka [2], realise system as set of *autonomous*, *self-contained*, and *reactive actors*.

4 Solution considerations and proposed approach

The proposed framework contains two control loops as shown in Fig. 3: (i) a model centric loop for training of RL agent and evaluation of new policies prior to their implementation in a real system, and (ii) real time control loop that include real system. We adopt simulation as an aid to compute micro-behaviours and observe emerging macro behaviours, system *state* and *rewards*; and use RL agent (or controller) as a computing machinery for deriving suitable *actions*.

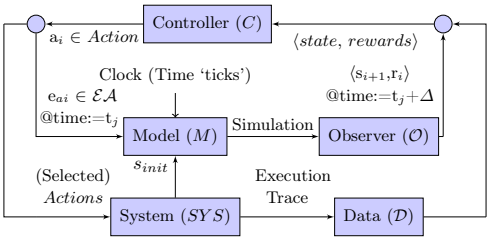


Fig. 3. Proposed approach.

A meta-model to represent systems using an extended form of *actor* is shown in Fig. 4. Here, a system is a set of `Actors`, whose characteristics can be described using a set of variables or `Properties`. Each `Actor` has its own `State` and `Behaviour`. They interact with other `Actors` by consuming and producing `Events`, where an incoming (*i.e.* consumed) `Event` may trigger a `Behaviour` unit that can change the state of an `Actor`, send `Events` and create new `Actors`.

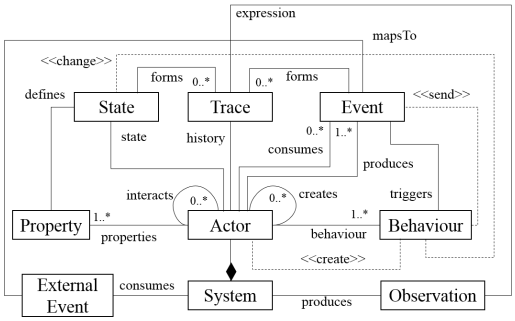


Fig. 4. Meta model to represent complex system using agents termed as 'actor'.

We extend this canonical form of an `Actor` with a notion of `Trace` (a sequence of information about `State` and `Events`) and an explicit construct to describe uncertainty in behavioural specification (as shown in Fig. 5). Formally, an actor (ACT) is a five-tuple: $\langle \mathcal{S}, \mathcal{EC}, \mathcal{EP}, Tr, \mathcal{B} \rangle$, where

- $\mathcal{S}$ A set of labels that represent `States` of an `Actor`.
- $\mathcal{EC}$ A finite set of `Events` that an `Actor` can consume.
- $\mathcal{EP}$ A finite set of `Events` that an actor can produce. Here, $\mathcal{EC}$ and $\mathcal{EP}$ are not disjoint set (`Events` $\mathcal{EC} \cup \mathcal{EP}$ are consumed within an actor).
- Tr A finite sequence of a tuple, where each tuple captures consumed `Event`, corresponding `State` and produced `Event`, *i.e.*,
  $s_0 \xrightarrow{ec0} \langle s_1, E_{ps1} \rangle \xrightarrow{ec1} \langle s_2, E_{ps2} \rangle \ldots \xrightarrow{ec(k-1)} \langle s_k, E_{pk} \rangle$, where
  $\{e_{c1}, \ldots, e_{c(k-1)}\} \in \mathcal{EC}$ and $E_{ps1}, \ldots, E_{pk} \subset \mathcal{EP}$
- $\mathcal{B}$ A set of behavioural units. We consider that every behavioural unit $B \in \mathcal{B}$ is a set of programs that contain a sequence of stochastic statements. An abstract syntax to specify these programs is presented in Fig. 5.

```
stmt ::= become(state_new)        State change of an actor
  | send(event_i, ACT_k)          Send event to an actor
  | create ACT(state_init)        Create new actor
  | e_1:stmt_1+...+e_n:stmt_n     Guarded statements
  | probably(e_prop):stmt         Probabilistic statement
```

Fig. 5. Abstract Syntax of Behavioural Statements.

In conformance with the meta model presented in Fig. 4, the system can be formally described as a quadruple $M = \langle \mathcal{ACT}, \mathcal{EA}, CLK, \mathcal{O} \rangle$, where $\mathcal{ACT}$ is a finite but dynamic set of actors; $\mathcal{EA}$ is a fixed and finite set of external `Events`, which are triggered from external sources; CLK is a clock that triggers virtual time `Events` or simulation 'ticks' (as external events) to progress simulation; and $\mathcal{O}$ is a set of observations. An observation is a tuple $\langle AS, Fact \rangle$, where AS is a set of actor states and $Fact$ are temporal expressions on actor traces (*e.g.*, occurrence of events over time). Two critical components of the control setup are described below.

1. Computation of $\mathcal{O} \rightarrow \mathcal{A}$: The observations $\mathcal{O}$ consist of the inventories at time $(t - \Delta)$, the order forecast (expected depletion) $f_{i,j}$ in the next time period, and product meta-data v_i, w_i, and l_i. There are five input variables per product, leading to a total input size of $5kn$. The output size is kn, and each output can take any value between 0 and $X_{i,j}$. The number of variables in computing such a mapping directly using RL is infeasibly large. Therefore, we compute this mapping iteratively, one product at a time. We use a form of RL known as Advantage Actor[1] Critic (A2C) [21] to compute the actions. The Actor is a neural network with 5 inputs and 11 outputs, representing discretised and normalised actions between 0 and 1. The Critic is a neural network with the same 5 inputs but a single output representing the expected *value* of the current observations. The basic reward function given in (1) was augmented for

Algorithm 1 Compute observations using Actor Simulation

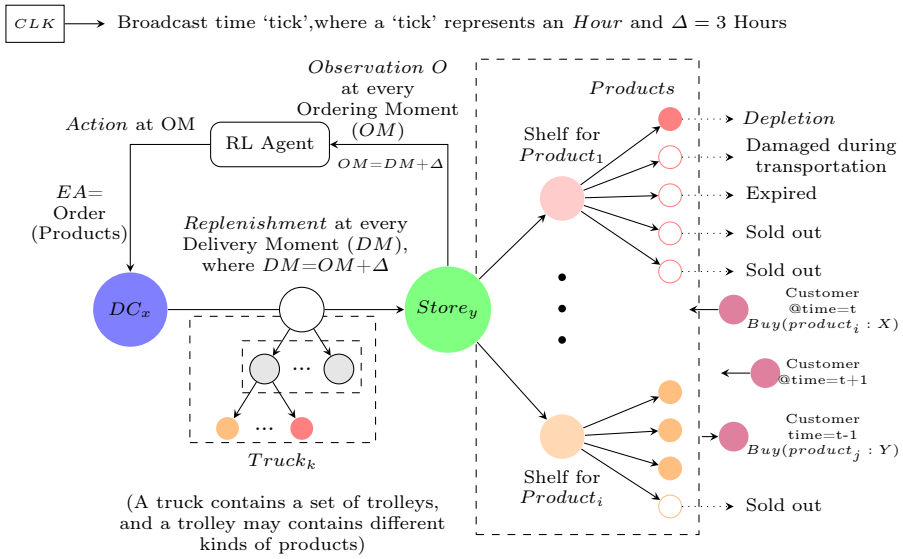
procedure SIMULATE(M_{init}, D, Δ)
  Duration: D, observation interval: Δ, $interval := 0$, $time := 0$, state := $Active$
  $\forall\ ACT_i \in \mathcal{ACT}$: `create` $ACT_i(s_0)$ ▷ Initiate simulation by instantiating actors
  while (time != D) **do** ▷ Simulate for D time duration
    `receive`($event_{ext}$) ▷ M receives an external event
    if ($event_{ext}$ is 'tick') **then** ▷ If event is a time event
      $time := time + 1$
      if (state := $Active$) **then**
        $\forall\ a_i \in \mathcal{ACT}$: `send`($event_{ext}$,$a_i$) ▷ Broadcast time event
      if ($interval = \Delta$) **then**
        $interval := 0$
        $O :=$ `observe`(M) ▷ Compute O from all S and Tr of $\mathcal{ACT}$
        `notify`(O, C) ▷ Notify $\langle state_i, reward_i \rangle$ to C
        `state` := $Pause$ ▷ Pause time event for RL agent
      else
        $interval := interval + 1$
    if ($event_{ext} \in \mathcal{EA}$) **then** ▷ If event is a RL Agent action
      `state` := $Active$ ▷ Restart time event for next observation
      for $a_i \in \mathcal{ACT}$ (of M) **do**
        if ($\exists\ e_x$ such that $e_x \in \mathcal{EC}$ of ACT_i **And**
          map(e_{ext},$event_x$) $\in \mathcal{MAP}$) **then**
          `send`(e_x,a_i) ▷ Send event to relevant actors

[1] Not to be confused with 'actor' in the simulation context

the purposes of training the agent, by a penalty on any actions that could not be feasibly implemented in the system because of capacity exceedance. This allows the RL agent to relate the effect of individual decisions on the net system reward.
2. Computing $\mathcal{A} \rightarrow \mathcal{O}$: The updates to $O \in \mathcal{O}$, the actor states, and events, are computed through simulation. As shown in Algorithm 1, all actors of an initial model (M_{init}) are instantiated to a random state or a system state to initialise a simulation. Actors execute their behaviours in their respective threads, interact with each other through actor events, change their states (possibly several times to respond to external and internal events) and create new actors. The external events that include time 'tick' and events corresponding to the RL actions are propagated to all relevant actors and allowed them to react for time duration Δ before the emergent states and traces are observed. The observed and computed O is then notified to the controller for the *next* RL action.

5 Illustration, validation and discussion

We use an actor language [8] that supports proposed actor abstraction to model a supply chain network as shown in Fig. 6. The simulation dashboard depicting the emerging reward related factors in a hypothetical retail chain is shown in Figure 7. We use a data set spanning one year derived from a public source [18]. A total of 220 products were chosen from the data set, and their meta-data (not available originally) were defined manually. A single store and a single truck are used in the results presented here (see Sec 6 for extensions). Forecasts were computed using a uniform 10-step trailing average for each product. The store



Legends: *Circle*: *Actor*, *Arrow*: Event interaction, and *Dotted Box*: containment
Observation O at OM_t:$\langle$ *Product Inventories* at OM_t, *Reward* from $OM_{(t-1)}$ to $OM_t\rangle$
Reward: Function of products unavailability, products expired and number of shelves looked empty

Fig. 6. An implementation of supply chain replenishment case study described in Fig. 2

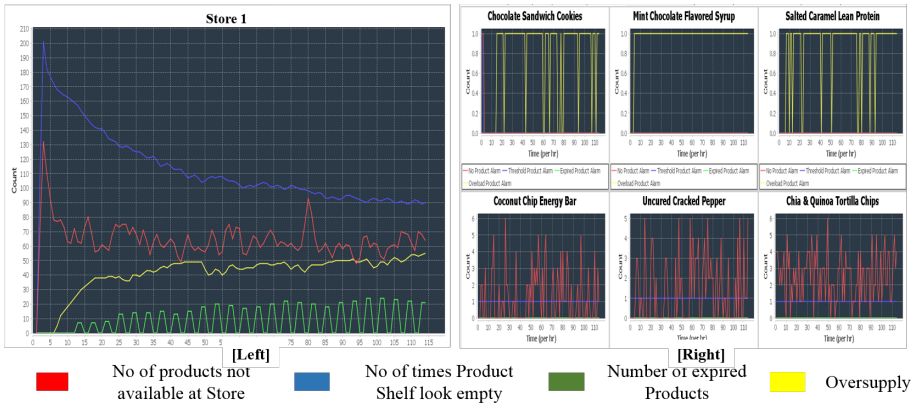


Fig. 7. [Left] Instance trace of product unavailability, emptiness of shelves, product wastage and product over-supply for a shop. [Right] Traces of individual products.

capacity, truck volume and weight capacity, and labour counts were computed based on the order volumes seen in the data.

The time between successive delivery moments set to 6 hours (leading to 4 DM per day). The lead time Δ was 3 hours. The truck volume constraint was set lower than the average order numbers in order to stress the system. The initial normalised inventory level for each product is set to 0.5, and the level below which penalty is imposed is set to $\rho = 0.25$. Of the order data set, the first 225 days (900 DM) were used for training, while the remaining 124 days (496 delivery moments) were retained for testing. Fig. 8 [Left] shows the training over 15 episodes, each spanning the 900 DM in the training data set. The average reward over all 220 products is seen to increase as training proceeds. The reward is compared with an industry-standard replenishment heuristic adapted from prior literature [9]. We see that the reward at the end of training exceeds the heuristic performance, and this advantage is retained on the test data set as well (plotted using separate markers at the ends of the curves). Also shown in Figure 8 [Left] is the 'exploration rate' of RL, which is the probability with which the RL algorithms takes randomised actions.

The performance advantage is due to the nature of Fig. 8 [Right], which plots the inventory levels of products on the test data set (496 delivery moments). Both algorithms begin with an initial (normalised) inventory level of 0.5 for all products. However, RL is able to maintain a higher average inventory level than the heuristic. The characteristics of the Critic and Actor networks of the RL agents are illustrated in Fig. 9. The value attached by the Critic network is shown in Figure 9 [Left], as a function of the inventory level (after averaging over all other feature values). The peak value is near the penalty threshold $\rho = 0.25$. The value drops quickly below this level. There is also a decrease in the estimated value at very high inventory levels, due to the higher rates of product wastage. Fig. 9 [Right] shows the replenishment quantity requested by the Actor network, again as a function of the inventory level of the product after averaging over all other features. We note that the requested replenishment rises as inventory levels drop from 0.5 towards the penalty threshold of 0.25.

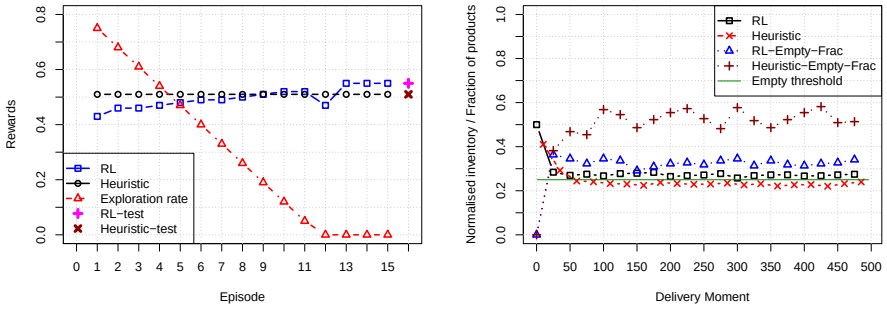


Fig. 8. [Left] Evolution of rewards during training, in comparison with the heuristic algorithm. [Right] Trend of average inventory levels across all 220 products over time.

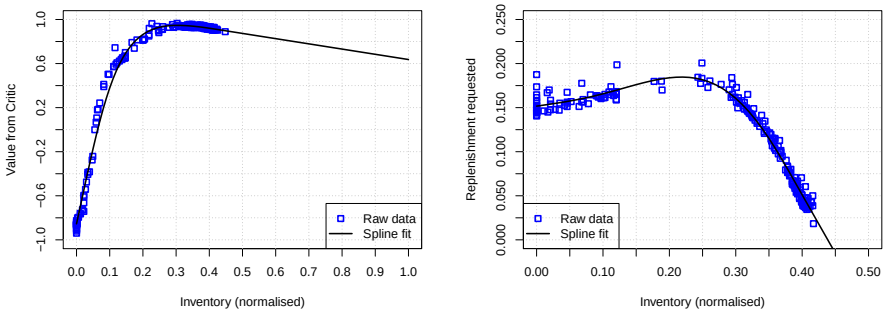


Fig. 9. [Left] Estimate of state value as a function of inventory. [Right] Replenishment requested by the RL agent as a function of current inventory level.

6 Conclusion

We described the use of a realistic closed loop multi-agent simulation model for training a reinforcement learning based control policy, as opposed to the traditional use of analytical expressions for rewards. Initial tests show that training using proposed approach is both feasible (in terms of computational time and expense) and effective (in terms of discrimination power between subtle differences in system behaviour). The use of the proposed actor based simulation as an environment to understand the overall implication of multiple RL actions (produced for different parts of a network) and locally optimised solutions for subsystems in a global system context, can also be viewed as a viable option.

The ultimate goal of this work is to develop a closed-loop simulation and reinforcement learning framework, that allows us to deploy the trained agent on a real system with minimal subsequent adjustments. Towards this end, we have attempted to make the actor-based simulation as detailed as possible, while remaining within the bounds of design effort and computational feasibility. A trained version of the reinforcement learning algorithm for computing replenishment orders is expected to become operational in a grocery retail network

with approximately 10^3 stores and 10^5 products per store, in December 2019. The current setup has been tested on a single store scenario with nearly 10^4 products, which generates approximately 4×10^5 Actors in the simulation. The challenge in the coming year is to extend the capability to full system simulation while retaining computational feasibility. We believe this is feasible, based on the following considerations. First, the current implementation of the simulation and learning loop works on a single laptop. There is thus scope for increasing the computational power as necessary. Second, the decision-making portion ($\mathcal{O} \rightarrow \mathcal{A}$ map) works independently for each product, allowing us to parallelize the online workflow. Finally, it may be possible to loosely partition complex supply chain networks into sub-networks, further reducing the computational complexity.

References

1. Agha, G.A.: Actors: A model of concurrent computation in distributed systems. Tech. rep., DTIC Document (1985)
2. Allen, J.: Effective akka. O'Reilly Media, Inc. (2013)
3. Anderson, P.: Perspective: Complexity theory and organization science. Organization science **10**(3), 216–232 (1999)
4. Armstrong, J.: Erlang - a survey of the language and its industrial applications. In: Proc. INAP. vol. 96 (1996)
5. Barabási, A.L., Bonabeau, E.: Scale-free networks. Scientific american **288**(5), 60–69 (2003)
6. Bouabdallah, S., Noth, A., Siegwart, R.: Pid vs lq control techniques applied to an indoor micro quadrotor. In: Proc. of The IEEE International Conference on Intelligent Robots and Systems (IROS). pp. 2451–2456. IEEE (2004)
7. Caro, F., Gallien, J.: Inventory management of a fast-fashion retail network. Operations Research **58**(2), 257–273 (2010)
8. Clark, T., Kulkarni, V., Barat, S., Barn, B.: ESL: An actor-based platform for developing emergent behaviour organisation simulations. In: International Conference on Practical Applications of Agents and Multi-Agent Systems. pp. 311–315. Springer (2017)
9. Condea, C., Thiesse, F., Fleisch, E.: Rfid-enabled shelf replenishment with backroom monitoring in retail stores. Decision Support Systems **52**(4), 839–849 (2012)
10. Duan, Y., Andrychowicz, M., Stadie, B., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., Zaremba, W.: One-shot imitation learning. In: Proc. of Conference on Neural Information Processing Systems (NIPS). vol. 31 (2017)
11. Gabel, T., Riedmiller, M.: Distributed policy search RL for job-shop scheduling tasks. International Journal of Production Research **50**(1), 41–61 (2012)
12. Giannoccaro, I., Pontrandolfo, P.: Inventory management in supply chains: a reinforcement learning approach. International Journal of Production Economics **78**(2), 153–161 (2002)
13. Godfrey, G.A., Powell, W.B.: An ADP algorithm for dynamic fleet management, i: Single period travel times. Transportation Science **36**(1), 21–39 (2002)
14. Hewitt, C.: Actor model of computation: scalable robust information systems. arXiv preprint arXiv:1008.1459 (2010)
15. Holland, J.H.: Complex adaptive systems. Daedalus pp. 17–30 (1992)
16. Iacob, M., Jonkers, D.H., Lankhorst, M., Proper, E., Quartel, D.D.: Archimate 2.0 specification. Van Haren Publishing (2012)

17. Jiang, C., Sheng, Z.: Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. Expert Systems with Applications **36**(3), 6520–6526 (2009)
18. Kaggle: Instacart market basket analysis data. `https://www.kaggle.com/c/instacart-market-basket-analysis/data` (Retrieved August 2018)
19. Khadilkar, H.: A scalable reinforcement learning algorithm for scheduling railway lines. IEEE Transactions on ITS (2018)
20. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
21. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: Advances in neural information processing systems. pp. 1008–1014 (2000)
22. Lee, H.L., Padmanabhan, V., Whang, S.: Information distortion in a supply chain: The bullwhip effect. Management science **43**(4), 546–558 (1997)
23. Macal, C.M., North, M.J.: Tutorial on agent-based modelling and simulation. Journal of simulation **4**(3), 151–162 (2010)
24. Mayne, D.Q., Rawlings, J.B., Rao, C.V., Scokaert, P.O.: Constrained model predictive control: Stability and optimality. Automatica **36**(6), 789–814 (2000)
25. Meadows, D.H., Wright, D.: Thinking in systems. Chelsea Green Publishing (2008)
26. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
27. Mortazavi, A., Khamseh, A.A., Azimi, P.: Designing of an intelligent self-adaptive model for supply chain ordering management system. Engineering Applications of Artificial Intelligence **37**, 207–220 (2015)
28. Powell, W.B.: Ai, or and control theory: A rosetta stone for stochastic optimization. Princeton University (2012)
29. Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited, (2016)
30. Sabri, E.H., Beamon, B.M.: A multi-objective approach to simultaneous strategic and operational planning in supply chain design. Omega **28**(5), 581–598 (2000)
31. Shervais, S., Shannon, T.T., Lendaris, G.G.: Intelligent supply chain management using adaptive critic learning. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **33**(2), 235–244 (2003)
32. Silver, E.A.: Operations research in inventory management: A review and critique. Operations Research **29**(4), 628–645 (1981)
33. Simon, H.A.: The architecture of complexity. In: Facets of systems science, pp. 457–476. Springer (1991)
34. Smith, S.A., Agrawal, N.: Management of multi-item retail inventory systems with demand substitution. Operations Research **48**(1), 50–64 (2000)
35. Sutton, R., Barto, A.: Reinforcement learning. MIT Press, 2 edn. (2012)
36. Thomas, M., McGarry, F.: Top-down vs. bottom-up process improvement. IEEE Software **11**(4), 12–13 (1994)
37. Topaloglu, H., Powell, W.B.: Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. INFORMS Journal on Computing **18**(1), 31–42 (2006)
38. Valluri, A., North, M.J., Macal, C.M.: Reinforcement learning in supply chains. International journal of neural systems **19**(05), 331–344 (2009)
39. White, S.A.: BPMN modeling and reference guide (2008)