Discover Artificial Intelligence

Discover

Research

Applying reinforcement learning to plan manufacturing material handling

Swetha Govindaiah[1] · **Mikel D. Petty**[2]

Received: 10 May 2021 / Accepted: 2 August 2021
Published online: 22 September 2021

Abstract
Applying machine learning methods to improve the efficiency of complex manufacturing processes, such as material handling, can be challenging. The interconnectedness of the multiple components that make up real-world manufacturing processes and the typically very large number of variables required to specify procedures and plans within them combine to make it very difficult to map the details of such processes to a formal mathematical representation suitable for conventional optimization methods. Instead, in this work reinforcement learning was applied to produce increasingly efficient plans for material handling in representative manufacturing facilities. Doing so included defining a formal representation of a realistically complex material handling plan, specifying a set of suitable plan change operators as reinforcement learning actions, implementing a simulation-based multi-objective reward function that considers multiple components of material handling costs, and abstracting the many possible material handling plans into a state set small enough to enable reinforcement learning. Experimentation with multiple material handling plans on two separate factory layouts indicated that reinforcement learning could consistently reduce the cost of material handling. This work demonstrates the applicability of reinforcement learning with a multi-objective reward function to realistically complex material handling processes.

Keywords Machine learning · Reinforcement learning · Material handling · Planning · Multi-objective learning

1 Introduction and motivation

Current trends in advanced manufacturing, such as "make to order" fabrication and short lifespan products, require manufacturers to be flexible and responsive in order to remain competitive. In a review on manufacturing flexibility, Sethi defines flexibility as "… adaptability to a wide range of possible environments that it may encounter" [1]. Of the different types and forms of manufacturing flexibility, the focus of this work is on material handling flexibility [2, 3]. *Material handling* (MH) is the activity of transporting materials from place to place within a manufacturing facility. The cost of MH is significant and non-value-added (i.e., the end product's value does not benefit from the MH component of the manufacturing expenses), thus MH managers strive to plan and conduct MH so as to enable smooth and efficient production while minimizing MH cost. The goal of this research was to facilitate fast and effective changes to MH plans in response to external uncertainties such as change in demand [4].

✉ Mikel D. Petty, pettym@uah.edu; Swetha Govindaiah, swethagovindh@gmail.com | [1]HelpSystems, LLC, 4500 Lockhill Selma Road, San Antonio, TX 78249, USA. [2]Computer Science Department, University of Alabama in Huntsville, 301 Sparkman Drive OKT N353, Huntsville, AL 35899, USA.

Unfortunately, MH plans for realistic factories can be and often are very complicated [5]. Representing an MH plan in a manner sufficiently formal for conventional mathematical optimization requires extensive definitions, notations, and assumptions (as detailed in Sect. 3). Reducing the complexity of MH plans to the point that mathematical optimization methods could be applied to produce theoretically optimum MH plans appears to be an unsolved problem.

Therefore, in order to produce fast and effective changes to MH plans reinforcement learning (RL) was applied instead of mathematical optimization. Starting from simple initial MH plans, the RL process produced a sequence of iteratively modified plans. Those plans were tested, and their costs estimated, using an abstract discrete event simulation of a factory. The cost of the MH plans calculated by the simulation was used as the reward in the RL algorithm. After each plan was generated and tested, the RL algorithm revised it using one of a set of MH-specific plan change operators. Based on the rewards, the RL algorithm learned which of the operators to apply in different situations to improve a material handling plan over the iterations of the learning process. This process was performed for two separate factory layouts and MH scenarios.

The research questions examined in this work were:

- How can MH in a realistically complex factory environment be mapped to the structure of an RL process?
- How should MH routes be generated and how can they be improved?
- What is a suitable RL reward function when applied to MH?
- How should the multiple components of the MH costs be included in a multi-objective reward function?
- How can an MH plan be changed in response to a reward to produce a better plan?
- Can the RL algorithm learn a policy to improve the MH plan?

A commercial office furniture manufacturing and assembly plant located in Athens, Alabama USA was carefully studied and served as a representative example of manufacturing and MH for this work.

The remainder of this paper is organized as follows.[1] Section 2 presents background concepts and reviews applicable prior work on the application of RL to MH. Section 3 defines the problem and presents a formal representation of an MH plan suitable for processing by an RL algorithm. Section 4 discusses the learning process and reward calculation. Section 5 reports and analyzes the experimental results for the first of the two factory layouts, and Sect. 6 does the same for the second factory layout. Section 7 presents the conclusions of the work and Sect. 8 lists recommendations for future work.

2 Background

This section provides background information on the two main fields of this research, MH and RL, and summarizes selected relevant research literature applicable to MH planning.

2.1 Material handling

MH is the activity of transporting materials within a manufacturing facility, e.g., from a storage area to a workstation, or from a workstation to a loading dock, or between a building and a vehicle [10]. MH often plays a significant role in the smooth and timely production of finished products and is important in industrial competitiveness [11]. Effective MH not only contributes to an efficient production environment but also implicitly supports planning, forecasting, resource allocation, and inventory management. Within a single manufacturing facility, MH may use a variety of manual, semi-automated, and automated equipment that supports the manufacturing process and interfaces with supply chain operations. An MH plan details the quantities and types of material to be transported, where the materials are transported from and to, and how the materials are transported [12]. Figure 1 shows examples of two types of transport equipment used for material handling within manufacturing facilities: a tow tractor, also known as a tugger, and a forklift. (Both of these types of equipment are represented in the experimentation that is reported in Sects. 5 and 6.)

The potential for changes in the market demand a manufacturer's products motivates the need for MH flexibility. A change in demand will typically require changes to product production, and thus to MH, in the manufacturing facility.

[1] Some sections of this paper are based on [6–9], but the text in those sections has been extended and clarified. Sections 2.5 and 6 of this paper are completely new, and Sect. 6 describes results not previously reported.

Fig. 1 Examples of equipment used for material handling; a human puller, i.e., cart carrying bins of parts (left) and a forklift moving heavy materials (right). Credits: (Left) KBS Industrieelektronik GmbH, "Pick-by-Light Kommissionierfahrzeug mit neun Zielbehältern", July 1 2010, Used under Creative Commons Attribution-Share Alike 3.0 Unported license, Online at https://commons.wikimedia.org/wiki/File:Pick-by-Light_Kommissionierfahrzeug_PickTerm_Cart.jpg, Accessed July 21 2021. (Right). Bakker, J. J., "Sonneborn Refined Products BV Spaarndamseweg 466", June 12 2009, Used under Creative Commons Attribution 2.0 Generic license, Online at https://commons.wikimedia.org/wiki/File:Forklifter_at_Sonneborn.jpg, Accessed July 18 2021

Some demand changes may be seasonal, such as peaks in demand for school furniture just prior to the start of a new school year. Other factors may lead to more frequent, even weekly, changes in demand. To accommodate a change in demand a facility's MH process must be re-planned [4, 13]. Although manual processes (e.g., lean MH) and semi-automated tools (e.g., enterprise resource planning) are available for planning MH, an automated process that can quickly generate an MH plan for a changed demand in a given factory could be very useful.

2.2 Reinforcement learning

Machine learning methods may be grouped into three categories: *supervised learning*, *unsupervised learning*, and *reinforcement learning* [14, 15]. Supervised and unsupervised learning methods typically require large data sets, either to train the learning algorithm (supervised learning) or to provide sufficient information for the algorithm to learn the pattern or structure in the data (unsupervised learning). In contrast, RL does not require large data sets; instead, in RL the data is generated during the learning process in the form of rewards for actions taken, which serve as evaluative feedback to a learning agent [16, 17]. RL has increased in importance in recent years because of its applicability in a wide range of disciplines for which identifying all possible examples to train the agent is difficult; those applications include robotics, gaming, computer vision, business, control system engineering, simulation-based optimization, networks, anti-crime operations [18] and cybersecurity [19].

The central idea of RL is that the learning agent learns, over time, what action to take in any given situation by a process analogous to methodical trial and error. The learning agent tries the different actions available to it in different situations and evaluates the outcome of each action, both in terms of the action's immediate effect on the environment and its long-term contribution to the learning agent's overall goals.

In its basic form, RL can be understood in terms of five key elements: environment, agent, state, action, and reward. Figure 2 illustrates the relationship between these elements. The *environment* is anything which is outside the learning agent in the RL problem.

The *agent* is the learner, which learns to map states to actions by attempting actions and interacting with the environment. The *state* is the agent's internal representation or abstraction of a particular situation or configuration of the environment as perceived by the agent. An *action* is an operator that changes some attribute or aspect of the environment that may result in a new state. An agent in state s_i may select action a_i to move towards the goal, which takes the agent to a new state s_{i+1} and produces a reward r_{i+1}. The *reward* is an external signal (which is calculated by a reward function) from the environment in response to an action which tells the agent whether the action taken is positive or negative with respect to the agent's goal. The reward is a method of communicating the objective of the RL problem to the agent. In general, the agent seeks to select actions that maximize its long-term cumulative

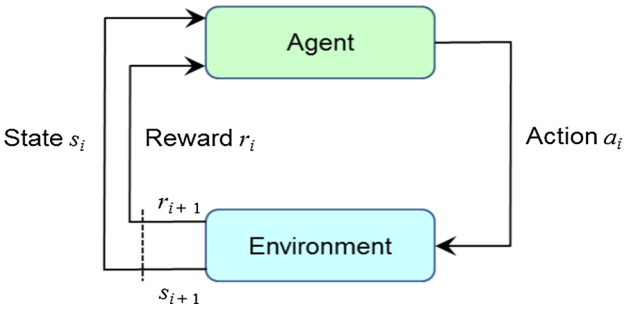

Fig. 2 General framework for reinforcement learning [16]

reward. In simple formulations of reinforcement learning, the reward value is a scalar [16]. (The notation used in this paper for representing RL elements was adapted from [16].)

Two additional components of the RL process are the policy and value functions. The *policy* is a total function π: $S \times A \rightarrow pr$, where S is a set of states, A is a set of actions, $A(s)$ is the set of actions available in state s, with $A(s) \subseteq A$, and $pr \in [1]$ is a probability. If $\pi(s, a) = pr$, then the probability of the learning agent choosing action a when in state s is pr.

A *value function* is a secondary reward function that estimates the long-term value, i.e., the cumulative future reward, when following policy π. There are two types of reward functions. The first, denoted $v_\pi(s)$, is the expected reward starting from state s and following policy π. The second, denoted $q_\pi(s, a)$, is the expected reward starting from taking action a in state s following policy π. If the reward value of taking a particular action being in a particular state stays the same over time, then the RL problem is called *stationary*; otherwise, it is called *non-stationary*.

The value of each state and state-action pair (s, a) can be determined by the Bellman equations, which are also called dynamic programming equations [20]. The Bellman equations specify how the values of the current state and state-action pair of the learning agent can be estimated using the successor state and state-action pair, respectively. The value of state s_i under policy π is denoted $v_\pi(s_i)$; function v_π is called the *state-value function for policy* π. Similarly, the value of state-action pair (s_i, a_i) under policy π is denoted $q_\pi(s_i, a_i)$; function q_π is called the *action-value function for policy* π. The current state value ($v_\pi(s_i)$) and state-action pair value ($q_\pi(s_i, a_i)$) are updated using successive state value ($v_\pi(s_{i+1})$) and state-action value ($q_\pi(s_{i+1}, a_i)$). Equations (1) and (2) are the Bellman equations for v_π the state-value function for policy π and q_π the action value function for policy π, for the simplest RL method, the temporal difference (TD (0)) method:

$$v_\pi(s_i) \;= v_\pi(s_i) \;+ \mathrm{a}[r_{i+1} + \gamma v_\pi(s_{i+1})\; v_\pi(s_i)] \tag{1}$$

$$[q_\pi(s_i, a_i) \;= q_\pi(s_i, a_i) \;+ \alpha[r_{i+1} + \gamma q_\pi(s_{i+1}, a_i)\; q_\pi(s_i, a_i)] \tag{2}$$

where s_i and s_{i+1} are the states at steps i and $i+1$ respectively, a_i is the action at step i, r_{i+1} is the reward at step $i+1$, α represents a step-size parameter such that $0 \le \alpha \le 1$ and γ represents a discount rate parameter such that $0 \le \gamma \le 1$ [16].

Tabular RL methods use a table to record the approximate values of the state (or state-action pair). Tabular methods are typically used when there is a small number of states and actions. In tabular stationary RL, the step size is the mean of the number of visits to the particular state. In continuous RL problems, as the number of iterations i tends to ∞ the sum of future rewards tends to ∞, so to keep the future rewards finite, a discount rate is used.

RL requires a balance between greedy selection of the action with the maximum reward, as far as is currently known, and non-greedy selection of actions with unknown or currently non-maximum rewards in order to learn their effectiveness. In the RL context, the former is called *exploitation* and the latter is called *exploration*. These aspects differentiate RL learning from supervised learning, in which the agent is explicitly taught.

The central idea of RL is that the agent learns to influence or gain control of an environment over which it initially has no control by improving its policy. A policy which is better than or equal to all other policies for the same problem is called an *optimal policy* [16]. Finding an optimal policy is difficult for most practical problems because of issues such as the lack of computational resources to do an exhaustive search and unknown environment dynamics. Hence, for most practical purposes an improved or approximately optimal policy is the goal.

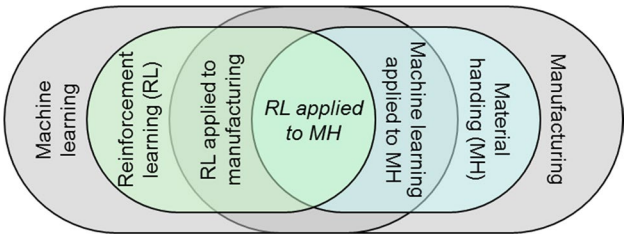

Fig. 3 Relationships between the related work topics

2.3 Applying reinforcement learning to material handling

The application of RL to manufacturing MH was both motivated and challenged by special characteristics of the MH application.

- *Motivation* The inputs to the MH process are a specific factory layout (including workstations, walkways, and storage areas within the factory known as warehouses), a collection of MH equipment and workers to operate that equipment, and a given level of customer demand for products to be manufactured. From an MH perspective, the demand is manifested as a set of rates at which the workstations consume parts of different types. A change to any of these inputs (layout, equipment, or demand) would likely require a change to the MH plan. Consequently, there is a very wide range of possible inputs, and the associated MH plans can be very specific to a given set of inputs. Also, examples of MH inputs and corresponding plans were not available in sufficient number and generality to train unsupervised learning methods.
- *Motivation* The MH plans produced during an RL learning sequence could be used later as the starting plan for a subsequent RL learning sequence executed in response to a new customer demand, potentially shortening the later learning sequence or improving the later plan.
- *Challenge* Factories are complicated. A typical manufacturing facility contains a wide range of physical elements, including workers, workstations, raw materials, pre-made parts, manufacturing equipment, transportation equipment, computers and communication networks, and many others. These physical elements support a number of tightly interconnected processes, including planning, inventory control, production, and MH. To apply RL to MH, the elements of the factory involved in the MH process must be represented in terms of the elements of an RL algorithm, i.e., environment, states, actions, and rewards. Doing so with sufficient mathematical specificity and formality was a significant challenge because of the larger number of variables involved.
- *Challenge* RL algorithms learn by selecting actions and seeing how well they work. Clearly, testing the numerous MH plans produced by the RL algorithm by actually trying each one in a real factory is completely impractical. Therefore, a simulation model of the factory and the MH process was developed and used to assess the performance of the MH plans generated by the RL algorithm and generate the reward for each. The simulation model was based on standard modeling techniques from discrete event simulation but is deterministic rather than stochastic and somewhat more abstract than a conventional discrete event simulation model.

2.4 Related work

This section briefly surveys selected examples of related work. Figure 3 illustrates the relationships between the relevant topics. Of particular interest are applications of RL to MH and applications of RL with multi-objective reward functions. Although considerable research has been performed on individual MH issues, most studies have focused on dynamic scheduling for fixed layout problems; responding to changes in demand has received little attention. This research is focused on applying RL for generating increasingly efficient MH plans for more complex scenarios with multiple assembly lines, routes, part types, and equipment types in response to temporally varying demand.

2.4.1 RL applied to manufacturing

The potential for RL to identify or generate an improved or optimal policy in real time has led to its application in many subfields of manufacturing, such as production scheduling [21] and routing [22]. The RL literature includes a number of

papers that describe the use of RL for learning control policies in various areas of manufacturing, including generating a dynamic scheduling policy for a multi-agent RL system [23]; learning an inventory replenishment policy [24]; addressing single-machine dispatching rule selection problem in production scheduling [25]; learning dispatching policies in a multi-agent job-shop scheduling problem [26]; performing maintenance task scheduling [27]; path planning for mobile robots in genetic algorithm-based task scheduling in dynamic intelligent warehouse environments [28]; studying global supply chain problems [29]; and developing routing policies in multi-agent job-shop environments [22].

Recent work applied RL to a multi-load carrier scheduling problem [5]. In that work, a single dolly train serves a one-dimensional assembly line of multiple workstations whenever a request arrives for parts from one or more workstations. The dolly train can carry multiple loads (containers). MH requests are generated using a look-ahead request generating policy, and heuristic-based dispatching algorithms and standard pickup and drop-off rules are used to generate a drop-off sequence.

The RL reward is related to both throughput of the assembly line and the MH distance. The RL state is represented using four features, and each state contains three actions. The action values are approximated using a radial basis function with Q-learning [30]. The experimental setup is relatively simple; it includes only one assembly line. The RL algorithm primarily tries to learn a multi-load carrier scheduling policy, rather than learning a policy to improve the MH plan [5].

2.4.2 Multi-objective reward functions

An optimization problem where more than one objective is optimized simultaneously is referred to as a Multi-Objective Optimization Problem (MOOP) in the literature [31]. The MH application considered in this work has four objectives: ensure that no workstations run out of parts during the given plan duration, minimize the number of pieces of equipment used in the MH plan, minimize the number of parts stored at each workstation, and minimize the total distance traveled by all of the pieces of equipment in the MH plan. These objectives potentially compete with each other, for example, reducing the number of parts stored at a workstation increases the likelihood of that workstation running out of parts.

Each of these objectives is associated with an individual reward signal; hence in the RL MH problem the learning agent receives a vector of rewards from the environment after each action. As mentioned earlier, conventional RL algorithms typically learn from a scalar reward. If the reward is not scalar, as in multi-objective problems, it must be somehow converted to a scalar so that RL can be applied. However, the most effective way in general to perform this mapping of a reward vector to a scalar reward method is an open research question in RL research.

RL problems with multiple objectives are referred to as multi-objective RL (MORL) in the literature [32]. The two main classes of MORL algorithms are single-policy and multi-policy. Single-policy algorithms map the reward vector to a scalar reward and then learn the single optimal policy, whereas multi-policy algorithms learn multiple policies (one for each of the corresponding objectives). Usually these MORL algorithms generate a set of compromised solutions called a *Pareto optimal set* [32]. Single-policy MORL algorithms include linear scalarization [33], non-linear scalarization such as Chebyshev scalarization [34], lexicographic ordering-based scalarization [35], and hypervolume indicator-based (the volume of the area between the Pareto set and the reference point) algorithm [36].

Multi-policy algorithms include the convex-hull value-iteration algorithm [37] and Pareto Q-learning [38]. The hypervolume indicator has been used as a benchmark metric to compare the single-policy and the multi-policy MORL algorithms in order to evaluate their performance and limitations in an empirical study [32]. Recent research has shown how deep RL can be applied to solve MORL problems [39, 40]. Other methods, such as the geometric approach [41], have been applied to MORL problems as well.

2.4.3 RL in plan space

A neural network trained with a temporal difference algorithm was used to learn the heuristic evaluation function over states for job-shop scheduling of NASA's space shuttle processing tasks [42, 43]. The main goal was to find the shortest schedule for many simultaneous jobs, each consisting of multiple tasks, that satisfied both temporal and resource constraints. Two operators, "REASSIGN-POOL" and "MOVE", were applied to reassign a resource if it was over-allocated and to move the task within the temporal limit to satisfy the violated resource requirement, respectively. A resource-dilation factor, which is independent of the length of the schedule, was used to measure the immediate rewards. With several modifications, the algorithm eventually converged and outperformed the previous simulated annealing-based iterative repair method [42, 43].

According to Sutton, that type of problem is an instance of learning in a "plan space", which is similar to this research [16]. The objectives in [42] were to generate a conflict free schedule by removing temporal and resource constraints and to minimize the duration of the schedule. Although the proposed method involves plan-space learning, learning stops once the two temporal and resource constraints are satisfied. To optimize the duration of the schedule, the plan space method reviews the plans and selects the one with the shortest duration. They used a neural network-based function approximator by carefully selecting the features representing the schedule as input for estimating the value of each schedule after applying each change operator. In contrast, the method used here includes all of the objectives in the learning process and all of the variables of the plan that are used as the input to the simulation which generates the states. Moreover, job-shop scheduling is a scheduling problem, as opposed to the multi-objective MH plan optimization problem in this work.

2.5 Comparison of reinforcement learning with other optimization methods

This section briefly compares and contrasts RL, the method used in this research, with several other optimization methods that could conceivably be applied to the MH problem. The other methods are well-known and excellent descriptions of them are available, thus no attempt is made to provide comprehensive tutorials here.

Evolutionary algorithms explore a space of possible solutions, or solution parameterizations, for a problem using methods inspired by biological evolution. A population of possible solutions is generated, and each is evaluated as a solution to the problem using a problem-specific fitness function. The "fittest" solutions, i.e., those with the best fitness function values, are used to create a next generation of possible solutions using evolution-inspired operations including crossover and mutation. The process iterates until some stopping condition is reached, such as insufficient generation-to-generation improvement. In genetic algorithms, which are arguably the best-known type of evolutionary algorithms, the possible solutions are typically strings of numbers representing parameters to some solution function or process.

As explained in more detail in [16], evolutionary algorithms differ from the RL method in several significant ways. RL methods "learn while interacting with the environment, which evolutionary methods do not" [16]. Evolutionary methods use a population of multiple possible solutions, that are each evaluated once for fitness, and then may be transformed into new possible solutions using general operations. RL uses a single policy, which is analogous to a single possible solution, but that policy is evaluated and changed many times during the learning process, and those changes may be made by problem-specific operations (as in the research described in this paper). Finally, RL uses value functions, which estimate the cumulative current and future value of specific states in the solution process controlled by the policy, whereas evolutionary methods "search directly in policy space" guided by fitness function evaluations of the complete solutions [16].

Hill climbing algorithms seek to optimize (either minimize or maximize) an objective function $f(\mathbf{x})$, where $\mathbf{x}$ may be a vector of input values, by heuristically searching the space of possible input values. Typically, hill climbing proceeds iteratively; in each iteration, an element of $\mathbf{x}$ is adjusted in some way and a new value of the objective function $f(\mathbf{x})$ is calculated; if the new value of $f(\mathbf{x})$ is better than the previous value, the change to $\mathbf{x}$ is retained. The iterations continue until the value of $f(\mathbf{x})$ cannot be improved.

Hill climbing has some similarities to RL; the adjustments to $\mathbf{x}$ in hill climbing can be seen as analogous to the actions in RL, and the objective function $f(\mathbf{x})$ in hill climbing can be seen as analogous to the reward function in RL. There are also differences; computing the new objective function value in hill climbing depends only on the new value of $\mathbf{x}$, whereas the value function in RL estimates not only the value of the next action but all the actions that follow it. Hill climbing is a greedy method, in that the search always moves in the direction that locally improves the objective function value, whereas the standard ε-greedy RL algorithm includes both exploitation (greedy optimization) and exploration (stochastic exploration of non-locally optimal actions). For this reason, RL is less likely than hill climbing to end at a solution that is locally, but not globally, optimum.

Simulated annealing, which is similar to hill climbing and has been described as an adaptation of the Metropolis–Hastings algorithm [44], addresses the problem of converging on a local, rather than global, optimum by allowing the possibility of accepting inputs to the objective function that produce a less-optimal value. The probability of such an acceptance gradually decreases over successive iterations, which is analogous to the slow cooling and hardening of a molten material. This aspect of simulated annealing is similar to the exploration possibility in ε-greedy RL. However the other differences between RL and hill climbing also apply to simulated annealing; the absence in simulated annealing of something like the RL value function is specifically mentioned in [16].

3 Definition of the problem

This section defines the RL MH problem and gives a formal representation of a realistically complex MH problem in terms of RL elements.

3.1 Terminology

Key terms generally used in manufacturing and in this research are defined as follows:

- *Part* (noun): An individual item that is part of a finished product, i.e., one of the building blocks of a product.
- *Warehouse* (noun): A location in the factory where parts are stored; this location has a maximum capacity for each type of part stored and a current inventory for each type of part stored.
- *Route* (noun): A sequence of locations through which each piece of equipment moves to pick up or deliver parts, or containers of parts.
- *Workstation* (noun): A location where work is performed. Workstations consume one or more types of parts by adding them to a product. Each workstation has a consumption rate for each type of part that it consumes and may have a buffer for holding unconsumed parts.
- *Demand* (noun): The type and number of items to produce. Demand is realized in this research as the rates at which each workstation consumes different type of parts.
- *Plan* (noun): A specific configuration of routes, with assignments of pieces of equipment, scheduled execution times, and the parts to be delivered for each of the routes.
- *Non-consumed parts* (noun): Parts not consumed by a workstation which would have been consumed during the period of plan execution but were not because the workstation had run out of parts. Non-consumed parts measure lost production.
- *Mean parts threshold* (noun): An average number of parts required to keep the workstations running during the plan execution period.

When planning MH processes, this research is primarily interested in four constraints:

- Ensuring that none of the workstations run out of parts.
- Minimizing the number of pieces of equipment needed to deliver parts.
- Minimizing the number of parts stored at each workstation.
- Minimizing the total distance traveled by all pieces of equipment while delivering parts from the warehouse to workstations.

Satisfying the first constraint would be simple if not for other three constraints, and vice-versa, to satisfy the constraints simultaneously can make MH planning challenging.

3.2 Assumptions and simplifications

The following characteristics of the factory MH process are assumed:

- A workstation may consume more than one type of parts.
- The rates at which the workstations consume parts are known.
- Each workstation has a buffer (i.e., temporary storage location) for each type of part that the workstation consumes, and the capacities of those buffers are known.
- A single piece of equipment may transport parts for multiple workstations on a single trip.
- Any piece of MH equipment can physically reach any of the workstations to deliver parts.
- All MH within the factory is included in the MH plan, i.e. parts are not delivered to the workstations other than as specified in the MH plan.

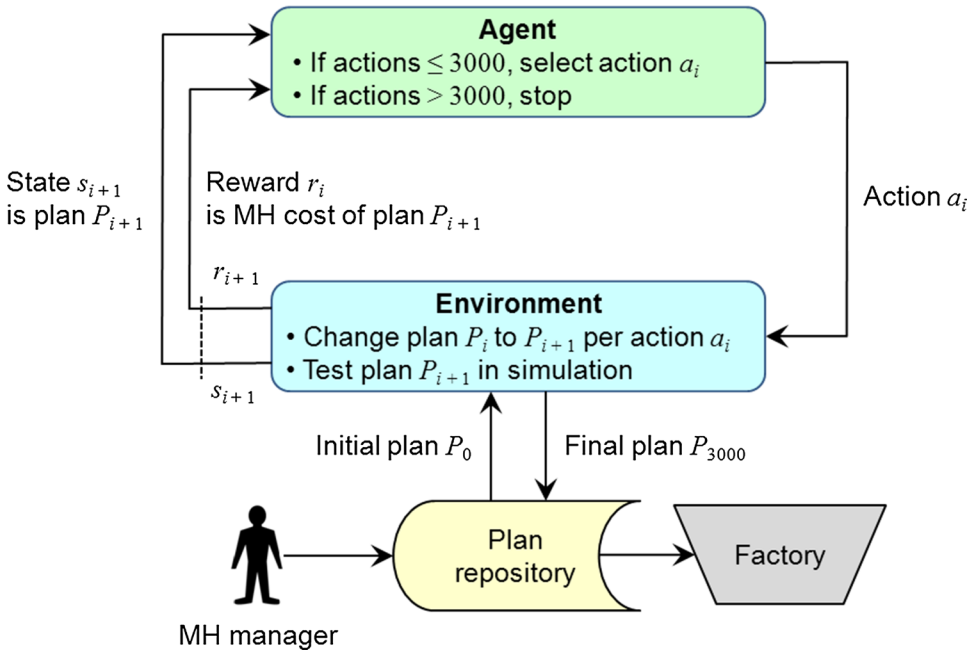

Fig. 4 Material handling reinforcement learning process

- Sufficient MH resources, such as pieces of equipment and the workers to operate the equipment, will be available to execute the MH plan.
- Moving partially completed products between workstations and moving completed products to a shipping area are not part of the MH plan.

The following simplifying assumptions were made; any could be relaxed in future work:

- Workstations and MH equipment do not malfunction or break down.
- Once execution of a MH plan is initiated, it is not pre-empted. There are no preemptive or unscheduled requests for parts from the workstations that affect the MH plan.
- The rates at which the workstations consume parts do not change during execution of an MH plan.[2]
- The time required to unload parts at a workstation does not change during execution of an MH plan.
- The factory layout (i.e., the location of the warehouses, workstations, and connecting walkways) does not change during execution of an MH plan.
- The reward associated with a particular MH plan does not change over time, i.e., the same plan will produce the same reward at any point in a learning sequence.

3.3 MH as an RL problem

The MH reinforcement learning process is shown in Fig. 4. The elements of the MH planning task have been mapped to those of RL in the following way. The RL *environment* is the *factory*, which includes all the workstations, intersections, warehouses, and walkways, all the equipment, and all the different types of parts. The RL *state* is an abstract internal representation of an MH *plan* and its *reward* (to be explained later). The RL *action* is a *change to a plan*. Finally, the RL *reward* is the *cost of a MH plan*.

In the figure, initial MH plans are created manually and stored in the plan repository. A plan learning sequence begins when an initial plan $\boldsymbol{P_0}$ is selected from the repository and tested using an abstract simulation. The resulting reward $\boldsymbol{r_1}$, i.e., the MH cost for plan $\boldsymbol{P_0}$, is calculated by the abstract factory simulation and then passed to the agent along with the state $\boldsymbol{s_1}$, which is determined from the plan and reward. Then the agent checks whether the total

[2] A change in product demand, which could change the workstations' consumption rates, would trigger the generation of a new MH plan, not a change to an MH plan while it was executing.

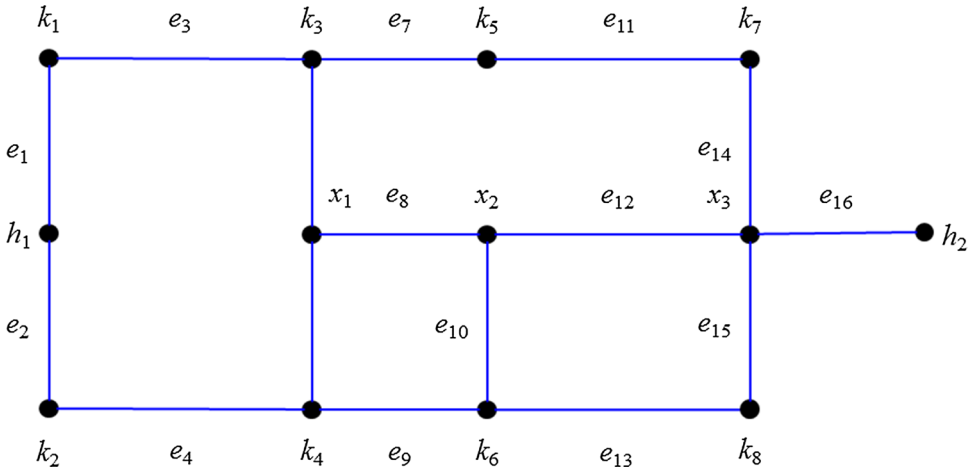

Fig. 5 Example of a factory layout represented as a graph

number of actions (changes to the plan) performed so far is equal to 3000 (this value was chosen empirically, based on preliminary testing). If so, then the learning sequence stops and the plan is moved into the repository. Otherwise, the agent makes another change to the plan in an attempt to improve it. The learning agent is learning in "plan space", i.e., the agent explores the space of possible plans by making small changes to each successive plan in the sequence. RL is employed to help the agent to learn what types of plan changes work best in different situations.

3.4 RL state and environment

To use RL to modify an MH plan, a formal mathematical representation of the factory layout, the material handling scenario, and the MH plan are required. They are represented as follows.

Factory layout The factory layout is represented as a graph $G_F = (V_F, E_F)$ with vertex set V_F and edge set E_F. The vertices represent the workstations, warehouses, and walkway intersections, and the edges represent the walkways which connect the warehouses, workstations, and intersections; the walkways are bidirectional and allow movement in either direction. The vertex set VF can be partitioned into three types of vertices:

- $K = \{k_1, k_2, k_3, \ldots\}$ is the set of workstations; $|K|$ = number of workstations.
- $X = \{x_1, x_2, x_3, \ldots\}$ is the set of intersections; $|X|$ = the number of intersections.
- $H = \{h_1, h_2, h_3, \ldots\}$ is the set of warehouses; $|H|$ = the number of warehouses.

Thus $V_F = K \cup X \cup H$ and $|V_F| = |K| + |X| + |H|$.

Not all pairs of vertices are connected by edges (walkways) and there are no edges from a vertex to itself; thus $E_F \subseteq V_F \times V_F$ and $v_i \neq v_j$ for all $(v_i, v_j) \in E_F$; $|E_F|$ = number of edges.

Graphs have been used to represent factory layouts in prior research, e.g., [45]. A graph provides information needed for planning MH, e.g., how the warehouses and workstations are connected by walkways, and omits unneeded details, e.g., the physical dimensions of a workstation. Figure 5 shows a small example of a factory layout represented as a graph using the notation just defined.

Material handling scenario The representation of the material handling situation has five components, as follows:

- Equipment Q Set of all pieces of equipment available for MH in the factory for a given plan period. $Q = \{q_1, q_2, q_3, \ldots\}$ and q_i = piece of equipment i, $i \in Z^+$; $|Q|$ = total number of different pieces of equipment.
- Part type IT Set of all types of parts available in the factory for MH consumption for a given plan period. $IT = \{it_1, it_2, it_3, \ldots\}$ and it_i = part type i, $i \in Z^+$; $|IT|$ = total number of different types of parts.
- Consumption rate C. Number of parts of each type consumed by each workstation in an hour.

$$C = \begin{bmatrix} c_{11}c_{12} & \cdots & c_{1|IT|} \\ \vdots & \ddots & \vdots \\ c_{|K|1}c_{|K|2} & \cdots & c_{|K||IT|} \end{bmatrix}$$

where c_{ij} = number of parts of type j consumed by workstation i per hour;
$1 \le i \le |K|$ and $1 \le j \le |IT|$.

- Buffer B Number of parts of each type that can be stored at each workstation.

$$B = \begin{bmatrix} b_{11}b_{12} & \cdots & b_{1|IT|} \\ \vdots & \ddots & \vdots \\ b_{|K|1}b_{|K|2} & \cdots & b_{|K||IT|} \end{bmatrix}$$

where b_{ij} = maximum number of parts of type j that can be stored at workstation i;
$1 \le i \le |K|$ and $1 \le j \le |IT|$.

- Plan duration T. Time duration (in minutes) over which the MH plan will be executed.

Plan The material handling plan consists of a set of routes: $P = \{p_1, p_2, p_3, \ldots\}$; where p_i = route i, $i \in Z^+$. Each route p_i has four components, W_i, PE_i, F_i, and D_i, defined as follows:

- Walk (graph theory) W_i: A sequence of vertices in the factory layout such that adjacent vertices are connected by a distinct edge;
  where $W_i = (w_1, w_2, w_3, \ldots, w_n)$, $w_j \in V_F$, $(w_j, w_{j+1}) \in E_F$ for $1 \le j < n - 1$) and $w_1, w_n \in H$. In terms of MH, each complete transit of a walk by a piece of equipment delivering parts is referred to as a *trip*.
- Piece of equipment PE_i: A piece of equipment used in the route i; where $PE_i \in Q$.
- Times trips start F_i: List of times that the trips of route i start;
  where $F_i = (t_1, t_2, t_3, \ldots, t_f)$, $0 \le t_j < T$, $t_j \le t_{j+1}$ for $1 \le j < f$ and inter-trip interval $VF_i = t_{j+1} - t_j$ then, $F_i = (1 \cdot VF_i, 2 \cdot VF_i, 3 \cdot VF_i, \ldots, f \cdot VF_i)$.
  Here the inter-trip interval is a constant, thus the trips are evenly spaced.
  The first trip cannot start before VF_i.
- Deliveries D_i: Single list of triples (i.e., 3-tuples), where
  $D_i = (d_1, d_2, d_3, \ldots, d_k)$, $|D_i| = k$ and $d_j = (w, x, y)$ with w = workstation, x = part type, and y = number of parts, i.e., $w \in K$, $x \in IT$ and $y \in Z^+$.

3.5 RL actions

The actions available to the RL learning agent are ten distinct operations that change an MH plan in ways that are intended to improve it. Those plan change operations, or actions, are defined as follows.

IncreaseTheNumberOfTrips Increase the frequency of the route serving the workstation that had the greatest shortage of parts. Let $d_j \in D_i$ be the delivery in route i of plan P such that $d_j[1]$ is the workstation that has the largest number of non-consumed parts of type $d_j[2]$. This action increases the number of times parts will be delivered to $d_j[1]$ by decreasing the inter-visit interval VF_i, i.e., $VF_i = VF_i - x$ where $0 \le x < T$. The value of x is a constant set to a value found empirically during preliminary testing. If more than one delivery is made in route i, then delivery d_j will be removed from the route i and a new route m will be formed with new trip start times generated using VF_i i.e., $VF_m = VF_i - x$ where $0 \le x < T$. Route m will be added to the plan P. All of the remaining route information such as a piece of equipment is copied from the original route. The walk for the new route is calculated using Warshall's shortest path algorithm [46]. The new route will start from a warehouse and drop off part type $d_j[2]$ at workstation $d_j[1]$ before returning to the warehouse. If none of the workstations in plan P is running out of parts, then this action selects the $d_j \in D_i$, such that $d_j[1]$ is the workstation that has the largest number of extra parts of type $d_j[2]$ stored, and then increases the number of times parts will be delivered to $d_j[1]$.

DecreaseTheNumberOfTrips Decrease the frequency of the route serving the workstation that had the greatest excess of parts. Let $d_j \in D_i$ be the delivery in route i of plan P such that $d_j[1]$ is the workstation that has the largest number of extra parts of type $d_j[2]$ stored. This action reduces the number of times parts will be delivered to $d_j[1]$ by increasing the inter-visit interval VF_i, i.e., $VF_i = VF_i + x$, $0 \leq x < T$. The value of x is a constant set to a value found empirically during preliminary testing. If more than one delivery is made in route i, then delivery d_j will be removed from the route and a new route m will be formed with new trip start times generated using VF_i, i.e., $VF_m = VF_i + x$ where $0 \leq x <$ T. Route m will be added to the plan P. All of the remaining route information, such as a piece of equipment, is copied from the original route. The walk for the new route is calculated using Warshall's shortest path algorithm [46]. The new route will start from the warehouse and drop off part type $d_j[2]$ at workstation $d_j[1]$ and return back to the warehouse. If none of the workstation of plan P has extra parts stored, then this action selects the $d_j \in D_i$ such that $d_j[1]$ is the workstation that has the largest number of non-consumed parts of type $d_j[2]$ and decreases the number of times parts will be delivered to $d_j[1]$.

IncreaseTheNumberOfPartsDelivered Increase the number of parts delivered per trip to the workstations that had the greatest shortage of parts. Let $d_j \in D_i$ be the delivery in route i of plan P such that $d_j[1]$ is the workstation that has the largest number of non-consumed parts of type $d_j[2]$. This action increases the number of parts of type $d_j[2]$ delivered to $d_j[1]$ by increasing the number of parts $d_j[3] = d_j[3] + m$ where $m \in Z^+$, $d_j \in D_i$. The value of m is a constant set to a value found empirically during preliminary testing. If none of the workstations in plan P are running out of parts, then this actions picks the $d_j \in D_i$ such that $d_j[1]$ is the workstation that has the largest number of extra parts of type $d_j[2]$ stored and increases the number of parts of type $d_j[2]$ delivered to $d_j[1]$. (Recall the definition given earlier; "non-consumed parts" are parts that would have been consumed had the workstation not run out of parts.)

DecreaseTheNumberOfPartsDelivered Decrease the number of parts delivered per trip to the workstations that had the greatest excess of parts. Let $d_j \in D_i$ be the delivery in route i of plan P such that $d_j[1]$ is the workstation that has the largest number of extra parts of type $d_j[2]$ is stored. This action decreases the number of parts of type $d_j[2]$ delivered to $d_j[1]$ by decreasing the number of parts $d_j[3] = d_j[3] - m$ where $m \in Z^+$, $d_j \in D_i$. The value of m is a constant set to a value found empirically during preliminary testing. If none of the workstations in plan P have extra parts stored, this action picks workstation $d_j \in D_i$ such that $d_j[1]$ is the workstation that has the largest number of non-consumed parts of type $d_j[2]$ and decreases the number parts of type $d_j[2]$ delivered to $d_j[1]$.

ChangeNumberAndOrTypeOfEquipment Split the route serving the workstation with the greatest shortage of parts into two routes, and assign a piece of equipment to the new route. Let $d_j \in D_i$ be the delivery in route i of plan P such that $d_j[1]$ is the workstation that has the largest number of non-consumed parts of type $d_j[2]$. If more than one delivery is made in the route i and at least one workstation is running out of parts in that route i, then this action splits the route i into two routes. An unassigned piece of equipment from Q is assigned to the second route. All of the remaining route information, such as trip start times, is copied from the original route. If there is only one delivery in route i or none of the workstations of D_i in route i are running out of parts, then this action tries to remove one of the deliveries from route i and adds it to any other route in plan P which has same type of equipment as route i. This action requires sorting the routes of plan P in descending order of number of different type of parts on the pieces of equipment of each route. This action tries to reduce the number of pieces of equipment used in the plan P.

MergeWithTheRouteWithSameWorkstation Remove a delivery to a workstation from one route and add it to another route that already includes a delivery to that workstation. Let i be the route with the smallest number of workstations to be served in the plan P. Remove $d_j \in D_i$ in route i in the plan P from the beginning of D_i and add it to route x such that x is also delivering the parts to the same workstation as $d_j[1]$ (as each workstation can consume multiple types of parts), i.e., $d_j[1] = d_h[1]$ where $d_j \in D_i$, $d_h \in D_x$ and $|D_x| \geq |D_i|$. If more than one such route exists (such as route x), then this action selects the very first one in the sorted route list. This action requires sorting the routes of plan P in descending order of number of different type of parts on the pieces of equipment of each route. If the equipment capacity and equipment type do not permit adding the selected workstation $d_j[1]$ to any other route in the sorted plan P, this action will consider the next delivery d_{j+1} in the list of D_i in the same route i. This action tries to reduce the number of pieces of equipment used in the plan P.

MergeWithNeighborWorkstation Remove a delivery to a workstation from one route and add it to another route that already includes a delivery to a nearby workstation. Let i be the route with the smallest number of workstations to be served in the plan P. This action removes delivery $d_j \in D_i$ in route i in the plan P from the beginning of D_i and adds it to route x such that at least one workstation $d_h[1]$ of D_x is the neighbor of $d_j[1]$ of D_i, i.e., $(d_j[1], d_h[1]) \in E_F$ or if $d_j[1]$ and $d_h[1]$ have a path connected only by one or more intersections, where $d_j \in D_i$ and $d_h \in D_x$. If more than one such route exists (such as route x), then this action selects the very first one in the sorted route list. This action requires sorting the routes of plan P in descending order of number of different type of parts on the pieces of equipment of each route. If the

equipment capacity and equipment type do not permit adding the selected delivery d_j to any other route in the sorted plan P, the action will consider the next delivery d_{j+1} in the list of D_i in the same route i. This action tries to reduce the number of pieces of equipment used in the plan P.

MergeWithTheRouteWithinTheWalk Remove a delivery to a workstation from one route and add it to another route that already passes by that workstation. Let i be the route with the smallest number of deliveries D_i to be served in the plan P. This action removes delivery $d_j \in D_i$ in route i in the plan P from the beginning of D_i and adds it to route x if workstation $d_j[1]$ is in the walk of route x, i.e., $W_i \cap W_x = d_j[1]$ where $d_j \in D_i$. If more than one such route exists (such as route x), then this action selects the very first one in the sorted route list. This action requires sorting the routes of plan P in descending order of number of different type of parts on the pieces of equipment of each route. If the equipment capacity and equipment type do not permit adding the selected delivery d_j to any other route in the plan P, the action will consider the next delivery d_{j+1} in the list of D_i in the same route i. This action tries to reduce the number of pieces of equipment used in the plan P.

MergeWithRouteArbitrarily Remove a delivery to a workstation from one route and add it to another route arbitrarily, i.e., without any other conditions. For any arbitrarily selected delivery $d_j \in D_i$ in route i in the plan P, this action removes the delivery d_j from route i and adds it to the arbitrarily selected route x. Route x must have the same equipment type as route i and must have capacity to carry part $d_j[2]$.

SplitTheRouteArbitrarily Remove a delivery to a workstation from one route and create a new route that includes only that delivery arbitrarily, i.e., without any other conditions. For any arbitrarily selected delivery $d_j \in D_i$ in route i in the plan P, split route i at delivery j, create a new route x, and then add it into the plan P. All of the remaining information, such as trip start times, is copied from the original route.

Some of the actions require selecting a particular workstation or route in a plan P to which the action is applied, that selection is done by reviewing the information from the abstract simulation of the plan (see next section for further explanation).

3.6 RL reward

As mentioned earlier, in RL the result of selecting and performing an action in a given state is both a new state and a reward; the latter indicates the value of the selected action in the given state. In this work, the reward is an assessment of the new MH plan produced by performing a plan change operation. Clearly, assessing MH plans by testing them in an actual factory would be impractical, so an alternative method must be found. This section describes how the rewards are calculated using a simulation-based multi-objective reward function.

3.6.1 Reward model

The reward is calculated as the cost of executing the MH plan for a one six-hour shift. The cost is calculated using a computer simulation of MH developed specifically for this purpose. (A similar approach, using a computer simulation to evaluate flexible flow shop schedules, is described in [47].) The simulation model is based on concepts from discrete event simulation (DES) [48], but is highly abstract. For example, detailed occurrences that might be separate events in a conventional DES model, such as visiting each workstation in a route, are abstracted into a single event, namely executing the route. The model uses a Future Event List (FEL) which is initialized using the times trips start list of all the routes of the plan P. The model executes the plan by performing each event in the FEL. Because the FEL for a given plan P is initialized completely at the beginning of a simulation, later events are not scheduled by earlier events, as is typical in conventional DES.

The main activity of the simulation is route execution, i.e., starting from the warehouse, visiting each of the workstations on the route, and returning to the warehouse. This is done for each route in a plan P. The time required to execute a route is deterministic because it is calculated directly from the distances between the successive pairs of vertices in the route. The reward model maintains the current time and distance lists for each distinct piece of equipment used in the plan P, and it maintains a separate list for the sum of parts stored and sum of non-consumed parts for each workstation and part type in the plan P. These lists record the data at every visit to a particular workstation to drop off a particular type of part.

The plan is simulated to determine the actual time spent visiting each workstation and to record the sum of stored parts and non-consumed parts, which is used to calculate the reward, i.e., the MH cost. During a simulation the model generates additional information that can be used to fill in the details of the next action selected.

Each simulation runs for T minutes of simulated time; in the experiments reported later, $T = 360$ (one six-hour shift). All the routes of the plan are assumed to have already been executed once and their parts delivered at the start of the simulation so that none of the workstations will run out of parts immediately at the beginning of the simulation. The routes are also executed one additional time at the end of the simulation without calculating the visiting times and distance, in order to record the sum of stored parts and the sum of non-consumed parts for each workstation-part type.

3.6.2 Notations used for calculating RL reward for MH plan *P*

The following notation is used to define the details of the reward model.

$m(a, b)$	Euclidian distance between two adjacent vertices in the factory, such that $a, b \in V_F$ and $(a, b) \in E_F$; $m(a, b) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$.
s_v	Speed of a piece of equipment in plan P (2.235 m/s)
$t(a, b)$	$m(a, b) / s_v$ = travel time between two vertices in the factory, such that $a, b \in V_F$ and $(a, b) \in E_F$
M	Total distance to execute a given plan P
$V_q(t, g)$	Current time t of a piece of equipment $g \in Q$ in a plan P
$m_r(i)$	Total distance for executing route i
$t_r(i)$	Total time for executing route i
$V_d(i, j, h)$	Last visiting time to workstation j to drop off part type k of route i where $j \in K$ and $h \in IT$
$P_d(i, j, h)$	Total parts of type h stored at workstation j during the last visit to workstation j of route i where $j \in K$ and $h \in IT$
$A(i, j, h)$	Number of parts of type h consumed by workstation j since the last visit to workstation j of route i where $j \in K$ and $h \in IT$
$T_d(i, j, h)$	Number of parts of type h expected to be consumed by workstation j since the last visit to workstation j of route i where $j \in K$ and $h \in IT$; calculated using consumption rate C
$B(i, j, h)$	Number of parts of type h currently at workstation j of route i after every visit where $j \in K$ and $h \in IT$
$Z(i, j, h)$	List $(z_1, z_2, z_3, \ldots, z_{\lvert Fi \rvert + 1})$ of the sum of the number of parts of type h stored at every visit to workstation j of route i where $j \in K$ and $h \in IT$
$V_c(i, j, h)$	Number of visits to workstation j to drop off part type h of route i where $j \in K$ and $h \in IT$
$H_d(i, j, h)$	Total extra parts of type h stored at workstation j of route i since the last visit to workstation j of route i where $j \in K$ and $h \in IT$
V_q	Total number of distinct pieces of equipment used in the plan P
Wp	Total number of deliveries in the given plan P
$Y(i, j, h)$	List $(y_1, y_2, y_3, \ldots, y_{\lvert Fi \rvert + 1})$ of the sum of non-consumed parts of type h at every visit to workstation j of route i where $j \in K$ and $h \in IT$
$Plty_d(i, j, h)$	Total number of non-consumed parts (defined in the next section) of type h of workstation j of route i, where $j \in K$ and $h \in IT$.

3.6.3 Reward response variables

Several response variables produced by the abstract MH model when simulating a plan are used to calculate the RL reward. Those response variables are:

- **Distance travelled *M***: Total distance (in meters) travelled by all pieces of equipment in the plan P to deliver the parts from the warehouse to the workstations during a given plan period T. This includes the distance travelled after the parts are delivered to the last workstation and the equipment returns to the warehouse. An example of a typical route starting from the warehouse $h_1 \in H$ and visiting four workstations ($k_1, k_2, k_3, k_4 \in K$) and returning back to the warehouse in dotted line is shown in Fig. 6. The total distance for executing route i in the figure is

$$m_r(i) = m(h_1, k_2) + m(k_2, k_4) + m(k_4, x_1) + m(x_1, k_3) + m(k_3, k_1) + m(k_1, h_1) \tag{3}$$

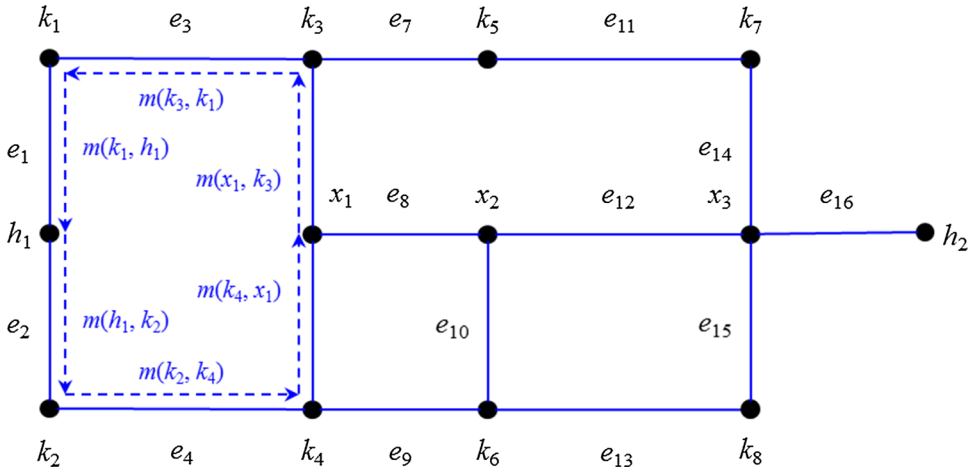

Fig. 6 Sample route

where $m(h_1, k_2)$ = distance between warehouse $h_1 \in H$ and the first workstation $k_2 \in K$ in the route, and $m(k_1, h_1)$ = distance between the last workstation $k_1 \in K$ back to warehouse $h_1 \in H$.

Similarly, the total time for executing route i in the figure is

$$t_r(i) = t(h_1, k_2) + t(k_2, k_4) + t(k_4, x_1) + t(x_1, k_3) + t(k_3, k_1) + t(k_1, h_1) \tag{4}$$

where $t(h_1, k_2)$ = travel time between warehouse $h_1 \in H$ and the first workstation $k_2 \in K$ in the route, and $t(k_1, h_1)$ = travel time between the last workstation $k_1 \in K$ back to warehouse $h_1 \in H$.

- **Equipment utilization $V_{utilization}$**: Total number of distinct pieces of equipment needed to execute the given plan P.
- **Mean parts *V***: Total time averaged quantity of all the parts that have been delivered to all the workstations but have not yet been used in production and are therefore stored in the workstation's part storage buffer, i.e., the average of all the parts stored at all the workstations at any given time within a given plan period T. The minimum number of parts stored at each workstation should be sufficient to keep the workstation from running out of parts before the next delivery.
- **Non-consumed parts *Penalty*:** Number of parts that would have been consumed by all workstations had they been operating at their normal rate for the entire plan period T, but were not consumed because a workstation had run out of parts. If the MH plan has not delivered sufficient parts to the workstations to keep up with their consumption rates and one or more workstations have run out of parts then production stops. Production stoppages are expensive and disruptive. Therefore, the *Penalty* response variable is heavily weighted in the reward calculation and avoiding non-consumed parts due to stoppages is a goal of the learning process.

3.6.4 Weighted sum method

Because there are multiple reward response variables, the weighted sum method [49] was used to scalarize the vector of reward response variables so that the RL method can be applied. The weighted sum method is one of the most commonly used approaches for scalarizing a vector reward. As the name describes, each four response variables in the reward vector is weighted according to the importance of that variable to the final reward. In general, a single scalar RL reward can be calculated as

$$R = \sum_{i=1}^{j} Re_i \cdot N_i \cdot w_i \tag{5}$$

where j = the number of response variables in the reward vector, Re_i = the value of reward response variable i, N_i = the cost per unit of the reward response variable i, and w_i = the weight of reward response variable i.

As defined earlier, the four reward response variables were M, $V_{utilization}$, V, and *Penalty*. Thus the expanded formula for reward R is

$$R = (M \cdot N_1 \cdot w_1) + (V_{utilization} \cdot N_2 \cdot w_2) + (V \cdot N_3 \cdot w_3) + (Penalty \cdot N_4 \cdot w_4) \quad (6)$$

The four reward response variables are not all in the same units; distance travelled M is in meters, equipment utilization $V_{utilization}$ is in number of pieces of equipment, mean parts V is in number of parts, and non-consumed parts *Penalty* is also in number of parts. Therefore, the reward response variables' values are normalized to use cost as a common unit. Each of the response variables' values are converted to a cost by multiplying the variable's value by a quantity representing the cost of that variable's unit; thus M is multiplied by cost per meter travelled, V is multiplied by cost per part per unit time stored at a workstation, $V_{utilization}$ is measured as the cost per piece of equipment used, and *Penalty* is measured as cost per part that a workstation ran out of and thus was unable to consume during the execution of an MH plan. The four cost normalization multipliers are denoted N_i in Eq. (5) and N_1, N_2, N_3, and N_4 in Eq. (6). The values for the four cost multipliers were estimated by subject matter experts based on historical data from the Steelcase factory used as the example in this work. Similarly, the weights w_1, w_2, w_3, w_4 that were used to represent the relative importance of the four response variables were elicited from subject matter experts in the same facility.[3]

Importantly, note that the calculated reward is a cost, which MH managers want to reduce, thus the RL algorithm seeks to *minimize* the reward, not *maximize* it.

Figure 7 gives the pseudocode of the reward calculation, i.e., the abstract simulation of the MH plan. Lines 1–7 create the FEL from the times trips start (TS_i) for each route i of plan P. Each event notice consists of the piece of equipment PE_i serving route i, the route number, and the time at which each trip is supposed to start. The trip time of each route can be reviewed without creating the list, although the list is convenient for implementation. Next, the list is sorted in increasing order of the time the trip starts, as shown in line 8. Then, all variables of the each workstation and part type are initialized as shown in lines 9–12. The first trip to each workstation starts at minute 0 of the simulation. Lines 13–38 model the actual execution of the route. All the relevant data for computing the RL reward R is recorded in these steps. For each route i and each workstation j on i, lines 15–16 calculate and update the total distance travelled and the current time of PE_i. Then, lines 28 and 31–35 calculate and update the total sum of parts of type h stored at workstation j since the last visit and the sum of non-consumed parts of type h by workstation j since the last visit, respectively. The total mean parts V and non-consumed parts *Penalty* variables for the entire plan P are calculated at lines 39–45. Then, at line 46, the total RL reward, which is the MH cost, is calculated using the weighted sum method.

4 Learning process

In general, the goal of RL is to find a policy that optimizes (minimizes or maximizes) total reward of time. As stated in [16], finding an optimum policy is difficult, even for relatively simple tasks such as board games (which are much simpler than the MH problem), because of computational costs and memory constraints. Therefore, the goal of this research was not to find a strictly optimal policy, but rather to find an effective policy that produced improved (less costly) MH plans for a given demand.

Because this work sought to generate an effective policy which could simultaneously optimize multiple reward response variables, many improved plans were generated which together form a Pareto front [32]. Certain plans minimize distance travelled, others minimize mean stored parts, whereas others minimized equipment usage. The ability to generate multiple optimal plans may be considered an advantage of the RL method in that MH managers can select from among the generated plans the one most best suited for their requirements.

4.1 Temporal difference learning

The RL MH problem is a continuing problem (i.e., a continual process control problem) as defined in [16] because a definite stopping condition is not available. A method to predict the state that will result from taking a particular action in the current state is not available, thus the probability of any particular state being the next state is unknown. Therefore, a model-free RL learning method is used to learn a policy that will improve an MH plan. In particular, tabular Sarsa(0), a temporal difference algorithm with one-step look ahead, was used [16].

[3] In this work the values for the cost multipliers N_1, N_2, N_3, and N_4 are in U. S. dollars and those values, as well as the weights w_1, w_2, w_3, and w_4, are specific to the example Steelcase factory. Other manufacturing facilities will likely have different cost multipliers and weights.

```
1.  // Create FEL from plan P
2.  for i ← 1 to |P|
3.      for x ← 0 to |TS_i|
4.          evt ← (PEi, p_i, t_x)
5.          FEL.insert(evt)
6.      end for
7.  end for
8.  Sort(FEL)
9.  ∀ i:  m_r(i) ← 0, t_r(i) ← 0
10. ∀ t, g:  V_q(t, g) ← 0
11. ∀ i, j, h:  P_d(i, j, h) ← d_j[3] ∈ D_i, V_c(i, j, h) ← 1, V_d(i, j, h) ← 0, A(i, j, h) ← 0
12. z_1 ∈ Z(i, j, h) ← 0; y_1 ∈ Y(i, j, h) ← 0; M ← 0
13. for i ← 1 to FEL.size()
14.     for j ← 1 to |D_i|
15.         M ← M + m(j − 1, j)
16.         V_q(t, g) ← V_q(t, g) + m(j − 1, j) / s_v
17.         T_d(i, j, h) ← (V_q(t, g) − V_d(i, j, h)) · C_jh
18.         H_d(i, j, h) ← P_d(i, j, h) − T_d(i, j, h)
19.         if H_d(i, j, h) ≤ 0 then
20.             B(i, j, h) ← d_j[3]
21.             A(i, j, h) ← P_d(i, j, h)
22.         else
23.             B(i, j, h) ← d_j[3] + H_d(i, j, h)
24.             A(i, j, h) ← T_d(i, j, h)
25.         end if
26.         T_n ← A(i, j, h)
27.         a_1 ← P_d(i, j, h); d ← C_jh; n ← −((t_n − a_1)/(d − 1))
28.         Sum ← Z(i, j, h)[(V_c(i, j, k))] ← (n/2)(2a_1 + (n − 1)d)  // part-time units
29.         P_d(i, j, h) ← B(i, j, h)
30.         V_d(i, j, h) ← V_q(t, g)
31.         if H_d(i, j, h) < 0 then
32.             Y(i, j, h) [(V_c(i, j, h))] ← −H_d(i, j, h)
33.         else
34.             Y(i, j, h) [V_c(i, j, h)] ← 0
35.         end if
36.         V_c(i, j, h) ← V_c(i, j, h) + 1
37.     end for
38. endfor
39. V ← 0; Penalty ← 0
40. for j ← 1 to W_p
41.     Z_avg(i, j, h) ← (∑_{d=1}^{d=|Z(i,j,h)|} Z(i, j, h)[d]) / |Z(i, j, h)|
42.     V ← V + Z_avg(i, j, h) / T
43.     Plty_d(i, j, h) ← ∑_{d=1}^{d=|Y(i,j,h)|} Y(i, j, h)[d]
44.     Penalty ← Penalty + Plty_d(i, j, h)
45. end for
46. R ← (w_1 · M · N_1) + (w_2 · V_utilization · N_2) + (w_3 · V · N_3) + (w_4 · Penalty · N_4)  // reward
```

Fig. 7 Reward calculation pseudocode

At the outset of this work, our initial intention was to use the entire MH plan as the RL state. However, an MH plan contains many variables, each with many possible values (see Sect. 3.3); consequently, the number of possible different states, i.e., the state space of possible MH plans, is extremely large. Overly large state spaces are in general problematic for RL because the learning process is not able to visit each state often enough to try all of the actions available in each state and thereby learn an effective policy for each state.

Therefore, to reduce the size of the state space, the RL state was calculated from values of variables in the reward function, rather than using the MH plan itself as the state. (This is quite similar to function approximation in RL using features of the state [16], although in this work the state was calculated from the reward function's output.) Several different ways

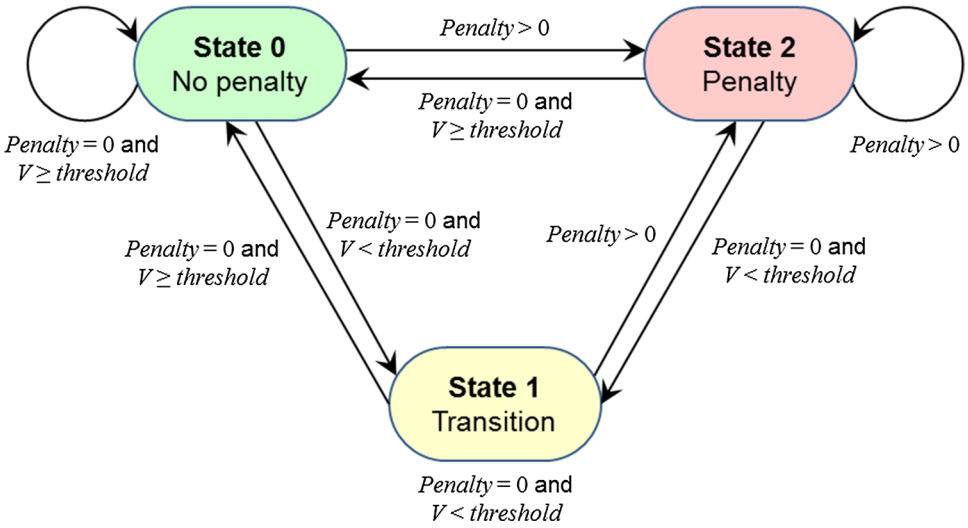

Fig. 8 RL state transition diagram

of doing so, with differing numbers of states, were manually devised based on knowledge of the application and then empirically tested. The key challenge was to identify a set of features that were simpler than the complete MH plan, but contained essential information needed to help the RL algorithm select an action that would improve the MH plan. After several iterations with state calculation schemes that did not perform well, testing with a simple scheme with two states (No penalty and Penalty) showed some promise, but clearly had room for improvement. Close examination of the two-state experimental results revealed that the RL algorithm would sometimes choose actions that overcorrected, switching too drastically from actions appropriate for a No penalty state to actions appropriate for a Penalty state, or vice versa. From that insight, a third state representing a transitional state "between" No penalty and Penalty was added. These states were used in the result report here:

- State 0, No penalty.*Penalty* = 0 with $V \geq$ mean parts threshold.
- State 1, Transition.*Penalty* = 0 with $V <$ mean parts threshold.
- State 2, Penalty.*Penalty* > 0.

Figure 8 illustrates these three states and the possible transitions between them. For example, from the No Penalty state, if the agent selects an action that changes the MH plan in a way that the reward model when executed for the changed plan returns a reward with a non-zero *Penalty* value in the reward vector, then transitions to the Penalty state. If the reward model instead returns a reward with a zero *Penalty* value and with the mean parts variable greater than the mean parts threshold, then the state remains in the No penalty state. Finally, if the reward model returns a reward with a zero *Penalty* value and with the mean parts variable less than the mean parts threshold, then the next state is the Transition state.

The Transition state was added after preliminary testing showed that using just two states, No penalty and Penalty, was ineffective. With only two states, it was possible that actions that are normally positive, such as reducing the number of parts delivered to a workstation so as to reduce the cost associated with storing excess parts, could be invoked when the number of parts at a workstation was very close to the minimum level required to avoid running out of parts. In that situation the action would reduce the parts enough to cause a workstation (or workstations) to run out of parts, producing a non-zero *Penalty* value, and the action would receive a large negative reinforcement, causing it to not be selected again, in spite of the fact that for many plans it would often be effective at reducing the cost. The Transition state provided a means for the learning process to learn to distinguish between situations when such actions are desirable and situations when they are not.

The best value for mean parts threshold (labeled "threshold" in Fig. 8) is learned along with the policy by the RL process. The threshold value was set to zero initially, then was updated during execution as the moving average of the mean parts *V* stored whenever a transition into the Penalty state occurred. This eliminated the need to select an empirical value for the means parts threshold, and allows the algorithm to be applied to any demand.

1. Initialize $q_\pi(P, s, a) \leftarrow 0.5\ \forall\ s \in S, a \in A(s)$
2. $\alpha \leftarrow 0.1; \gamma \leftarrow 0.7; \varepsilon \leftarrow 0.3$
3. Calculate initial state s_0 from initial plan P_0 (see Table 3 for initial plans)
4. Randomly choose action a_0
5. **for** $i \leftarrow 0$ to 2999 // 3000 actions
6. Apply action a_i to P_i to produce plan P_{i+1}
7. Calculate reward r_{i+1} from P_{i+1} using reward model described in Figure 7
8. Calculate state s_{i+1} from reward r_{i+1}
9. // Choose next action a_{i+1} using ε-greedy policy
10. **if** rand() > ε *then*
11. Choose a_{i+1} with maximum action value from $A(s_{i+1})$
12. **else**
13. Choose $a_{i+1} \in A(s_{i+1})$ randomly
14. **end if**
15. Calculate actual reward $ar \leftarrow r_i - r_{i+1}$
16. // Update the value of the previous state action pair
17. $q_\pi(P_i, s_i, a_i) \leftarrow q_\pi(P_i, s_i, a_i) + \alpha[a_r + \gamma(q_\pi(P_{i+1}, s_{i+1}, a_{i+1}) - q_\pi(P_i, s_i, a_i))]$
18. // Update the values for the next iteration
19. $P_i \leftarrow P_{i+1}; s_i \leftarrow s_{i+1}; a_i \leftarrow a_{i+1}$
20. **end for**

Fig. 9 RL plan learning pseudocode

Figure 9 gives the pseudocode of the RL MH plan learning algorithm. The learning process starts by selecting a naive initial plan P_0, initializing the action values to 0.5, and setting the values for the RL Q-learning parameters learning rate α, discount rate γ, and ε-greedy random action selection probability ε in line 1. (The specific values for α, γ, and ε used in this work were chosen based on guidance from the RL literature and preliminary testing.) Then the algorithm calculates the reward for the initial plan by executing the abstract simulation (described in Sect. 3.6) and sets the initial state (line 3).

In the initial state s_i, the agent takes action a_i based on the ε-greedy policy to produce a new plan P_{i+1} as indicated in lines 6–7. From the new plan P_{i+1}, the simulation is used to calculate the reward r_{i+1} and new state s_{i+1} as shown in line 8. The ε-greedy action selection process is used to select a new action a_{i+1} from the new state as shown in lines 9–13.

Instead of using the direct MH cost to update the action value, the algorithm uses the difference of the previous and the current immediate reward as an actual reward as shown in line 14. Then the previous action value is updated using the new action value and the MH cost from the actual reward (line 15 in Fig. 9). The current plan, state, and action are updated as shown in line 19. This process repeats until 3000 actions have been taken. The number of actions per run was set at 3000 based on preliminary testing, which showed that in almost all runs the reward had stabilized and the agent had explored the actions available in each state often enough to learn which actions were useful by the time 3000 actions had been completed.

4.2 Observations from the reward calculation

As described earlier, the reward is calculated using a simulation. That simulation produces considerable information beyond simply the values of the four response variables that are used to calculate the reward. For example, because the simulation computes the details of the MH process at discrete time steps, a time series for each of the factory variables is available. Some information available from the simulation, such as the identity of a specific workstation that runs out of parts and the type of parts it runs out of, or time series of parts stored at each workstation, are potentially very useful for improving the MH plan. A feature of this work is to exploit that additional information within the context of RL. The additional data produced by the simulation is retained in a list. For practical applications of RL, such as the MH application studied here, the additional information can reduce the size of the state space by retaining the additional information as supplementary information with each state, rather encoding it all as the state.

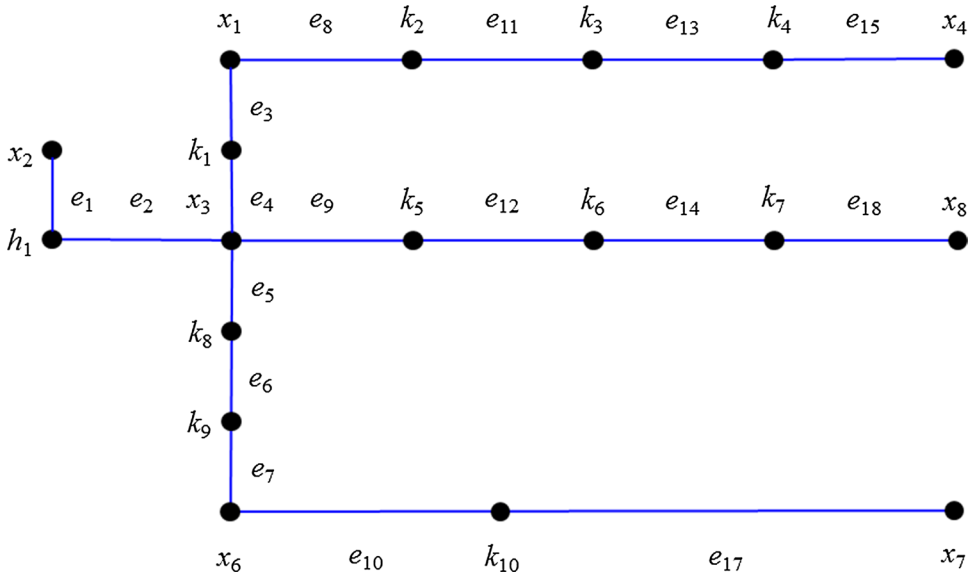

Fig. 10 Factory layout 1

Table 1 Equipment capacity limits

Equipment type	Number	Part type limit
Tugger	3	11
Forklift	3	4
Human puller	3	7

4.3 Action selection

The information produced by the abstract simulation is exploited in the action selection process. That process has two stages: first, an action is selected from among those available in the state, and second, the specific parameters of the chosen action are set. The stages proceed as follows:

1. *Select the action*. One of the actions available in the current state is chosen using the conventional RL ε-greedy method and the current action values learned by the RL process for those actions.
2. *Set the action's parameters*. As described earlier (Sect. 3.5), each of the actions has a number of specific parameters. For example, action *IncreaseTheNumberOfPartsDelivered* has two parameters: part type and the number of parts to be increased. Once an action has been chosen, the specific values of that action's parameters are calculated by heuristics using the additional data produced by the abstract simulation. The action parameters' calculations are specific to each action, but all actions use the data produced by the abstract simulation. For example, for *IncreaseTheNumberOfPartsDelivered*, the part type and number of parts are calculated using the penalty list $Y(i, j, h)$ from the abstract simulation.

5 Experiments, results, and analysis for factory layout 1

This section reports and analyzes the experimental results for the first factory layout. Factory layout 1 is the smaller and simpler of the two factory layouts. As shown in Fig. 10, factory layout 1 consisted of 10 workstations (k_1, k_2, k_3, …, k_{10}), seven intersections (x_1, x_2, x_3, …, x_7), and one warehouse (h_1).

The ten workstations consume a total of 20 different types of parts. Each workstation had a temporary storage buffer for parts that it has not consumed yet. The buffers' capacity for each part type was twice the number of parts of that type consumed by the workstation in an hour.

Table 2 Workstations, consumption rate, and allowable equipment for each part type for factory layout 1.

Part type	Workstation name	Consumption per hour	Equipment that can transport part type
it_1	k_4	66	Tugger
it_2	k_3	89	Forklift
it_3	k_2	92	Human puller
it_4	k_1	25	Tugger, Forklift
it_5	k_5	40	Tugger, Human puller
it_6	k_6	66	Tugger
it_7	k_7	20	Forklift
it_8	k_8	15	Human puller
it_9	k_9	22	Tugger, Forklift
it_{10}	k_{10}	30	Tugger, Human puller
it_{11}	k_4	79	Tugger
it_{12}	k_3	37	Tugger
it_{13}	k_2	88	Forklift
it_{14}	k_1	49	Forklift
it_{15}	k_5	38	Human puller
it_{16}	k_6	28	Human puller
it_{17}	k_7	55	Human puller
it_{18}	k_8	40	Tugger, Human puller
it_{19}	k_9	30	Tugger, Human puller
it_{20}	k_{10}	85	Tugger, Human puller

Table 3 Initial MH plans

Initial plan	Initial number of parts of each type	Initial number of part types served per route	Visit frequency per route	Starting state
1	2	1	12	No penalty
2	0.2	1	12	Penalty
3	2	>1	12	No penalty
4	0.2	>1	12	Penalty
5	0.2	1	24	Penalty
6	0.2	>1	24	Penalty

Three types of equipment were used: tugger, forklift, and human puller. Each type of equipment could carry multiple parts of the same or of different types. Each equipment type was limited in the number of different part types it could carry; those limits are shown in Table 1. Nine pieces of equipment (three of each type) were used in total, and each part type could be carried on multiple types of equipment. In the current implementation, pieces of equipment are limited in terms of different types of parts they can transport, but not in the number of parts.

Table 2 shows the consumption rate of each part type by each workstation and the pieces of equipment that can be used to transport parts of that part type. In an actual application, the consumption rates would be derived from the demand prior to the start of the MH planning process; during the process they are constant (see Table 2). The consumption rates and equipment combination are generated randomly for the experiments. The equipment type combination and equipment limit on number of part types must be checked while moving the delivery from one route to another during the execution of a plan change action to ensure no physical constraints are violated.

The RL MH plan improvement process was tested with six different initial MH plans; those initial plans are summarized in Table 3. Recall that Table 3 describes the initial MH plans; the learning process is intended to modify an initial plan in ways that will improve it. In the table, the first column identifies the initial plan. The second column shows how many parts are present in each workstation's temporary parts buffer at the beginning of the shift; the two values in the column, 2.0 and 0.2, indicate 200% and 20% of the workstation's consumption rate of each part type respectively. The third column specifies the number of the deliveries present in each route of the plan. If the value in the column is 1, each route in the

Table 4 Weights and costs of the reward response variables

Response variable	Weight	Cost per unit ($)
Distance travelled *M*	0.2	0.02
Equipment utilization $V_{utilization}$	2	100
Mean parts *V*	0.8	1
Unconsumed parts *Penalty*	5	1

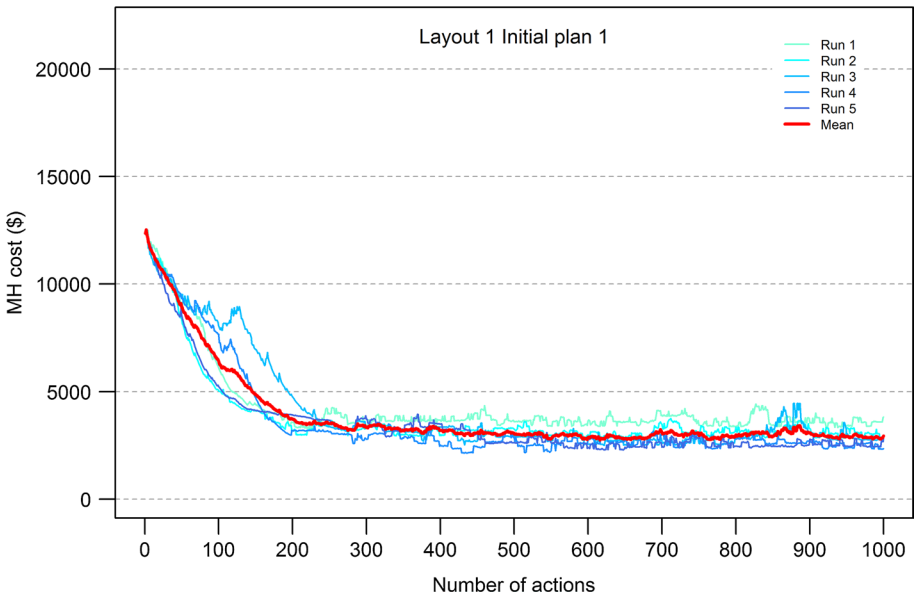

Fig. 11 MH cost improvement for initial plan 1 on factory layout 1

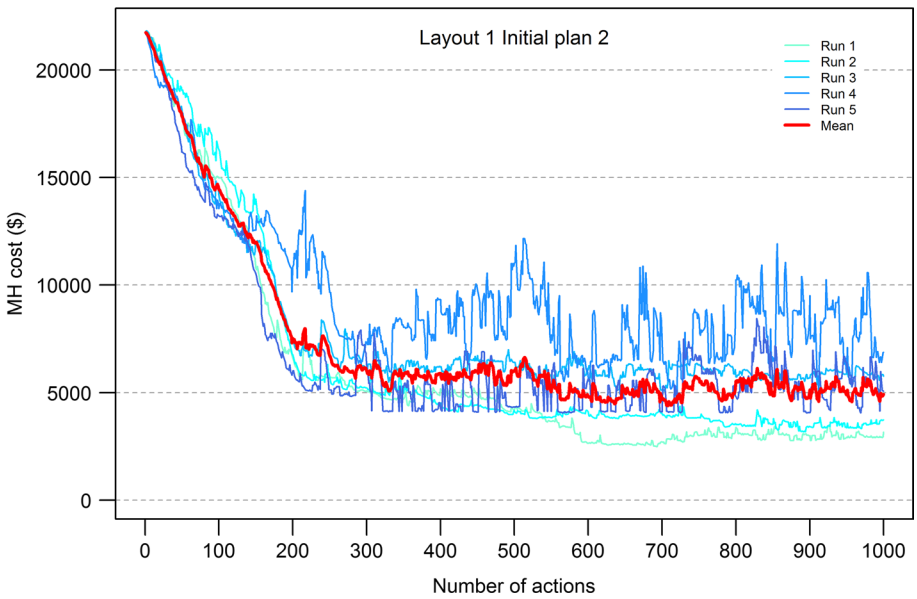

Fig. 12 MH cost improvement for initial plan 2 on factory layout 1

plan consists of one piece of the equipment delivering one type of part to one workstation. If the value is > 1, then a route may include multiple deliveries initially. The fourth column shows the frequency at which the workstation is visited; the two values 12 and 24 correspond to deliveries of parts to the workstation every 30 min and every 15 min respectively. Finally, the fifth column shows the state the learning process starts in for that plan.

Table 4 shows the weights and cost per unit of response variables used in the experiment. The weights of the response variables were elicited from material handling subject matter experts at the Steelcase office furniture factory in Athens

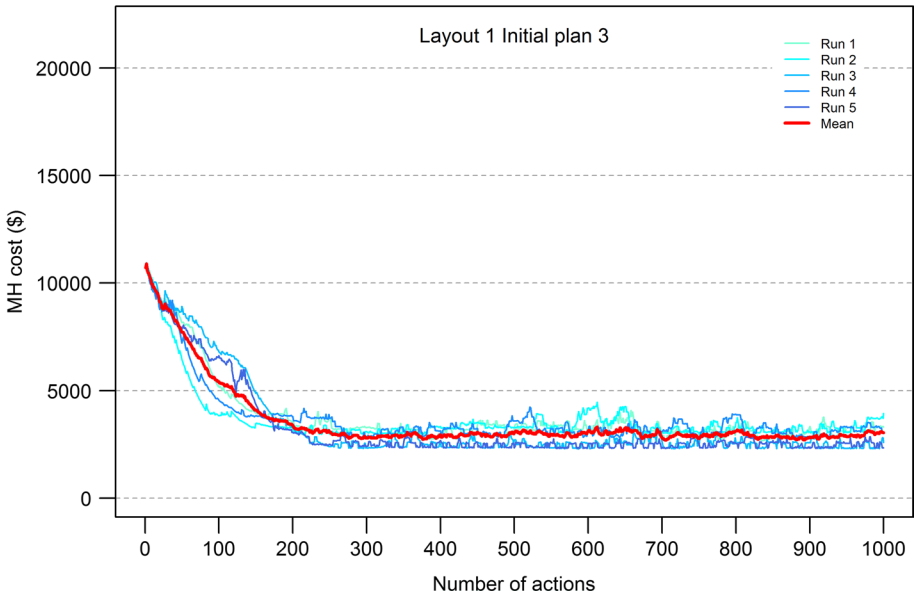

Fig. 13 MH cost improvement for initial plan 3 on factory layout 1

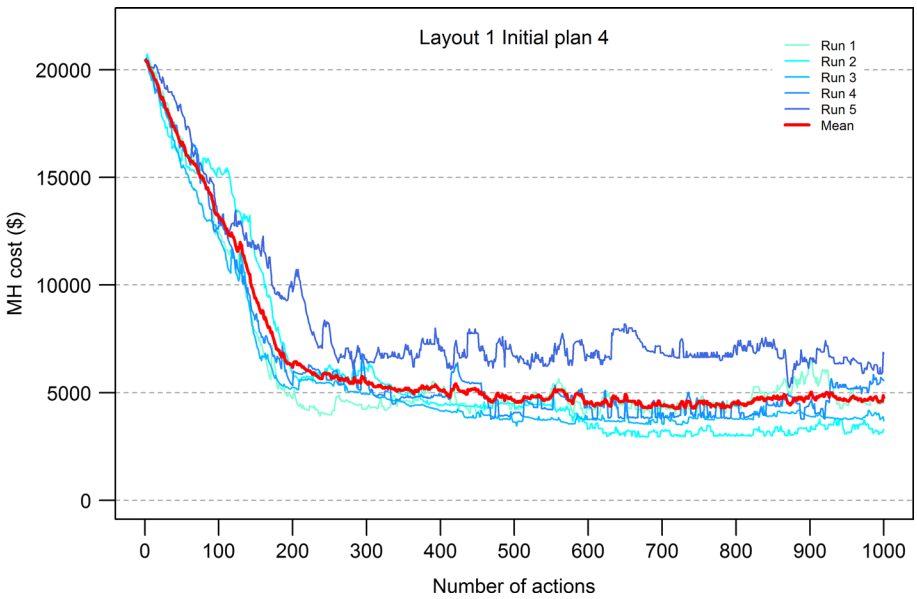

Fig. 14 MH cost improvement for initial plan 4 on factory layout 1

Alabama. However, the cost values should be treated as relative, not absolute, i.e., we do not assert that the current implementation accurately estimates actual material handling costs, only that it can reduce those costs.

Figures 11, 12, 13, 14, 15 and 16 show the results of the experiments for the six initial plans; there is one figure for each of the initial plans. In each figure, the *x* axis represents the number of plan change actions taken and the *y* axis represents the MH cost of a plan after each action. The RL process was run five times for each of the six initial plans. The narrow blue lines show the MH costs for the five individual runs for each initial plan and the wide red line shows the mean MH cost of those five runs.

As the figures show, the majority of the learning occurred within the first 250 actions, after which the cost of the MH plan tended to stabilize. Although the cost stabilized early, and only the first 1000 actions are shown for each initial plan, in total the algorithm was run for 3000 actions so that the learning agent had ample opportunity to explore and improve the plan. The slower stabilization observed for initial plan 5 suggests that the frequent visits to the workstation in that initial plan slowed the rate at which the plan could be improved.

Table 5 shows the final RL action values of all the actions for each state and for each initial plan. The action values were calculated using Bellman updates after each action selection as shown in Fig. 9 (line 15). A lower action value

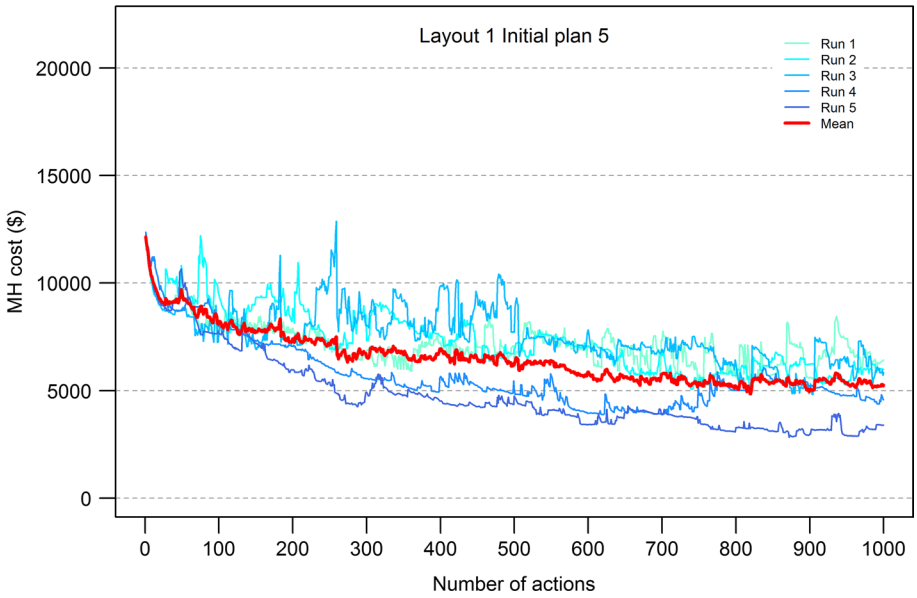

Fig. 15 MH cost improvement for initial plan 5 on factory layout 1

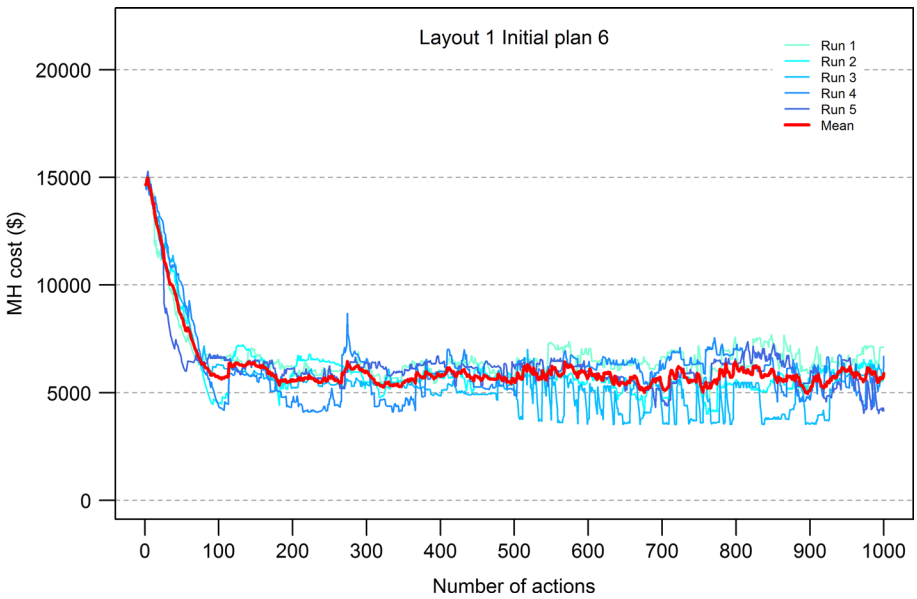

Fig. 16 MH cost improvement for initial plan 6 on factory layout 1

for a particular action in a particular state indicates that selecting that action in that state could lead to less desirable rewards. These values are the average of the five runs of 3000 iterations of each initial plan. The action values indicate that in the Penalty state an increase in the number of parts and the number of trips is appropriate whereas a decrease in the number of parts and trips is not appropriate, and vice-versa when the agent is in the No penalty state. The remaining actions generally lead to plan improvements (i.e., cost reductions) when there is no penalty, which the learning agent is expected to learn.

Table 6 shows the mean MH cost reduction for factory layout 1 for each initial plan, averaged over 5 runs of 3000 actions each. As can be seen by comparing initial cost to final cost, the RL process produced a significant reduction in the total MH cost for every initial plan.

For both factory layouts, the RL MH plan generating algorithm and the abstract discrete event simulation of the factory serving as the reward function were implemented in C++ using the Visual Studio 2017 framework on a Dell Core i5 computer. Each run of 3000 actions required approximately 5 min to execute on that computer. The Floyd-Warshall all-pairs shortest-paths algorithm [46] was used to pre-calculate the distances between the places in the layouts before learning began.

Table 5 Final action values of the RL actions for the six initial plans

Action	Initial plan 1			Initial plan 2		
	State 0	State 1	State 2	State 0	State 1	State 2
IncreaseTheNumberOfTrips	−17.8	−18.3	24.0	3.5	−3.3	30.2
DecreaseTheNumberOfTrips	16.6	−1.7	−25.9	26.5	−0.5	−55.1
IncreaseTheNumberOfPartsDelivered	−45.1	−14.2	53.4	−62.3	−5.6	57.9
DecreaseTheNumberOfPartsDelivered	45.1	11.1	−65.0	54.4	8.6	−69.4
ChangeNumberAndOrTypeOfEquipment	42.0	30.4	−159.9	105.9	1.5	−258.4
MergeWithTheRouteWithSameWorkstation	26.6	5.4	1.4	13.3	0.0	69.4
MergeWithNeighborWorkstation	39.1	20.2	35.3	40.9	3.9	73.5
MergeWithTheRouteWithInTheWalk	20.9	−1.1	18.8	21.5	7.9	31.3
MergeWithTheRouteArbitrarily	27.3	13.4	21.2	47.0	4.6	49.4
SplitTheRouteArbitrarily	−257.5	−201.5	−170.3	−312.4	−3.3	−230.6
Action	Initial plan 3			Initial plan 4		
	State 0	State 1	State 2	State 0	State 1	State 2
IncreaseTheNumberOfTrips	−30.9	−5.8	8.5	−28.8	−0.6	1.3
DecreaseTheNumberOfTrips	5.5	−2.2	−23.7	−0.6	0.2	−29.2
IncreaseTheNumberOfPartsDelivered	−40.8	−8.5	25.1	−34.4	3.1	29.2
DecreaseTheNumberOfPartsDelivered	32.5	5.0	−30.2	30.5	5.2	−37.1
ChangeNumberAndOrTypeOfEquipment	52.5	9.9	−112.3	39.8	29.2	−150.6
MergeWithTheRouteWithSameWorkstation	38.5	4.0	2.4	15.9	1.4	19.1
MergeWithNeighborWorkstation	74.5	6.0	25.5	32.3	17.1	43.7
MergeWithTheRouteWithInTheWalk	20.4	2.7	10.3	19.5	3.8	14.1
MergeWithTheRouteArbitrarily	45.5	10.4	27.0	23.1	11.7	7.9
SplitTheRouteArbitrarily	−280.9	−209.9	−158.2	−143.1	−31.8	−178.5
Action	Initial plan 5			Initial plan 6		
	State 0	State 1	State 2	State 0	State 1	State 2
IncreaseTheNumberOfTrips	−18.6	−0.7	46.1	−17.0	−20.0	−14.2
DecreaseTheNumberOfTrips	56.8	6.8	−79.8	49.6	8.3	−89.7
IncreaseTheNumberOfPartsDelivered	−80.3	1.1	65.5	−61.1	−6.0	44.2
DecreaseTheNumberOfPartsDelivered	63.1	−3.1	−61.3	55.8	6.3	−62.9
ChangeNumberAndOrTypeOfEquipment	114.7	5.3	−152.6	125.5	10.4	−251.4
MergeWithTheRouteWithSameWorkstation	43.1	−0.5	49.8	57.2	2.1	40.8
MergeWithNeighborWorkstation	77.7	12.5	64.6	71.4	−3.5	77.2
MergeWithTheRouteWithInTheWalk	16.2	−7.2	−4.6	40.9	1.5	16.4
MergeWithTheRouteArbitrarily	96.2	3.3	42.1	33.2	12.8	45.5
SplitTheRouteArbitrarily	−339.9	−18.9	−312.9	−358.8	−160.4	−292.5

Table 6 Reduction in MH cost produced by the RL process for factory layout 1

Initial plan	Initial MH cost	Final MH cost	Cost reduction (%)
1	10,208.7	1526.7	85.04
2	19,824.6	1893.8	90.44
3	7961.9	1689.3	78.78
4	18,848.5	1500.3	92.04
5	9990.7	1982.4	80.15
6	15,433.3	1631.5	89.42

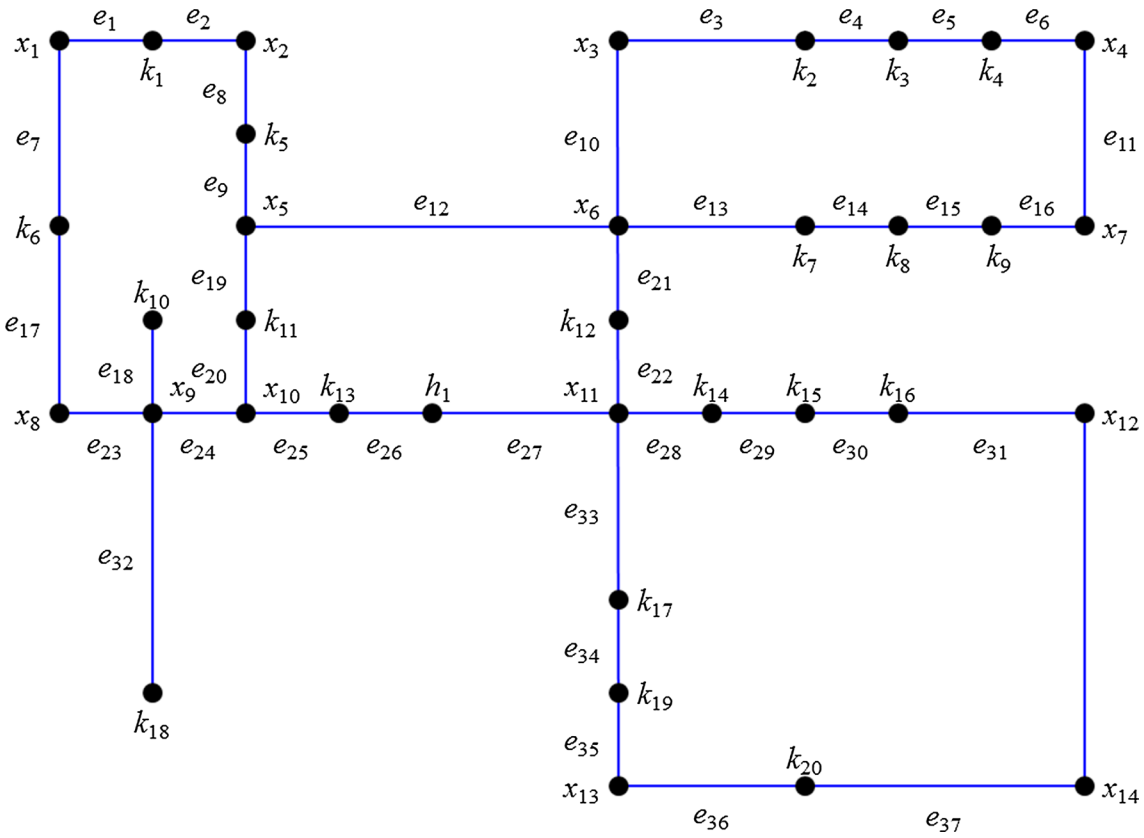

Fig. 17 Factory layout 2

6 Experiments, results, and analysis for factory layout 2

This section reports and analyzes the experimental results for the second factory layout. Any details of the factory, the material handling equipment, the initial material handling plans, or of the experimentation not explicitly reported for factory layout 2 should be assumed to be the same as for factory layout 1.

Factory layout 2 is approximately twice as large as factory layout 1 and has more complex connectivity, and thus presented a greater challenge to the RL algorithm. As shown in Fig. 17, factory layout 2 consisted of 20 workstations (k_1, k_2, k_3, …, k_{20}), 14 intersections (x_1, x_2, x_3, …, x_{14}), and one warehouse (h_1).

The 20 workstations consume a total of 20 different types of parts. Table 7 shows the consumption rate of each part type by each workstation and the pieces of equipment that can be used to transport parts of that part type.

The RL MH plan improvement process was tested for factory layout 2 with six different initial MH plans; those initial plans have the same descriptions as for factory layout 1 (see Table 3). The weights and costs of the reward response variables were also the same as for factory layout 1 (see Table 4).

Figures 18, 19, 20, 21, 22 and 23 show the results of the experiments for the six initial plans; there is one figure for each of the initial plans. As before, in each figure, the *x* axis represents the number of plan change actions taken and the *y* axis represents the MH cost of a plan after each action. The RL process was run five times for each of the six initial plans. The narrow blue lines show the MH costs for the five individual runs for each initial plan and the wide red line shows the mean MH cost of those five runs.

As the figures show, for the more complex factory layout the learning process required more actions. The majority of the learning occurred within the first 500 actions, after which the cost of the MH plan tended to stabilize. Although the cost stabilized early, and only the first 1000 actions are shown for each initial plan, in total the algorithm was run for 3000 actions for each initial plan.

Table 7 Workstations, consumption rate, and allowable equipment for each part type for factory layout 2

Part type	Workstation name	Consumption per hour	Equipment that can transport part type
it_1	k_1	98	Tugger
it_2	k_2	45	Forklift
it_3	k_3	59	Human puller
it_4	k_4	23	Tugger or Forklift
it_5	k_5	15	Tugger, Human puller
it_6	k_6	33	Tugger
it_7	k_7	70	Forklift
it_8	k_8	69	Human puller
it_9	k_9	24	Tugger, Forklift
it_{10}	k_{10}	81	Tugger, Human puller
it_{11}	k_{11}	60	Tugger
it_{12}	k_{12}	40	Forklift
it_{13}	k_{13}	50	Human puller
it_{14}	k_{14}	20	Tugger, Forklift
it_{15}	k_{15}	80	Tugger, Human puller
it_{16}	k_{16}	88	Tugger
it_{17}	k_{17}	54	Forklift
it_{18}	k_{18}	65	Human puller
it_{19}	k_{19}	73	Tugger, Forklift
it_{20}	k_{20}	94	Tugger, Human puller
it_3	k_2	38	Human puller
it_{16}	k_{11}	82	Tugger
it_{10}	k_{14}	90	Tugger, Human puller
it_{19}	k_6	44	Tugger, Forklift
it_2	k_{15}	28	Forklift
it_5	k_{18}	58	Tugger, Human puller
it_8	k_5	61	Human puller
it_{12}	k_{20}	31	Forklift
it_{14}	k_4	95	Tugger, Forklift
it_1	k_8	68	Tugger

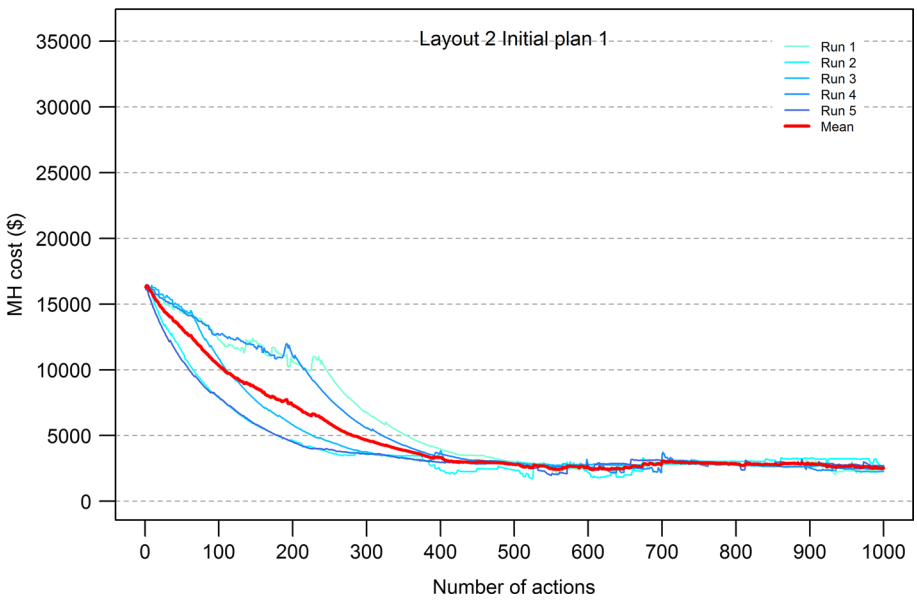

Fig. 18 MH cost improvement for initial plan 1 on factory layout 2

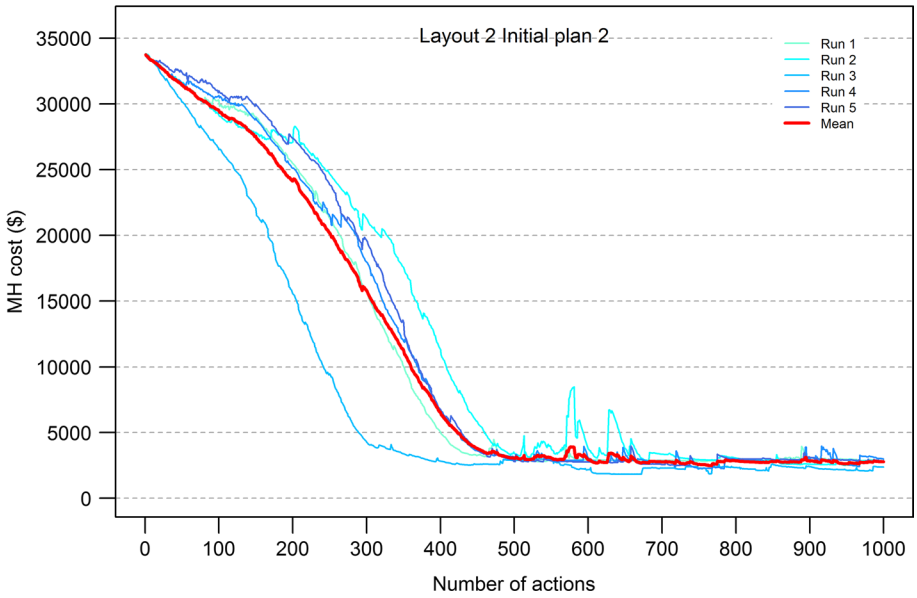

Fig. 19 MH cost improvement for initial plan 2 on factory layout 2

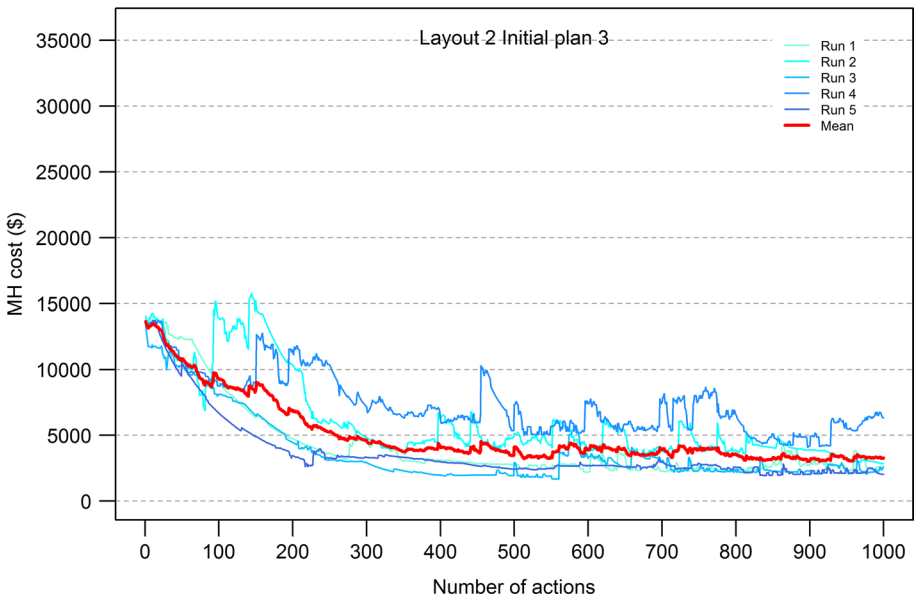

Fig. 20 MH cost improvement for initial plan 3 on factory layout 2

Table 8 shows the mean MH cost reduction for factory layout 2 for each initial plan, averaged over 5 runs of 3000 actions each. As can be seen by comparing initial cost to final cost, the RL process produced a significant reduction in the total MH cost for every initial plan.

7 Conclusions

All six of the research questions listed in Sect. 1 have been addressed. In particular, this work produced these results:

- A set of useful MH plan change operators was defined. This was challenging because the actions were intended to simultaneously improve, or at least not degrade, multiple reward response variables. Similar operators may enable RL to be applied to complex manufacturing environments with a large number of variables and where limited test data sets are available to train the system.
- A potentially very large state space was simplified using information from the reward calculation. Often, RL applications for problems with many states use neural networks or other non-linear function approximators to identify states

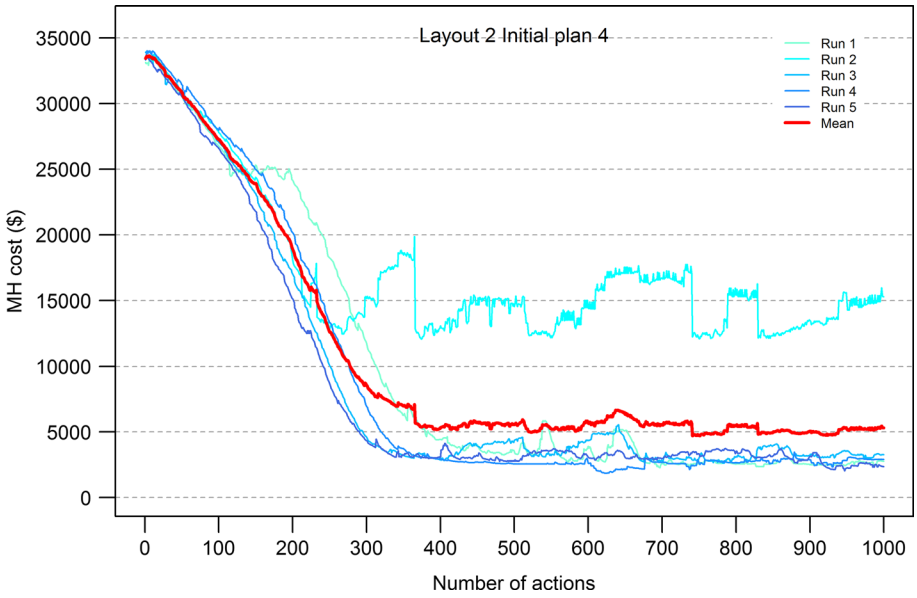

Fig. 21 MH cost improvement for initial plan 4 on factory layout 2

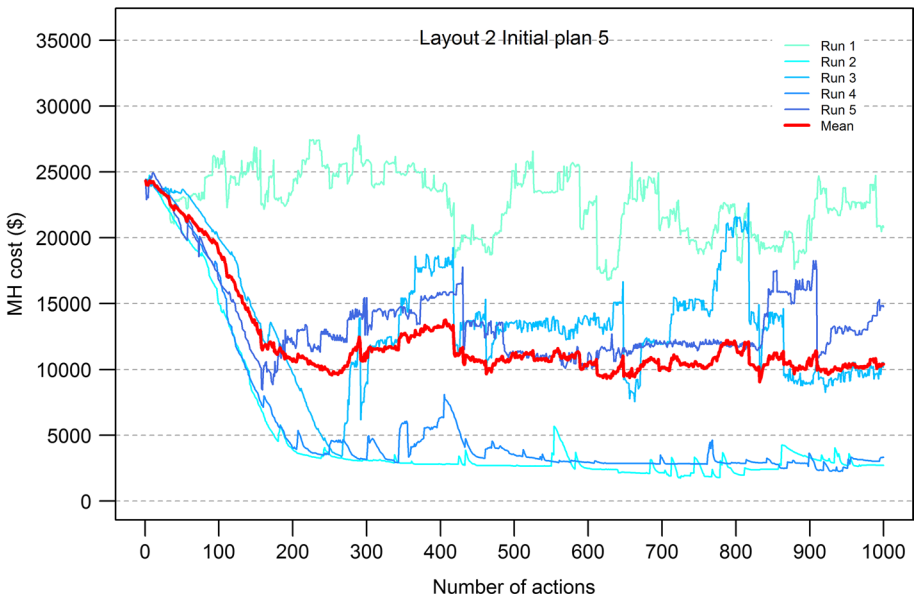

Fig. 22 MH cost improvement for initial plan 5 on factory layout 2

based on carefully hand-selected features. However, such approximators typically require large amounts of training data to learn the correct weights. The approach used here, deriving states from the reward calculation, can reduce the state space without requiring training data.

- Similarly, a potentially large set of RL actions was simplified using separate action selection and action parameterization stages; the latter stage exploited the supplemental information produced by the reward calculation.
- An important criterion for state transition, the mean parts threshold value, was learned concurrently with learning the action selection policy. This possibility was previously identified as future work in [16].
- During the experimental runs a sequence of increasingly efficient plans were produced from each initial plan, thereby enabling an MH manager to select from among those plans the one that is best suited to their requirements.
- As shown in Tables 6 and 8, the RL process consistently reduced MH costs across a range of different initial plans; the minimum cost reduction for both layouts and all 12 initial plans was approximately 54.7% and the mean cost reduction was approximately 82.8%. For most real-world factories, a reduction in MH costs of even a tenth of those magnitudes would be very significant.

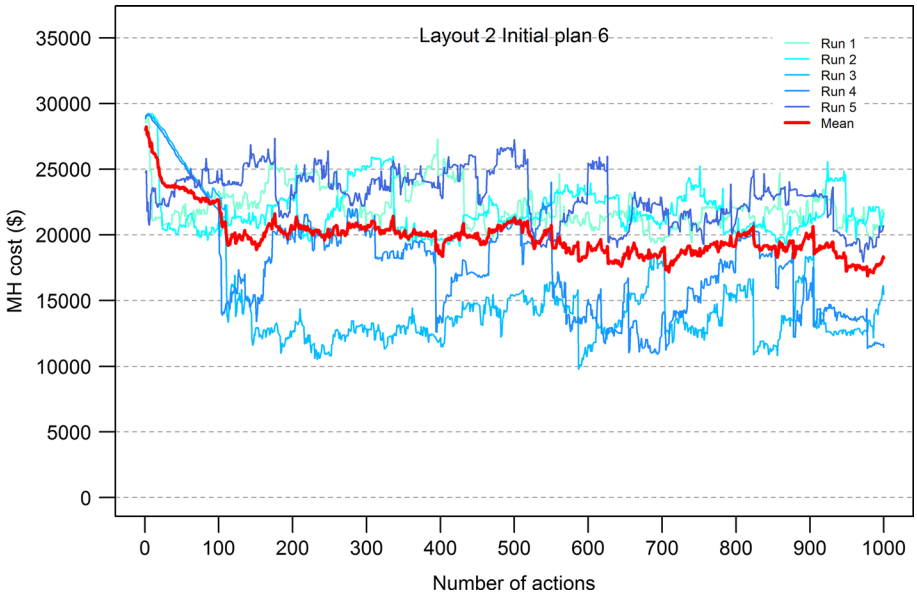

Fig. 23 MH cost improvement for initial plan 6 on factory layout 2

Table 8 Reduction in MH cost produced by the RL process for factory layout 2

Initial plan	Initial MH cost	Final MH cost	Cost reduction (%)
1	16,295.8	2777.1	82.96
2	33,735.2	2203.3	93.47
3	13,648.3	2404.2	82.38
4	33,443.1	2699.5	91.93
5	24,328.7	6713.7	72.40
6	28,014.9	12,700.9	54.66

The experimental results demonstrate that a conventional RL method (temporal learning) was able to learn a policy that could effectively reduce the cost of material handling plans. In this research, the action values were estimated using a one-step backup operation, i.e., the action values of a given state were used to estimate the action values of the previous state using the simulation-calculated immediate reward. Actions which reduced the reward, i.e., the cost of the material handling plan, were selected greedily in each state by following the standard ε-greedy policy. As is typical in RL approaches, the learning agent repetitively visited each state, using the estimated and recorded action values to learn an effective policy [16]. A simulation-calculated cost was used as an immediate reward by the RL algorithm, and some of the additional non-cost information produced by the reward calculation was used by problem-specific heuristics to parameterize the selected action; this heuristic action parameterization is unlike conventional RL algorithms. One of the contributions of this research is this demonstration that for complex problems such as material handling planning, additional reward calculation information, which we call reward observation, can be useful along with recording the action values of each state for generating and applying an efficient policy.

Overall, this work demonstrates that RL can be successfully applied to a realistically complex MH problem with a multi-objective reward function. A key feature of this research is its potential to lead to practical application, which distinguishes it from much of the prior work on applying RL to MH. The inputs to the RL process included factory layout, equipment capacities, part types, workstation consumption rates and buffer capacities, and plan duration. These are also common inputs used in practical MH modeling applications. Because the cost data and factory layouts used in the experiments were based on a real factory and the discrete event simulation used to evaluate the MH plans is consistent with the discrete event simulations typically used in the manufacturing industry, it is reasonable to expect that an RL algorithm using multiple reward response variables and a two-stage action selection process can be potentially applied to real world MH plan generation in response to changes in demand.

8 Future work

These results suggest several opportunities for future work.

- Increasing the number of RL states (without going too far) may allow generating even better plans. Realistic MH is a complex problem with many input variables (see Sect. 3.4). Defining states in a manner that distinguishes situations where different actions are appropriate, without producing an overly large state space, is challenging. Additional state aggregation schemes could be explored.
- Because MH is a multi-objective optimization problem, action selection methods other than ε-greedy, such as priority-based exploration, may offer better performance.
- Different methods of scalarization of the multiple reward variables could be tested.
- The uncertainties present in real-world MH, such as equipment breakdown or traffic congestion, are not represented in the current implementation. Adding these uncertainties could increase the potential relevance of this work.
- Additional response variables, such as the human resources used to perform MH, could be included in the reward function.
- This work considered only centralized and pre-planned MH operations. Alternative MH processes, including distributed and on-demand deliveries, could be studied.
- Experiments to determine how to leverage the improved MH plans to benefit other interconnected processes in the manufacturing facility could be valuable.
- Improvements to MH plans obtained using RL could be compared to improvements produced using other known methods, including the methods mentioned in Sect. 2.5 (evolutionary algorithms, hill climbing, and simulated annealing).
- After further improving and generalized the RL process for MH planning, processes for applying the process to real-world manufacturing facilities could be developed.

Acknowledgements No funding was provided for the specific research described in this article or for preparing this article. For a portion of the period of time spent performing the research, Govindaiah was supported (tuition and stipend) by a grant from Steelcase, Inc., Grand Rapids Michigan USA. Steelcase did not influence, review, approve, or endorse the results reported in this article.

Authors' contributions Both authors made substantial contributions to this article and to the research it describes. SG developed the formal representation of material handling plans, designed the algorithms, implemented the algorithms as software, and executed the experiments. She also wrote portions of the article and produced three of the figures in the article. MDP defined the overall project, designed the experiments, and assisted with the development of the formal representation of material handling plans and the design of the algorithms. He also wrote portions of the article, edited and revised the entire article, and produced twenty of the figures in the article. Finally, he revised the article in response to the reviewers' comments. Both authors read and approved the final manuscript.

Funding No funding was provided for the specific research described in this article. For a portion of the period of time spent performing this research, Govindaiah was supported (tuition and stipend) by a grant from Steelcase, Inc., Grand Rapids Michigan USA. However, Steelcase did not review, approve, or endorse the results reported in this article.

Data availability The datasets generated during the current study are available from the corresponding author on reasonable request.

Code availability The software code generated during the current study is available from the corresponding author on reasonable request.

Declarations

Competing interests The authors declare that there are no conflicts of interest or competing interests associated with this article or with the research it describes.

References

1. Sethi AK, Sethi SP. Flexibility in manufacturing: a survey. Int J Flex Manuf Syst. 1990;2(4):289–328.
2. Tiwari A, Tiwari A, Samuel C. Supply chain flexibility: a comprehensive review. Manag Res Rev. 2015;38(7):767–92.
3. Pérez Pérez M, Serrano Bedia A, López FM. Int J Prod Res. 2016;54(10):3133–48.
4. Pujawan IN, Smart AU. Factors affecting schedule instability in manufacturing companies. Int J Prod Res. 2012;50(8):2252–66. https://doi.org/10.1080/00207543.2011.575095.
5. Chen C, Xia B, Zhou BH, Xi L. A reinforcement learning based approach for a multiple-load carrier scheduling problem. J Intell Manuf. 2015;26(6):1233–45.
6. Govindaiah S. Reinforcement learning applied to manufacturing material handling. Huntsville: Ph.D. Dissertation, University of Alabama in Huntsville; 2019.
7. Govindaiah S, Petty MD. A discrete event simulation-based multi-objective reinforcement learning reward function for optimizing manufacturing material handling. Proceedings of the 2019 Simulation Innovation Workshop. Orlando FL, February 11–15 2019.
8. Govindaiah S, Petty MD. Applying reinforcement learning to plan manufacturing material handling, part 1: Background and formal problem specification. Proceedings of the 2019 ACM Southeast Conference. Kennesaw GA, April 18–20 2019, 168–171; https://doi.org/10.1145/3299815.3314451.
9. Govindaiah S, Petty MD. Applying reinforcement learning to plan manufacturing material handling, part 2: Experimentation and results. Proceedings of the 2019 ACM Southeast Conference. Kennesaw GA, April 18–20 2019, 16–23; https://doi.org/10.1145/3299815.3314427.
10. Coyle JJ. Management of Business Logistics. Mason: South-Western; 1992.
11. White JA. Material handling research: Needs and opportunities. In: Material Handling '90, Progress in Material Handling and Logistics, Vol 2. Berlin: Springer, 1991; https://doi.org/10.1007/978-3-642-84356-3_1.
12. Stephens M, Meyers F. Manufacturing facilities design and material handling. West Lafayette: Purdue University Press; 2013.
13. Jain A, Jain P, Chan F, Singh S. A review on manufacturing flexibility. Int J Prod Res. 2013;51(19):5946–70.
14. Bishop C. Pattern recognition and machine learning. New York: Springer; 2006.
15. Russell S, Norvig P. Artificial intelligence: a modern approach. 4th ed. Hoboken: Pearson Education Limited; 2021.
16. Sutton RS, Barto AG. Reinforcement learning: an introduction. 2nd ed. Cambridge: MIT Press; 2018.
17. Alpaydin E. Introduction to machine learning. 4th ed. Cambridge: The MIT Press; 2020.
18. Barbosa SE, Petty MD. Exploiting spatio-temporal patterns using partial-state reinforcement learning in a synthetically-augmented environment. Prog Artif Intell. 2015;3(2):55–71.
19. Bland JA, Petty MD, Whitaker TS, Maxwell KP, Cantrell WA. Machine learning cyberattack and defense strategies. Comput Secur. 2020. https://doi.org/10.1016/j.cose.2020.101738.
20. Bellman R. Dynamic programming. Princeton: Princeton University Press; 1957.
21. Aydin ME, Öztemel E. Dynamic job-shop scheduling using reinforcement learning agents. Robot Auton Syst. 2000;33(2–3):169–78.
22. Wang YC, Usher JM. A reinforcement learning approach for developing routing policies in multi-agent production scheduling. The Int J Adv Manuf Technol. 2007;33(3–4):323–33.
23. Paternina-Arboleda CD, Das TK. A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. Simul Model Pract Theory. 2005;13(5):389–406.
24. Sui Z, Gosavi A, Lin L. A reinforcement learning approach for inventory replenishment in vendor-managed inventory systems with consignment inventory. Eng Manag J. 2010;22(4):44–53.
25. Wang YC, Usher JM. Application of reinforcement learning for agent-based production scheduling. Eng Appl Artif Intell. 2005;18(1):73–82.
26. Gabel T, Riedmiller M. Adaptive reactive job-shop scheduling with reinforcement learning agents. 2008. http://ml.informatik.uni-freiburg.de/former/_media/publications/gr07.pdf. Accessed 10 May 2021.
27. Aissani N, Beldjilali B, Trentesaux D. Dynamic scheduling of maintenance tasks in the petroleum industry: a reinforcement approach. Eng Appl Artif Intell. 2009;22(7):1089–103.
28. Dou J, Chen C, Yang P. Genetic scheduling and reinforcement learning in multirobot systems for intelligent warehouses. Math Probl Eng. 2015. https://doi.org/10.1155/2015/597956.
29. Pontrandolfo P, Gosavi A, Okogbaa OG, Das TK. Global supply chain management: a reinforcement learning approach. Int J Prod Res. 2002;40(6):1299–317.
30. Watkins CJ, Dayan P. Q-learning. Mach Learn. 1992;8(3–4):279–92.
31. Hwang CL, Masud AM. Multiple objective decision making—methods and applications. Berlin: Springer; 1979.
32. Vamplew P, Dazeley R, Berry A, Issabekov R, Dekker E. Empirical evaluation methods for multiobjective reinforcement learning algorithms. Mach Learn. 2011;84(1–2):51–80.
33. Natarajan S, Tadepalli, P. Dynamic preferences in multi-criteria reinforcement learning. Proceedings of the 22nd international Conference on Machine Learning, 601–608. Bonn: ACM; 2005.
34. Van Moffaert K, Drugan MM, Nowé A. Scalarized multi-objective reinforcement learning: Novel design techniques. IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. Singapore: IEEE Press; 2013, 191–199.
35. Gábor Z, Kalmár Z, Szepesvári C. Multi-criteria reinforcement learning. Proceedings of the 15th International Conference on Machine Learning. San Francisco: ACM; 1998, 197–205.
36. Van Moffaert K, Drugan MM, Nowé A. Hypervolume-based multi-objective reinforcement learning. Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization. Berlin: Springer; 2013, 352–366.
37. Barrett L, Narayanan S. Learning all optimal policies with multiple criteria. Proceedings of the 25th International Conference on Machine Learning. Helsinki: ACM; 2008, 41–47.
38. Van Moffaert K, Nowé A. Multi-objective reinforcement learning using sets of pareto dominating policies. J Mach Learn Res. 2014;15(1):3483–512.

39. Mossalam H, Assael YM, Roijers DM, Whiteson, S. Multi-objective deep reinforcement learning. 2016. https://arxiv.org/ftp/arxiv/papers/1803/1803.02965.pdf. Accessed 10 May 2021.
40. Mocanu E, Mocanu DC, Nguyen PH, Liotta A, Webber ME, Gibescu M, Slootweg JG. On-line building energy optimization using deep reinforcement learning. IEEE Trans Smart Grid. 2018;10(4):3698–708.
41. Mannor S, Shimkin N. A geometric approach to multi-criterion reinforcement learning. J Mach Learn Res. 2004;5:325–60.
42. Zhang W, Dietterich TG. A reinforcement learning approach to job-shop scheduling. Proceedings of the 14th international joint conference on Artificial intelligence. Montreal: Morgan Kaufmann; 1995, 1114–1120.
43. Zhang W, Dietterich TG. Solving combinatorial optimization tasks by reinforcement learning: a general methodology applied to resource-constrained scheduling. J Artif Intell Res. 2000;1:1–38.
44. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of state calculations by fast computing machines. J Chem Phys. 1953;21(6):1087–92.
45. Plehn C, Stein F, Reinhart G. Modeling factory systems using graphs – Ontology-based design of a domain specific modeling method. Proceedings of the 20th International Conference on Engineering Design, Volume 4: Design for X, Design to X. Milan: The Design Society; 2015, 163–172.
46. Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 3rd ed. Cambridge: The MIT Press; 2009.
47. Rooeinfar R, Raissi S, Ghezavati VR. Stochastic flexible flow shop scheduling problem with limited buffers and fixed interval preventive maintenance: a hybrid approach of simulation and metaheuristic algorithms. Trans Soc Model Simul Int. 2018. https://doi.org/10.1177/0037549718809542.
48. Banks J, Carson JS, Nelson BL, Nicol DM. Discrete event system simulation. 5th ed. Upper Saddle River: Prentice Hall; 2010.
49. Fishburn PC. Additive utilities with incomplete product sets: application to priorities and assignments. Oper Res. 1967;15(3):537–42.