Reinforcement Learning for Layout Planning – Modelling the Layout Problem as MDP

Hendrik Unger, Frank Börner

Reinforcement Learning for Layout Planning – Modelling the Layout Problem as MDP

Hendrik Unger[1][0000-0002-3744-9886] and Frank Börner[1]

[1] Technische Universität Chemnitz, Erfenschlager Straße 73
09125 Chemnitz, Germany
hendrik.unger@mb.tu-chemnitz.de

Abstract. The layout problem has been a focus point of research in factory planning for over six decades. Several newly emerging techniques for example genetic algorithms have been applied to the problem to generate better solutions closer to practical application. Nevertheless, solving the layout problem without considerable simplification of the base problem still presents a challenge. This publication shows how to model the layout problem in the framework of Markov decision processes (MDP) to apply reinforcement learning as a novel approach for generating layouts. Reinforcement learning (RF) has previously not been applied to the layout problem to the best knowledge of the author. Research in other fields of study shows the enormous potential of RF and the capability to reach superhuman performance in a variety of tasks. Although RF may not provide a better solution for finding the global optimum in the layout problem than genetic algorithms or dynamic programming, we hope to be able to include more constrains that matter for real world planning applications while keeping the calculation time feasibility short for practical application.

Keywords: Layout Problem, Reinforcement Learning, Artificial Intelligence.

1 Motivation

A challenge in factory planning, which must be solved in every planning job, is the layout problem. This is the task of arranging the elements and subsystems contained in a production system, such as production machines and transport routes, optimally in a defined solution space within a coordinate system according to certain target criteria (usually based on the material flow) resulting in a definition of the target function. Classically, ideal layout planning is performed first, without existing external constrains for the solution space in the coordinate system. Then, the solution generated in this way is adapted to the real layout planning to the constraints that exist in the real project [1, p. 335], [2, p. 13ff], [3, p. 317f].

Central factors influencing the complexity of the layout problem are the number and the shape of the elements to be arranged. In the simplest case, only rectangular (2D planning) or cuboid-shaped (3D planning) elements are placed, which leads to a faster solution, but one that is further away from practical planning reality. Considering real machine floor plans where logistics and working areas need to be placed,

much more complex geometries emerge in practical application cases. Thus, solutions created with previous methods usually must be reworked by a planner [4, p. 427]. Another factor that complicates the solution of the layout problem is the optimality criterion. Classically, the material flow between the arranged elements is determined to achieve a global minimum [1, p. 323], [5], [6]. An efficient material flow correlates strongly with an efficient factory and accordingly lower operating costs of the factory life cycle [7, p. 200], [8, p. 348]. Nevertheless, there are numerous other influencing factors that must be considered in real planning. These include information flows, energy flows, media availability (compressed air, electricity, process gases, ...), the load-bearing capacity of the floor slab, the necessary illumination of the workplaces, as well as requirements for air purity, vibration behaviour, temperature stability and the influence of other system elements based on these factors. While optimization, considering several of these criteria usually leads to a conflict of objectives due to the mutually contradictory target criteria. Even using simple object contours and only the material flow as optimization criteria, the complexity of the problem and its solution space increases exponentially to the point where no optimal solution can be found in a reasonable time even with current computational technology - the layout problem is classified as NP-hard. In particular, the objective of arranging several elements in the order of magnitude of a real planning problem leads to the fact that many of the known solution algorithms and heuristics are no longer usefully applicable, because no solution can be found, or the computational effort is no longer economically justifiable. To be able to find usable solutions at all, it is common to reduce the existing complexity by reducing restrictions in the described target criteria. Thus, known important influencing variables are either disregarded or evaluated in a downstream step only. A local optimization takes place in many other procedures as well, so the quality of the final solution depends on the starting point of the algorithm.

Inconclusion, it can be stated that the computation time contributes a huge importance in the solution of the layout problem in practice, so many of the present algorithms may not be used in an economic application. In addition, in most cases not all relevant influencing factors of the problem are included in the optimization. Finally, a large part of the planning process is performed by humans based on empirical knowledge and domain-specific expertise. The simultaneous consideration of several objective criteria is limited by the cognitive abilities of the planner.

Our research goal is to apply reinforcement learning to the layout problem and evaluate its performance against other state of the art solution algorithms. This contribution shows our approach to formulating the layout problem in a way suitable to be solved by reinforcement learning algorithms.

2 Background

AlphaGo, AlphaGo Zero, and AlphaZero [9]–[11] achieved performances above that of reigning world champions in the extremely complex game of Go. AlphaZero was transferred to other complex games such as shogi and chess and was able to outperform humans there as well. Another machine learning success is represented by

Agent57 [12], an artificial intelligence that solves all 57 different games in the Atari benchmark [13]. These examples highlight the potential of reinforcement learning technology. In another domain, a problem closely related to the layout problem – the floor planning problem on microchips - has been successfully solved by RF [14], which increases the motivation for applying similar methods to the layout problem as well.

The underlying mathematical theory used is the Markov decision processes (MDP) [15]. Depending on how well the process can be modelled there are different possible approaches for finding a solution. If all parameters are well known and well modelled, an optimal solution can be determined using dynamic programming and optimization. Obtaining and solving a complete model of the layout problem is not always possible or economically feasible, as described in the previous section. RF is applicable without the full model of the MDP. The basic principle is shown in Figure 1.

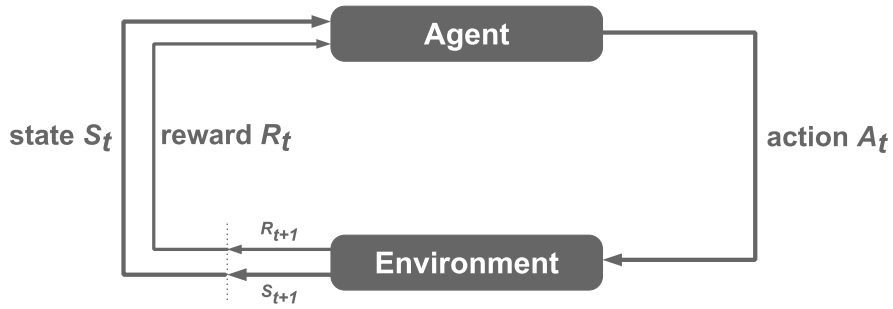

Fig. 1. Reinforcement Learning Cycle [16]

A learning agent interacts with an environment by selecting actions. The environment simulates the problem to be solved and provides feedback on the quality of the action according to the reward function as well as the new state of the environment. Based on this information, the agent selects a new action. The agent continuously learns from the interactions by exploring and tries to optimize its behavior in such a way that it receives the best possible reward. Considering the reward function there are several possible approaches for the design. The easiest solution would be to reward the action leading to the intended outcome of the problem with maximum reward and every other action with no or negative reward. Such an approach provides a particularly challenging problem to solve for the agent and would result in failure to learn anything for more complex environment due to the agent never achieving a positive reward. There are solutions to improve learning in such sparse reward environments in literature, e.g. curiosity [17] or attention [18]. The reward function can also supply additional smaller rewards for subgoals or behaviors that are considered well suited towards solving the bigger problem by the programmer. This results in easier learnable environments but also requires domain knowledge of the problem to be solved.

Defining sub rewards can push the agent into a predefined behavior, instead of creating a new and not previously known solution from scratch. Such a reward design needs to be carried out very carefully otherwise there is a risk of the agent not developing the desired behavior. Instead, a behavior coined “reward gaming” can emerge, which results for example in choosing an action leading to an easy achievable low reward instead of a (desired) more complex action leading to a higher reward.
To be able to solve the layout problem it must be modelled as an MDP. A possible solution how this can be achieved is described in the next section.

3 Modelling the Layout Problem as MDP

According to [19, p. 10ff], an MDP consists of states, actions, action model and a reward function. In addition, there is an agent that makes decisions and learns from their results. All elements that are not part of or directly influenceable by the agent are called environment. The boundary between agent and environment is not always firmly defined in the literature and can be determined in the context of modeling a problem [16, p. 37ff]. The solution approach RF, as a representative of model-free algorithms, only requires states S, possible actions A and a reward for the last action as input since the information about the model is collected and learned during interaction with the environment.

3.1 State

States in the MDP capture all relevant aspects of the environment and therefore describe the available solution space for the agent. The relevant influencing factors in practical layout planning applications must be reflected in the states. The quality of the final layout depends on the fulfilment of the requirements for every individual influence factor and is considered in the reward function. Since not all influence factors are relevant in every planning project, a mechanism is required to ignore irrelevant influences in the environment. Nevertheless, during the training process, the agent must learn how to deal with all potential influencing variables. In addition, the number of system elements to be arranged is not constant in the layout problem. From planning case to planning case, different numbers of workstations / machines must be placed in a layout. Almost all algorithms that can act as agents expect a constant size of the state space as input. In terms of a practical solution, it is not possible to train a variant of the agent for different numbers of system elements.
To meet the requirements described above, the states of the environment are encoded as images or image stacks. Depending on the requirements, colored images (RBG), greyscale images or a combination of both can be used. In principle, colored images allow more information to be contained, but they also place higher demands on the available resources for executing the environment. The agent is not always able to use the additional information effectively [20]. To reduce memory consumption and speed up computation, it is advisable to encode as many environmental states as possible in greyscale images. The actual resolution of the images is defined as a hyperpa-

rameter of the environment. This parameter has a significant influence on the resource requirements and calculation time of both the agent and the environment and should be kept reasonably low. Many algorithms acting as an agent employ neural networks internally. The size of the state space often defines the number of used neurons. With a larger number of neurons, more complex relationships can be modelled [21], so there may also be a correlation between the potential learning ability of the agent and the resolution of the environmental state, which has to be explored in an subsequent publication.
An example of the described architecture of an environmental state stack is shown in Figure 2.

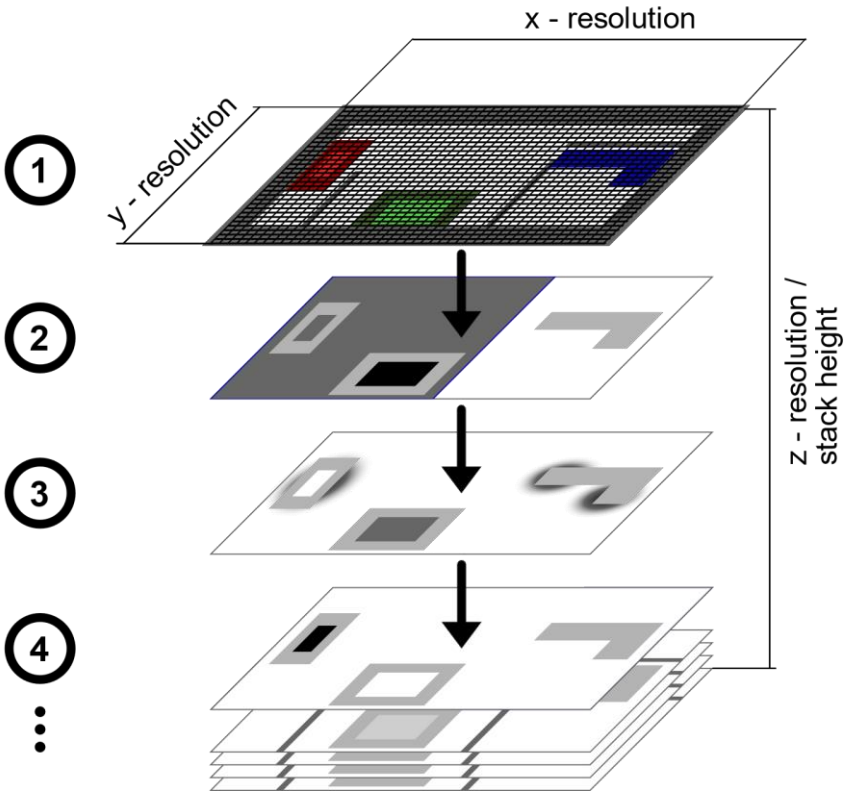

Fig. 2. Principle representation of an environmental state stack

While the resolution in x - and y - direction is defined by the hyperparameter mentioned above, stack height is derived from the selected influence factors of the layout problem. Each level of the environment state stack defines the need for a certain influence factor as well as the corresponding degree of fulfilment of the system elements that are going to be arranged. Greyscale images with 8-bit encoding result in a subdivision into 256 units, so that complex dependencies can be represented. The

darker an element is drawn, the higher the requirement or degree of fulfilment. The background / free space of the layout is also colored according to the requirement or degree of fulfilment in the space at this point. In Figure 2, Level 1 represents the basic layout, all interior and exterior walls and whether collisions occur between system elements and walls. Due to the complexity of this information, a colored representation was chosen. Level 2 encodes the foundation support load requirements of the individual system elements. The left part of the space has a reinforced floor slab, which is represented by darker color at this position in the greyscale image. Level 3 represents the emission of vibrations of the individual machines and their respective sensitivity to vibrations. These are arbitrary examples; the order of the influence factors can be chosen freely but must remain consistently after initial definition and training.
With the help of the described stacking system, an arbitrary but fixed set of influence factors can be represented with simultaneous flexibility regarding to the number of system elements in the layout problem. At the same time, decisions of the agent can be made more transparent by exporting the environment state stack as individual images, since each image encodes only one influence factor. All potentially occurring influence factors can be used to train the agent. In a deployment application, unused influence variables can be declared as "no requirements" by pure white images.

3.2 Actions

The action space represents the possibilities for the agent to influence the environment. The goal in the layout problem is the arrangement of system elements on plane. Accordingly, the agent must be enabled to change the x - and y - coordinates of each element. The problem described in the previous section – the requirement of many algorithms to obtain a fixed number of states as input variables - also applies to the output of the agent. Approaches for parametric action spaces exist in the literature [22], [23], but this imposes serious limitations on the algorithms that can be chosen for the agent. For this reason, the arrangement of system elements is modelled in a turn-based manner. In each interaction step, the agent can place one element in a continuous space in x - and y - direction. The active and the following element are represented in the state of the environment. Then, a turn then corresponds to the iteration over all system elements, that need to be arranged. However, the length of an episode can be defined differently:

One shot

The episode length corresponds to exactly one turn. Each element is only moved once by the agent. The advantages here are a simple structure, simple calculation of key figures for agent performance and low potential for reward gaming. On the other hand, the learning task at the beginning of the training is difficult for the agent. If a state with an occurring reward does not arise during the random exploration phase, then a training run may end without learning success.

Objective function change threshold

The end of the episode occurs when a defined objective function, or the reward function experiences a change below a specified threshold. This option, similar to the termination criterion in an optimization task, allows the agent to improve its initial solution and offers good potential for successful training. The risk for reward gaming is exceedingly high. A scenario in which the agent learns the termination threshold and then continually moves system elements just enough to obtain a positive reward each iteration is possible.

Fixed number of turns

This is a compromise between the two previous alternatives. The episode length is fixed, but multiple turns are allowed. It offers potential for optimizing the solution without creating too much risk for reward gaming. A fixed episode length simplifies the calculation of key figures of agent performance and the quantification of the training progress.

Termination by agent

The agent is given an additional binary action to choose from to end an episode. This approach can in principle be combined with any of the previous ones but requires a specific formulation of the reward function. To prevent infinite episodes, a punishment factor must be modelled that correlates with the episode length.

In the current state of research, the best approach for the action space cannot be determined yet and needs further experimentation.

4 Conclusion and further Research

In this paper we described our modelling approach for the layout problem tailored to the requirements of reinforcement learning as a solution algorithm. To check the plausibility and feasibility of the described model approach, a proof-of-concept implementation was created. In the process, the layout problem was simplified to central core aspects to achieve an executable state for a concept study more quickly. Only the material flow was used as an evaluation criterion for the quality of a layout and the reward. In addition, some constraints had to be implemented to obtain valid solutions. The most important aspect was the avoidance of collisions between the arranged objects and layout boundaries as well as within the objects themselves. The prototype already provides a simple interface for data transfer via Industry Foundation Classes (IFC) standard, result visualisation and the integration interface for state-of-the-art reinforcement learning algorithms. In the tests already conducted, a learning behaviour of the algorithms used so far, Proximal Policy Optimisation, [24] and Actor Critic using Kronecker-Factored Trust Region [25] could be observed. Depending on the formulation of the reward function and action space behaviours resembling other simple heuristics for graphically solving the layout problem like Schwerdtfegers's

heuristic [26, p. 149ff], could be observed after training the model for 20 – 40 CPU hours. This result shows that an easy formulation of the layout problem can in fact be solved by reinforcement learning.

The next steps for research include the incorporation of additional influence factors to the layout problem to achieve a complexity closer to a real-world planning task in factory planning. After transitioning the environment to an architecture that enables computation on clusters, the performance of additional reinforcement learning algorithms on this problem will be evaluated. Finally, the best performing solution will be tuned to maximum performance and benchmarked against other solution approaches for the layout problem.

References

[1] S. Wirth, M. Schenk, and E. Müller, *Fabrikplanung und Fabrikbetrieb*. 2014.

[2] VDI 5200 Blatt 1, "VDI 5200 Blatt 1." pp. 1–24, 2011.

[3] D. Arnold, H. Isermann, A. Kuhn, H. Tempelmeier, and K. Furmans, *Handbuch Logistik*. 2008.

[4] S. P. Singh and R. R. K. Sharma, "A review of different approaches to the facility layout problems," *Int. J. Adv. Manuf. Technol.*, vol. 30, no. 5–6, pp. 425–433, 2006, doi: 10.1007/s00170-005-0087-9.

[5] A. Kusiak and S. S. Heragu, "The facility layout problem," *Eur. J. Oper. Res.*, vol. 29, no. 3, pp. 229–251, 1987, doi: https://doi.org/10.1016/0377-2217(87)90238-4.

[6] Y. A. Bozer and R. D. Meller, "A reexamination of the distance-based facility layout problem," *IIE Trans. (Institute Ind. Eng.*, vol. 29, no. 7, pp. 549–560, 1997, doi: 10.1080/07408179708966365.

[7] U. Bracht, M. Dahlbeck, A. Fischer, and T. Krüger, "Combining simulation and optimization for extended double row facility layout problems in factory planning," 2018, doi: 10.1007/978-3-319-96271-9_3.

[8] S. Hawer, P. Ilmer, and G. Reinhart, "Klassifizierung unscharfer Planungsdaten in der Fabrikplanung," *ZWF Zeitschrift fuer Wirtschaftlichen Fabrikbetr.*, 2015, doi: 10.3139/104.111339.

[9] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, 2016, doi: 10.1038/nature16961.

[10] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, 2017, doi: 10.1038/nature24270.

[11] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science (80-.).*, 2018, doi: 10.1126/science.aar6404.

[12] A. P. Badia *et al.*, "Agent57: Outperforming the Atari Human Benchmark," Mar. 2020, Accessed: Oct. 30, 2020. [Online]. Available: https://arxiv.org/abs/2003.13350.

[13] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An evaluation platform for general agents," *J. Artif.*

Intell. Res., 2013, doi: 10.1613/jair.3912.

[14] A. Mirhoseini *et al.*, “Chip Placement with Deep Reinforcement Learning,” Apr. 2020, Accessed: Oct. 30, 2020. [Online]. Available: http://arxiv.org/abs/2004.10746.

[15] R. Bellman, *Dynamic programming*. Princeton: Princeton University Press, 1956.

[16] R. S. Sutton and A. Barto, *Reinforcement learning : an introduction*. 2018.

[17] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 6, pp. 4261–4270, 2017.

[18] A. Vaswani *et al.*, “Attention is all you need,” 2017.

[19] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes BT - Reinforcement Learning: State-of-the-Art*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[20] C. B. Ng, Y. H. Tay, and B. M. Goi, “Comparing image representations for training a convolutional neural network to classify gender,” *Proc. - 1st Int. Conf. Artif. Intell. Model. Simulation, AIMS 2013*, pp. 29–33, 2014, doi: 10.1109/AIMS.2013.13.

[21] K. Hornik, “Approximation Capabilities of Multilayer Neural Network,” *Neural Networks*, vol. 4, no. 1991, pp. 251–257, 1991.

[22] M. Hausknecht and P. Stone, “Deep reinforcement learning in parameterized action space,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–12, 2016.

[23] W. Masson, P. Ranchod, and G. Konidaris, “Reinforcement learning with parameterized actions,” *30th AAAI Conf. Artif. Intell. AAAI 2016*, pp. 1934–1940, 2016.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” pp. 1–12, 2017, [Online]. Available: http://arxiv.org/abs/1707.06347.

[25] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, “Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 5280–5289, 2017.

[26] C.-G. Grundig, *Fabrikplanung: Planungssystematik - Methoden - Anwendungen 5.,aktualisierte Auflage*. 2018.