Nested Rollout Policy Adaptation for Multiagent System Optimization in Manufacturing

Stefan Edelkamp[1] and Christoph Greulich[2]

[1]*Faculty 3 – Mathematics and Computer Science, University of Bremen, Bremen, Germany*

[2]*International Graduate School for Dynamics in Logistics, University of Bremen, Bremen, Germany*

Keywords: Multiagent System Simulation, Optimization, Monte-Carlo Tree Search, Manufacturing.

Abstract: In manufacturing there are not only flow lines with stations arranged one behind the other, but also more complex networks of stations where assembly operations are performed. The considerable difference from sequential flow lines is that a partially ordered set of required components are brought together in order to form a single unit at the assembly stations in a competitive multiagent system scenario. In this paper we optimize multiagent control for such flow production units with recent advances of Nested Monte-Carlo Search. The optimization problem is implemented as a single-agent game in a generic search framework. In particular, we employ Nested Monte-Carlo Search with Rollout Policy Adaptation and apply it to a modern flow production unit, comparing it to solutions obtained with a simulator and with a model checker.

1 INTRODUCTION

In this paper, we propose Nested Monte-Carlo Search for solving multiagent optimization problems by applying a search framework that links a (domain-specific) combinatorial problem to an implemented (domain-independent) search algorithm. To solve the problem, we selected a recent variant of Nested Rollout with Policy Adaptation (NRPA) (Edelkamp and Cazenave, 2016).

The application scenario we consider is an assembly-line network that is represented as a directed graph. Between any two successive nodes in the network, which represent entrances and exits of assembly stations as well as junctions, we assume a buffer of finite capacity. In those buffers, work pieces are stored, waiting for service. At assembly stations, service is given to work pieces. Travel time is measured and overall time optimized.

Especially in open, unpredictable, and complex environments, Multiagent Systems (MASs) determine adequate solutions for transport problems. For example, agent-based commercial systems are used within the planning and control of industrial processes (Dorer and Calisti, 2005; Himoff et al., 2006), as well as within other areas of logistics (Fischer et al., 1996; Bürckert et al., 2000), see (Parragh et al., 2008) for a survey.

Flow line analysis is often done with queuing theory (Manitz, 2008; Burman, 1995). Pioneering work in analyzing assembly queuing systems with synchronization constraints studies assembly-like queues with unlimited buffer capacities (Harrison, 1973). It shows that the time an item has to wait for synchronization may grow without bound, while limitation of the number of items in the system works as a control mechanism and ensures stability. Work on assembly-like queues with finite buffers all assume exponential service times (Bhat, 1986; Lipper and Sengupta, 1986; Hopp and Simon, 1989).

Our running case study is the so-called *Z2*, a physical monorail system for the assembling of tail-lights. Unlike most production systems, Z2 employs agent technology to represent autonomous products and assembly stations. The techniques we develop, however, will be applicable to most flow production systems. We formalize the production floor as a system of communicating agents and apply NRPA for analyzing its behavior optimizing the flow of production.

To make the paper self-contained we repeated the description of the Z2 and of the multiagent system. The contributions of this paper are: the encoding of the optimization problem as a single-player game, a flexible framework implementation, and a discussion of the advantages of employing an MCTS optimizer.

The paper is structured as follows. We kick off by introducing the Z2 as an example of a multiagent simulation system and formalize the multiagent optimiza-

Edelkamp S. and Greulich C.
Nested Rollout Policy Adaptation for Multiagent System Optimization in Manufacturing.
DOI: 10.5220/0006204502840290
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 284-290
ISBN: 978-989-758-219-6
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

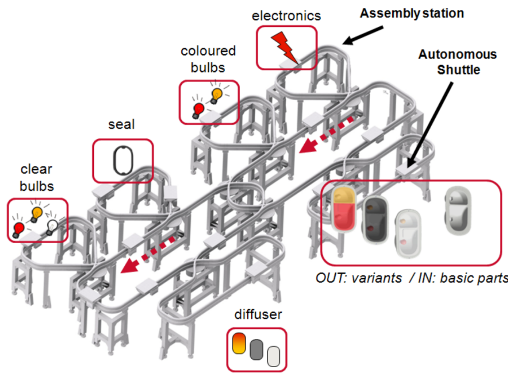

Figure 1: Assembly scenario for tail-lights (Morales Kluge et al., 2010).

tion problem we face. Next, we present the family of MCTS rollout algorithms including UCT, NMCS and NRPA. In the following section, we map the optimization problem to a single-player game which can be solved in our NMCS framework. We will see that the results compare positively with other optimization approaches for the same multiagent system. Finally, we conclude and discuss the impact of the work.

2 CASE STUDY: Z2

One of the few successful real-world implementations of a multiagent flow production is the so called *Z2* production floor unit (Ganji et al., 2010; Morales Kluge et al., 2010). The Z2 unit consists of six workstations where human workers assemble parts of automotive tail-lights. The system allows production of certain product variations as illustrated in Fig. 2 and reacts dynamically to any change in the current order situation, e.g., a decrease or an increase in the number of orders of a certain variant. At the first station, the basic metal-cast parts enter the manufacturing system on a dedicated shuttle. A monorail connects all stations, each station is assigned to one specific task, such as adding bulbs or electronics. The structure of the transport system is shown in Fig. 1. Each tail-light is transported from station to station until it is assembled completely. The monorail system has multiple switches which allow the shuttles to enter, leave or pass workstations and the central hubs. The goods transported by the shuttles are also autonomous, which means that each product decides on its own which variant to become and which station to visit. This way, a decentralized control of the production system is possible.

From the given case study, we derive a more general notation of an assembly-line network. System progress is non-deterministic and asynchronous,

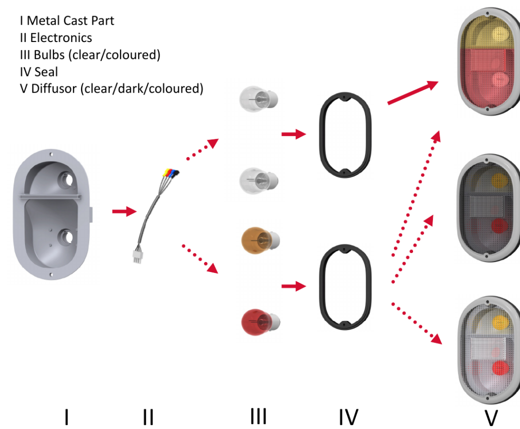

Figure 2: Assembly states of tail lights.(Ganji et al., 2010).

while the progress of time is monitored.

Definition 1 (Flow Production). *A* flow production floor *is tuple $F = (A,P,G,\prec,S,Q)$ where*

- *A is a set of all possible assembling actions*
- *P is a set of n products; each $P_i \in P$, $i \in \{1,\ldots,n\}$, is a set of assembling actions, i.e., $P_i \subseteq A$*
- *$G = (V,E,w,s,t)$ is a graph with start node s, goal node t, and weight function $w : E \to \mathcal{R}_{\geq 0}$*
- *$\prec = (\prec_1,\ldots,\prec_n)$ is a vector of assembling plans with each $\prec_i \subseteq A \times A$, $i \in \{1,\ldots,n\}$, being a partial order*
- *$S \subseteq E$ is the set of assembling stations induced by a labeling $\rho : E \to A \cup \emptyset$, i.e., $S = \{e \in E \mid \rho(e) \neq \emptyset\}$*
- *Q is a set of (FIFO) queues, all of finite size together with a labeling $\psi : E \to Q$*

Products P_i, $i \in \{1,\ldots,n\}$, travel through the network G, meeting their assembling plans/order $\prec_i \subseteq A \times A$ of the assembling actions A. The cost function uses a set of predecessor edges $Pred(e) = \{e' = (u,v) \in E \mid e = (v,w)\}$.

Definition 2 (Run, Plan, and Path). *Let $F = (A,P,G,\prec,S,Q)$ be a flow production floor. A* run *π is a schedule of triples (e_j,t_j,l_j) of edges e_j, queue insertion positions l_j, and execution time-stamp t_j, $j \in \{1,\ldots,n\}$. The set of all runs is denoted as Π. The run partitions into a set of n plans $\pi_i = (e_1,t_1,l_1),\ldots,(e_m,t_i,l_m)$, one for each product P_i, $i \in \{1,\ldots,n\}$. Each plan π_i corresponds to a* path*, starting at the initial node s and terminating at goal node t in G.*

3 MULTIAGENT SYSTEM

In the real-world implementation of the Z2 system, every assembly station, every monorail shuttle and ev-

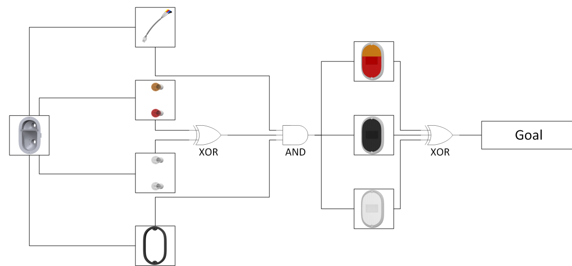

Figure 3: Preconditions of the various manufacturing stages.

ery product is represented by a software agent. Most agents in this MAS just react to requests or events which were caused by other agents or the human workers involved in the manufacturing process. In contrast, the agents which represent products are actively working towards their individual goal of becoming a complete tail-light and reaching the storage station. In order to complete its task, each product has to reach sub-goals which may change during production as the order situation may change. The number of possible actions is limited by sub-goals which already have been reached, since every possible production step has preconditions as illustrated in Fig. 3.

The product agents constantly request updates regarding queue lengths at the various stations and the overall order situation. The information is used to compute the utility of the expected outcome of every action which is currently available to the agent. High utility is given when an action leads to fulfillment of an outstanding order and takes as little time as possible. Time, in this case, is spent either on actions, such as moving along the railway or being processed, or on waiting in line at a station or a switch.

More generally, the objective of products in such a flow production system can be formally described as follows.

Definition 3 (Product Objective, Travel and Waiting Time)**.** *The* objective *for product i is to minimize*

$$\max_{1\leq i\leq n} \text{wait}(\pi_i)+\text{time}(\pi_i),$$

over all possible paths with initial node s and goal node t, where

- $\text{time}(\pi_i)$ *is the* travel time *of product P_i, defined as the sum of edge costs* $\text{time}(\pi_i) = \sum_{e\in\pi_i} w(e)$*, and*
- $\text{wait}(\pi_i)$ *the* waiting time*, defined as* $\text{wait}(\pi_i) = \sum_{(e,t,l),(e',t',l')\in\pi_i,e'\in Pred(e)} t-(t'+w(e'))$.

For this study, we provided the MAS model with timers to measure the time taken between two graph nodes. Since the hardware includes many RFID readers along the monorail, which all are represented by an agent and a node within the simulation, we simplified the graph and kept only three types of nodes: switches, production station entrances and production station exits. The resulting abstract model of the system is a weighted graph, where the weight of an edge denotes the traveling/processing time of the shuttle between two respective nodes (Greulich et al., 2015).

4 MONTE-CARLO TREE SEARCH

The randomized optimization scheme we consider belongs to the wider class of Monte-Carlo tree search (MCTS) algorithms (Browne et al., 2004). The main concept of MCTS is the random *playout* (or *rollout*) of a position, whose outcome, in turn, changes the likelihood of generating successors in subsequent trials. Prominent members in this class of reinforcement learning algorithms are *upper confidence bounds applied to trees* (UCT) (Kocsis and Szepesvári, 2006), and *nested monte-carlo search* (NMCS) (Cazenave, 2009). MCTS is state-of-the-art in playing many two-player games (Huang et al., 2013) or puzzles (Bouzy, 2016), and has been applied also to other problems than games like mixed-integer programming, constraint problems, function approximation, physics simulation, cooperative path finding, as well as planning and scheduling.

Cumulating in the success of AlphaGo (Silver et al., 2016) in winning a match of Go against a professional human player, the importance of MCTS in playing games and AI search is no longer doubted.

5 NESTED MONTE-CARLO SEARCH

Nested Monte-Carlo Search (NMCS) is a randomized search method that has been successfully applied to solve many challenging combinatorial problems, including Klondike Solitaire, Morpion Solitaire, Same Game, just to name a few. Recently, a large fraction of TSP instances have been solved efficiently at or close to the optimum (Cazenave and Teytaud, 2012a). NMCS compares well with other heuristic methods that include much more domain-specific information. NMCS is parameterized with the recursion *level* of the search which denotes the depth of the recursion tree is, and with the number of *iterations*, that shows the branching of the search tree. At each leaf of the recursive search a *rollout*, which performs and evaluates a random run.

What makes Nested Rollout Policy Adaptation (NRPA) (Rosin, 2011) notably different to UCT and

NMCS is the concept of learning a policy through an explicit mapping of moves to selection probabilities.

Beam-NRPA (Cazenave and Teytaud, 2012b) is an extension of NRPA that maintains B instead of one best solution in each level of the recursion. The motivation behind this is to warrant search progress by an increased diversity of existing solutions to prevent the algorithm from getting stuck in local optima. As the NRPA recursion otherwise remains the same, the number of playouts to a search with level L and (iteration) width N rises from N^L to $(N \cdot B)^L$. To control the size of the beam, we allow different beam widths B_l in each level l of the tree (our values for B_l in a level 5 search were $(1,1,10,10,10)$). At the end of the procedure, B_l best solutions together with their scores and policies are returned to the next higher recursion level. For each level l of the search, one may also allow the user to specify a varying iteration width N_l. This yields the algorithm Beam-NRPA to perform $\prod_{l=1}^{L} N_l B_l$ rollouts.

Beam-NRPA itself is inspired by the objective of higher diversity in the solution space of NRPA. Still, in very larger search spaces NRPA often dwells on inferior solutions. It simply takes too long to backtrack to less determined policies in order to visit other parts in the search space. High-Diversity NPRA (HD-NRPA) (Edelkamp and Cazenave, 2016) elaborates on this observation to increase the diversity of the search and provides some further algorithmic advances (e.g., instead of the moves executed in a rollout the policy table address of the chosen move and the code of its successors are stored, or the length of the rollout and its score are stored for each bucket in the beam).

6 GAME ENCODING

In the encoding as a single-player game, the amount of acting agents is significantly reduced in comparison to the original MAS. Similar to the encoding for model-checker-based approaches (Edelkamp and Greulich, 2016), decision making is modeled into the nodes while shuttles are merely integer values which are passed along the edges. Each edge is modeled as a queue to make sure that no shuttle can pass another. When put on an edge, a shuttle receives a waiting time which corresponds to the cost of the specific edge. A synchronizing function (Greulich and Edelkamp, 2016) ensures that time progresses for all shuttles. The node at the end of a directed edge is allowed to receive a shuttle only if it is first in its queue and its waiting time has passed. If a shuttle can be received by a node, the node provides a legal move for each outgoing edge. Hence, a set of all legal moves over all active agents can be obtained.

```
class Arena {
  public:
  Move rollout[MaxLength];
  int length;
  Arena() {
    length = 0;
    for (int i = 0; i< STATIONS; i++) {
      switch2entrance[i]->clear();
      exit2switch[i]->clear();
      entrance2exit[i]->clear();
    }
    for (int i = 0; i< STATIONS + 2*HUBS; i++)
      switch2switch[i]->clear();
    for (int i = 0; i < SHUTTLES; i++) {
      wait[i] = 0; cost[i] = i * 70;
      goals[i] = 0; color[i] = i%2;
      metalcast[i] = 1; diffusor[i] = 0;
      electronics[i] = bulb[i] = seal[i] = 0;
      switch2entrance[5]->push(i);
    }
  }
%  int code (Move m) { return m; }
  int legalMoves (Move moves [MaxLegalMoves]) {
    int m[3], mvs = 0;
    while (mvs == 0) {
      for (int p = 0; p < agent.size(); p++) {
        int k = agent[p]->nextLegalMove(m);
        for(int l=0;l<k;l++)
          moves[mvs++] = p*3 + m[l];
      }
      if (mvs == 0) increase_time();
    }
    return mvs;
  }
  void play (Move m) {
    rollout[length++] = m;
    agent[m/3]->executeMove(m%3);
  }
  bool terminal () {
    int reached = 1;
    for (int j=0; j<SHUTTLES; j++)
      reached &= goals[j];
    return (reached) || length == MaxLength-1;
  }
  double score () {
    int maximum = 0, total = 0;
    for (int j=0; j<SHUTTLES; j++)
      if (cost[j] > maximum) maximum = cost[j];
    int reached = 0;
    for (int j=0; j<SHUTTLES; j++)
      reached += !goals[j];
    return (reached * 1000) + maximum;
  }
}
```

Figure 4: Code for Z2 multiagent system optimization.

To play the game, the player has to choose one of the agents and one of its actions as the next move. Goal of the game is to finish a predefined number of

```
class Agent {
  public:
  Agent() {}
  virtual void executeMove(int m) = 0;
  virtual int nextLegalMove(int* moves) = 0;
};
```

Figure 5: Code for abstract agent class.

```
class Switch : public Agent {
  public:
  int In, Out, Station, B, C;
  Switch(int in, int out, int s, int b, int c):
  Agent(),In(in),Out(out),Station(s),B(b),C(c) {}
  void executeMove(int move) {
    if (move == 0) {
      int Shuttle = switch2switch[In]->pop();
      wait[Shuttle] += C; cost[Shuttle] += C;
      switch2entrance[Station]->push(Shuttle);
    }
    if (move == 1) {
      int Shuttle = switch2switch[In]->pop();
      wait[Shuttle] += B; cost[Shuttle] += B;
      switch2switch[Out]->push(Shuttle);
    }
    if (move == 2) {
      int Shuttle = exit2switch[Station]->pop();
      wait[Shuttle] += B; cost[Shuttle] += B;
      switch2switch[Out]->push(Shuttle);
    }
  }
  int nextLegalMove(int* moves) {
    int mvs = 0;
    if (receives(SW2SW_EN,In,Station))
      moves[mvs++] = 0;
    if (receives(SW2SW_PASS,In,Station))
      moves[mvs++] = 1;
    if (receives(EX2SW, Station,Station))
      moves[mvs++] = 2;
    return mvs;
  }
};
```

Figure 6: Code for one agent.

products in the shortest possible time before a predefined length is exceeded. The smaller the makespan for each agent found by the algorithm the higher the score of the play.

More formally, the (board) game is defined as (B, b_0, d, F, r) where B is the set of (board) positions, in our case consisting of all queue content, shuttle locations, and their respective cost values. The start position s_0 has all shuttles and all queues being empty, $d: B \to 2^B$ specifies the set of allowed actions for each $q \in B$, The set of final positions F consists of all states in which either all the individual goals or the maximal step sized is reached, and $r: B \to \mathbb{N}$ is the score function adding a constant (e.g., 1000) for each individual unreached goal, on top of the maximum of the individual cost values.

```
void increase_time() {
  int min = INF, d = 1;
  for (int p = 0; p < SHUTTLES; p++)
    if (0 < wait[p] && wait[p] < min) min = wait[p];
  if (min < INF) d = min;
  for (int p = 0; p < SHUTTLES; p++)
  if (wait[p] - d >= 0) {
    wait[p] -= d; cost[p] += d;
  }
  else wait[p] = 0;
}
bool receives(int channeltype, int i, int station) {
  int result = 0;
  Channel* channel = NULL;
  switch(channeltype) {
    case EN2EX: channel = entrance2exit[i];
      break;
    case EX2SW: channel = exit2switch[i];
      break;
    case SW2SW_PASS: channel = switch2switch[i];
      break;
    case SW2SW_EN: channel = switch2switch[i];
      break;
    case SW2EN:
      if (entrance2exit[station]->length() >= 1)
        channel = NULL;
      else channel = switch2entrance[station];
      break;
  }
  if (channel != NULL && channel->length() > 0) {
    int shuttle = channel->front();
    if (wait[shuttle] <= 0)
      result = 1;
  }
  return result;
}
```

Figure 7: Code for increase-time and receive action.

The components of the game induce a tree in the natural way with B as nodes, root b_0, d as edges and the final positions as leaves. A play(out) is then a path in the tree from b_0 to some leaf.

The software implementation (see Fig. 4–Fig. 7) is based on a framework which allows to employ several search algorithms such as MCTS, NMCS, NRPA, BEAM-NRPA and HD-NRPA (Edelkamp and Cazenave, 2016). For our experiments, we only focused on HD-NRPA since it is the most advanced implementation and provided the best results.

7 EXPERIMENTS

For the evaluation we used a single core of a personal computer infrastructure (Ubuntu 14.04 LTS (x64), Intel Core i7-4500U, 1.8 GHz, 8 GB).

We experiment with a rising number of vehicles

Table 1: Efficiency of MCTS for a rising number of shuttles, compared to previously published results.

	MCTS			LVT	DES
N	Length	Cost	CPU	Cost	Cost
2	48	2:54	1s	3:24	2:53
3	72	2:59	1s	3:34	3:04
4	99	3:08	2s	3:56	3:13
5	123	3:13	2s	4:31	3:25
6	153	3:22	5s	4:31	3:34
7	186	3:38	5s	5:08	3:45
8	213	3:45	5s	5:43	3:55
9	240	3:52	5s	5:43	4:06
10	267	3:52	5s	5:43	4:15
20	540	5:16	5s	8:59	5:59

and compare the results with the discrete event system (DES) model (Edelkamp and Greulich, 2016) and local virtual time (LVT) model (Greulich and Edelkamp, 2016), both implemented in the SPIN model checker using its branch-and-bound facility. While DES was faster than LVT, it had semantical problems with the proper progression of time. Therefore, in the MCTS implementation we decided to use the semantics of the LVT model.

Table 1 shows that the MCTS implementation scales best. It shows the length of the plan, the simulation time (Cost), and the runtime for a growing number of vehicles. Here we do not enforce prerequisites, namely that shuttles are protected from driving into a station if they have not all required components available. Given that SPIN is a full-fledged model checker that analyzes the encoding of the problem on the source-code level (resulting of traces that have thousands of steps) the result could have been expected, even though the search space is huge.

The CPU time bound for MCTS was 5 seconds, the RAM requirements remained rather small, less than 4MB for the largest instance, while the competitors require hundreds of MBs. As with the DES/LVT model, in cost we measure travel time plus some initial waiting time.

To help the solver to find valid solutions, we extended the objective fuction $(\textit{reached} * 1000) + \textit{maximum}$ by the term $(e_r * 10) + (b_r * 10) + (s_r * 10) + (d_r * 100)$, where e_r, b_r, s_r, and d_r are the violations to the assembling status of electronics, bulbs, seals, and diffusors, respectively.

We observe that there is a difference in the simulation times of LVT and DES even for two shuttles. Hence, we decided to reimplement LVT and have the two cost functions in a close match. Table 2 shows that (due to RAM usage) this implementation of LVT has difficulties to scale and failed for four vehicles, while the MCTS remained sufficiently fast (for larger models the bound of 5s turned out to be insufficient to solve all models with no constraint violations).

Table 2: Efficiency of MCTS for a rising number of shuttles, compared to reimplementation.

	MAS	MCTS		LVT	
N	Cost	Cost	CPU	Cost	CPU
2	4:01	3:17	5s	3:03	$<$1s
3	4:06	3:23	5s	3:19	79s
4	4:46	3:41	5s	–	–
5	4:16	3:59	5s	–	–
6	5:29	4:30	5s	–	–

In Table 2 we also added the results of the simulated multiagent system, where the agents chose the color of the lamp dynamically based on fuzzy logic decision rules that take the incoming orders and observed current queue lengths into account.

8 CONCLUSION AND DISCUSSION

Monte-Carlo Tree Search is a general exploration strategy that leads to concise solver prototypes not only for games but for many combinatorial optimization problems including multiagent optimization problems.

We proposed the application of MCTS to evaluate amultiagent system that controls the industrial production of autonomous products. As the flow of material is asynchronous at each station, queuing effects arise and additional constraints make the problem NP-hard. Besides validating the design of the system, the core objective of this work was to find plans that optimize the throughput of the system.

We modeled the production line as a set of communicating agents, with the movement of items modeled as communication channels. Experiments showed that the implementation is able to analyze the movements of autonomous products for the model, subject to the partial ordering of the product parts. It derived valid and optimized plans with several hundreds of steps using NRPA. A generic search framework helped to perform policy-based benchmarking. Considering the simplicity of the code, the sequentiality of the execution on one CPU core, the obtained results are promising.

NRPA is one means to find such *needle in the haystack*. It intensify the search with increasing recursion depth. The nestedness and policy refreshments relate to exponential restarting strategies known to be effective in the SAT community (Gomes et al., 2000).

An open problem is to find necessary/sufficient

criteria for the convergence of NMCS/NRPA. While as in most MCTS algorithms based on rollouts, we have *probabilistic completeness* in the sense that an optimal solution can always be found by chance. However, through nesting and adapting policies the success likelihood can become arbitrarily small, so that for now we cannot say by certain, that the optimum will be reached.

REFERENCES

Bhat, U. (1986). Finite capacity assembly-like queues. *Queueing Systems*, 1:85–101.

Bouzy, B. (2016). An experimental investigation on the pancake problem. In *Computer Games: Fourth Workshop on Computer Games*, pages 30–43, Cham. Springer International Publishing.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2004). A survey of Monte Carlo tree search methods. 4(1):1–43.

Bürckert, H.-J., Fischer, K., and Vierke, G. (2000). Holonic transport scheduling with teletruck. *Applied Artificial Intelligence*, 14(7):697–725.

Burman, M. (1995). *New results in flow line analysis*. PhD thesis, MIT.

Cazenave, T. (2009). Nested monte–carlo search. In *IJCAI*, pages 456–461.

Cazenave, T. and Teytaud, F. (2012a). *Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows*, pages 42–54. Springer.

Cazenave, T. and Teytaud, F. (2012b). Beam nested rollout policy adaptation. In *ECAI-Workshop on Computer Games*, pages 1–12.

Dorer, K. and Calisti, M. (2005). An adaptive solution to dynamic transport optimization. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 45–51. ACM.

Edelkamp, S. and Cazenave, T. (2016). Improved diversity in nested rollout policy adaptation. In *German Conference on AI (KI 2016)*.

Edelkamp, S. and Greulich, C. (2016). Using SPIN for the optimized scheduling of discrete event systems in manufacturing. In *SPIN 2016*, pages 57–77. Springer.

Fischer, K., Müller, J. R. P., and Pischel, M. (1996). Cooperative transportation scheduling: an application domain for dai. *Applied Artificial Intelligence*, 10(1):1–34.

Ganji, F., Morales Kluge, E., and Scholz-Reiter, B. (2010). Bringing Agents into Application: Intelligent Products in Autonomous Logistics. In *Artificial intelligence and Logistics (AiLog) - Workshop at ECAI 2010*, pages 37–42.

Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1-2):67–100.

Greulich, C. and Edelkamp, S. (2016). Branch-and-bound optimization of a multiagent system for flow production using model checking. In *ICAART 2016*.

Greulich, C., Edelkamp, S., and Eicke, N. (2015). Cyber-physical multiagent simulation in production logistics. In *MATES 2015*.

Harrison, J. (1973). Assembly-like queues. *Journal of Applied Probability*, 10:354–367.

Himoff, J., Rzevski, G., and Skobelev, P. (2006). Magenta technology multi-agent logistics i-scheduler for road transportation. In *AAMAS 06*, pages 1514–1521. ACM.

Hopp, W. and Simon, J. (1989). Bounds and heuristics for assembly-like queues. *Queueing Systems*, 4:137–156.

Huang, S.-C., Arneson, B., Hayward, R. B., Mueller, M., and Pawlewicz, J. (2013). Mohex 2.0: A pattern-based MCTS Hex player. In *Computers and Games*, pages 60–71.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *ECML*, pages 282–293.

Lipper, E. and Sengupta, E. (1986). Assembly-like queues with finite capacity: bounds, asymptotics and approximations. *Queueing Systems*, pages 67–83.

Manitz, M. (2008). Queueing-model based analysis of assembly lines with finite buffers and general service times. *Computers & Operations Research*, 35(8):2520 – 2536.

Morales Kluge, E., Ganji, F., and Scholz-Reiter, B. (2010). Intelligent products - towards autonomous logistic processes - a work in progress paper. In *Intern. PLM Conf.*

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008). A Survey on Pickup and Delivery Problems Part II: Transportation between Pickup and Delivery Locations. *Journal für Betriebswirtschaft*, 58(2):81–117.

Rosin, C. D. (2011). Nested rollout policy adaptation for monte carlo tree search. In *IJCAI*, pages 649–654.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS