Evolutionary Planning Heuristics in Production Management

Steven de Jong Nico Roos Ida Sprinkhuizen-Kuyper

Department of Computer Science, Universiteit Maastricht, Netherlands
Email: {steven.dejong, roos, kuyper}@cs.unimaas.nl

Abstract

Resource management problems are problems in which the goal is to *optimize the utilization of resources over time*. Dependencies between resources make it necessary to plan the proper utilization of these resources. Such a planning must often be constructed in an environment that is non-deterministic, for example due to competing agents. Traditional state-based planning approaches are not suited for this type of planning problems.

This paper addresses an important subclass of resource management problems, namely *production management problems*. In this class of problems, the production of a set of goal products must be optimized over time, in the absence of non-determinism and competing agents. Three methods for production management problems are investigated, viz., an extensive search method and two abductive planners using greedy search. One abductive planner uses a heuristic that has been designed especially for production management problems while the other abductive planner learns a heuristic from experience.

Experiments performed on a large and varied set of production management problems show that extensive search cannot be used for production management problems and that the abductive planner using the learned heuristic significantly outperforms the abductive planner using the designed heuristic. This indicates that the integration of learning elements in planning algorithms can improve the performance and versatility of these algorithms.

1 Introduction

Traditional planning techniques focus on the task of making a plan to achieve some fixed goal. Efficient algorithms for this task have been developed over the last decades [4, 8]. Different planning tasks arise in dynamic environments where the goal is to optimize the utilization of resources over time. Such planning problems have recently been studied in the context of *resource management games* [3, 7]. Resource management games are interactive computer games in which the objective is to use limited resources in order to construct and maintain a complex virtual environment. These games are relevant for AI research, not only because of their entertainment value, but also because they are modelled after real-world resource management problems and are regularly used for educational purposes at universities, business schools and military organizations [9].

Resource management problems[1] are in essence continuous planning problems in which the agent (human or computer) performing the management task must construct a sequence of actions that is expected to lead to a desired outcome. Like most planning problems, resource management problems are intractable and thus, large instances cannot be calculated completely [1]. However, there are three major differences between classical planning problems and the planning problems encountered in the domain of resource management. First, the focus of resource management problems is on resource utilization, and consequently, the goal is parameter optimization over time – maximizing or balancing one or more resources – whereas in classical planning, the goal is defined by a finite number of desirable states. Thus, the performance of a planning technique

[1] We will use the term 'resource management problems' to indicate the problems present in both computer games and the real world.

in resource management problems is not directly related to the efficiency, length or costs of the actual plan [7]. Second, actions can have side effects; these side effects can provide a player with unexpected opportunities or threats. This is not often the case in classical planning problems. Third, there can be non-deterministic factors and imperfect information, due to the environment and competing agents [6]. This is also not often the case in classical planning problems. In summary, resource management problems can be defined as planning problems in a potentially non-deterministic environment, which require parameter optimization over time and can be addressed by the selection of actions with side effects.

In [7], Nayerek presents an overview of planning systems that are able to handle resources. He concludes that these planning systems are flawed, because in most cases the resources are only used as auxiliary conditions to guarantee a valid plan. Even in most approaches that include an optimization of resource-related properties, the plan length still serves as the primary optimization criterion.

In this paper, we will investigate whether abductive planning, combined with unsupervised learning of planning heuristics, is a suitable approach for resource management problems. We will address this issue in a subdomain of resource management problems, namely *production management problems*. These problems contain the following elements. First, there is a fixed set of *products*. Second, there is a fixed set of *production actions*, which are used to convert certain products into other products or to obtain one or more products. These actions may require time and money. Some actions may also produce money (selling actions). Third, there are *constraints* such as a limited amount of time or money available. Fourth, the goal for a problem solver is to produce as much as possible of some of the products, denoted as the *goal products*. This can be done by developing an optimal sequence (or plan) of actions, given the available products, actions, constraints and money. Production management problems are a good domain of choice for this research, for three reasons, viz. (1) they are easy to formalize, (2) they are easily scalable to any desired size and (3) it is possible to develop difficult production management problems without including competing agents and other sources of uncertainty.

In section 2, we present a formalization of production management problems and analyze the complexity of such problems. In section 3, we discuss the methods used in this research. In section 4, we look at the experimental setup and the results obtained with regard to performance and running times of the methods discussed. In section 5, we conclude and look at future work.

2 Formalizing production management problems

In this section, we present a formalization of production management problems. In our study of the planning aspects of these general problems, we have focused on a simplified version, for which we ignore non-determinism and earning additional money that can be utilized to cover the costs of actions. The resulting problem can be formalized as follows.

- A problem is a tuple $< P, G, S, A, r, c, t, in, out, n, m >$.
- P is the set of products $\{p_1, \ldots, p_n\}$. The *size* of the problem, $|P|$, is denoted by n. The variable m denotes the amount of money initially available.
- The *problem state* s_t at a certain moment in time can be formalized as a vector as follows: $(m_s, q_1, \ldots, q_n)$. Here, m_s is the money available in this state, and the function $q_s : P \rightarrow \mathbb{N}$ defines the quantity available for each product in the state s given, i.e., q_1 for p_1, et cetera. The possibly infinite set of possible problem states is denoted by S and the start state by $s_0 \in S$.
- $G \subset P$ is the set of goal products. The reward per goal product, $r : P \rightarrow \mathbb{R}^+$, is specified by $p \notin G \rightarrow r(p) = 0$ and $p \in G \rightarrow r(p) > 0$. The total reward in a state s can be calculated using the function $R(s) = \sum_{p \in G} r(p) \cdot q_s(p)$. The goal of the problem is to reach a state s in which $R(s)$ is optimal.

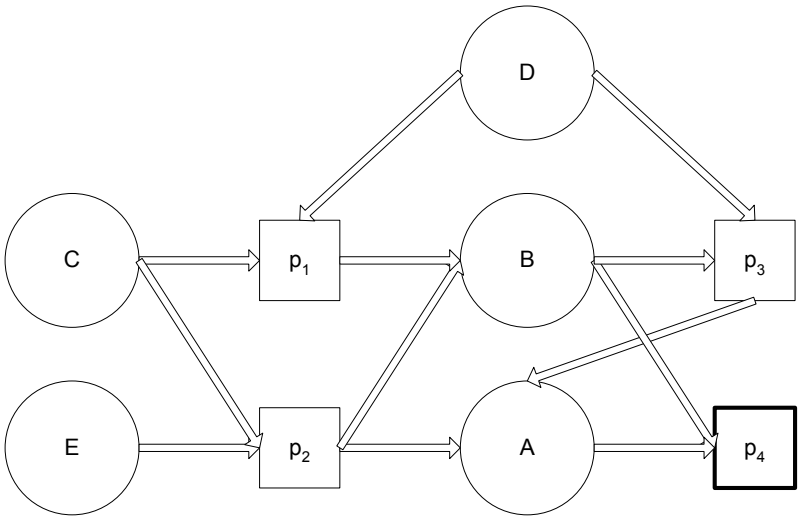

Figure 1: A small example production management problem that adheres to the formalism and constraints presented in this paper. For legibility, all coefficients have been omitted. The circles represent actions and the squares represent products. For example, action A requires products p_2 and p_3 and produces the goal product p_4.

- A denotes the set of actions that enable the transition from one state to another. Here, $c : A \to \mathbb{N}$ denotes the cost of action a and $t : A \to \mathbb{N}$ denotes the time action a takes to complete, assuming discrete timesteps. The function $in : A \to \wp(P \times \mathbb{N}^+)$ denotes the amounts of products required to execute a. Similarly, $out : A \to \wp(P \times \mathbb{N}^+)$ denotes the amounts of products produced by a if it is executed.

For this research, we introduce five additional constraints in addition to this formalism. Problems based on the formalism and these additional constraints fall more or less within the same class of difficulty and possess some convenient properties such as a strict ordering of actions. However, problems that adhere to these additional constraints are not essentially easier than problems that do not adhere to them.

1. Every action requires at most two products and produces one or two products. Every product is produced by exactly two actions.
2. Costs x and durations y of actions can be selected from a limited domain: $x \in c(a) \to x \in \{1, 2, 3, 4\}$ and $y \in t(a) \to y = 1$. Similarly, we limit the domain for the coefficients x for required and produced products p by implying the condition $(p, x) \in in(a) \vee (p, x) \in out(a) \to x \in \{1, 2, 3, 4\}$.
3. Cycles in the product chain are prevented by ensuring that an action cannot produce products with a lower index than the highest index of its required products, plus one.
4. We define $G = \{p_n\}$ and $r(p_n) = 1$.
5. The problem solver starts with an initial amount of money equal to $20 \cdot n$ (where n is the size of the problem) and none of the products present.[2] Thus, $s_0 = \{20 \cdot n, 0, \ldots, 0\}$.

A small example problem that adheres to the formalism and constraints presented here, is shown in figure 1 above.

[2]The number $20 \cdot n$ is intentionally rather small to keep the problem complexity manageable.

Experimental results obtained using an algorithm that can generate such problems randomly, show that $|A| \approx n + 1$. Solving such problems is a hard task; they are NP-hard problems[3] [1] and their search tree complexity can be derived as $(n + 2)^{8 \cdot n}$ on average. Obviously, search tree complexity alone is not a measure of problem complexity, because there might exist intelligent algorithms for quick traversal of this tree regardless of its size. However, brute force search algorithms will most definitely not be able to deal with problems with a large search tree complexity.

3 Methods

In this section we present three methods that have been evaluated using a set of production management problems based on the formalism and constraints presented in the previous section. First, we will look at straightforward depth first search, which is used to assess whether for small problems, an optimal solution can be determined.

Second, we will look at an abductive planner using a fixed heuristic especially designed for these production management problems. We have chosen for an abductive planner here in favour of a forward planner, because the former gives some guidance for leading the search in the right direction. Other planning algorithms, such as fast forward planning [5], which we plan to investigate in future research, place stronger demands on the heuristic used, because they do not guarantee that a plan actually leads to the goal.

Third, we will look at an abductive planning algorithm in which the planner has learned a heuristic through a combination of a neural network and a genetic algorithm. The task for the genetic algorithm is assumed to be less complex when it is used as a heuristic for an abductive planner than when it is used as a heuristic for a forward planner, because of the reason mentioned above.

3.1 Depth first search (DFS)

The depth first search algorithm performs extensive search: it traverses the entire search space in order to find a plan that produces the largest amount of the goal product. It starts in the initial problem state and then creates a tree by generating the problem state that results from applying each of the possible actions to this initial problem state. This process is repeated on each of the resulting states, until the end conditions hold, i.e., states are reached in which there is no money left for any action. Then, $R(s)$ is determined in each of these terminal states and the maximum $R(s)$ is returned along with the path that leads to the terminal state in question. Obviously, due to the complexity of the search tree, this method can only be used for small production management problems.

3.2 Fixed planning heuristics (FPH)

The fixed planning heuristics algorithm does not produce a complete plan of actions ahead of time, but rather generates a new plan every time an action must be selected. Although this is not a requirement for our current testset of production management problems, the algorithm is able to deal with non-deterministic factors because it can adapt its plan everytime it is executed.

The first step of the algorithm consists of determining the actions that produce the goal product p_n. From these actions, it chooses the one that contributes the most to the goal at the lowest price. If this action is executable, it is returned as the current action. If it is not executable, the algorithm finds the best action that makes the preferred action as executable as possible. This cannot be calculated exactly for all actions, due to the problem structure,[4] and is therefore determined using

[3] One can easily reduce a standard planning problem to a production management problem that adheres to the formalism and additional constraints presented above.

[4] The main problem here is that actions can have side-effects; they may produce a second product in addition to the product we are interested in. This second product might, or might not, be useful in the future. Therefore, in order to determine the exact value of an action, we should perform an analysis of all sequences of actions starting with this particular action. Performing this analysis is a task similar to performing an extensive search process and is therefore only feasible for small problems.

a heuristic that provides approximate information (this heuristic will be described below). This process continues until an action is found that is executable. Thus, in essence, the algorithm is a greedy abductive planner [8] that includes a manually designed heuristic for efficient traversal of the search tree, without any backtracking. We will now explain the heuristic developed for production management problems.

The fixed planning heuristic determines a heuristic value for each of the actions *independent* of the current state of the problem.[5] For every action a, this heuristic value, κ, denotes the cost of manufacturing exactly one unit of the goal product, using a product chain that begins with action a. It only examines such a product chain for which these costs are lowest, i.e., an optimal product chain. Furthermore, it assumes that all subsequent actions of this product chain can be executed and that the additional products that might be needed for this, are all present. Thus, actions with a low κ value are expected to be more attractive starts of a product chain than actions with a high κ value.

The heuristic value κ is calculated as follows. Let $a_{i,1}, \ldots, a_{i,n}$ be the optimal product chain i for the product p_n and let $p_{i,1}, \ldots, p_n$ be all products produced in this product chain. Thus, $p_{i,j}$ is produced by $a_{i,j}$. We define $\kappa(a_{i,j})$ as the costs for producing one unit of p_n using the part of the product chain starting at $a_{i,j}$, and we define $\pi(p_{i,j})$ as the number of products $p_{i,j}$ needed to produce one unit of p_n at the end of the product chain. Then:

$\pi(p_n) = 1$.
$\kappa(a_{i,n}) = \frac{c(a_{i,n})}{x}$ with $(p_n, x) \in out(a_{i,n})$.
$\pi(p_{i,j}) = \pi(p_{i,j+1}) \cdot \frac{y}{x}$ with $(p_{i,j}, y) \in in(a_{i,j+1})$ and $(p_{i,j+1}, x) \in out(a_{i,j+1})$ and $j \neq n$.
$\kappa(a_{i,j}) = \pi(p_{i,j}) \cdot \frac{c(a_{i,j})}{x} + \kappa(a_{i,j+1})$ with $(p_{i,j}, x) \in out(a_{i,j})$ and $j \neq n$.

The planning algorithm can now use the heuristic value κ, starting at an action $a_{i,n}$, to find an action that is executable and is expected to contribute the most to the production of the goal product.

3.3 Evolutionary planning heuristics (EPH)

The evolutionary planning heuristics algorithm is a combination of the aforementioned method and evolutionary computation. Instead of a manually designed heuristic for the traversal of the plan space search tree, the algorithm uses a heuristic that is generated by a neural network. In contrast to a manually designed heuristic, this heuristic does not have to be state-independent. The input of the fully connected network driving the heuristic, consists of the current problem state s_t. The output consists of a preference value for each action. The neural network is trained by weight optimization with a genetic algorithm, by means of offline learning:[6] a single problem is repeatedly offered until the algorithm, starting from the state s_0, is able to reach a state s in which the reward $R(s)$ is satisfactory.

This approach follows quite intuitively from the FPH approach, since the FPH approach has an obvious disadvantage. As has been mentioned earlier, it is difficult or even impossible to manually design a heuristic that can use the current problem state as one of its inputs. Therefore, a manually designed heuristic misses the ability to adapt its behavior to this current problem state. Moreover, in the case of rather small problems, we might be able to provide the planner with sufficiently accurate state-independent heuristics, but in larger problems, the algorithm must function without additional information (such as a heuristic) programmed in ahead of time, mainly because developing a sufficient heuristic for larger problems requires so much understanding each of these problems that it might be easier to actually solve them all manually.

The EPH algorithm, resulting from integrating learning mechanisms in a planning method, combines the strengths of the FPH approach (goal-directed planning, fast tree traversal and the

[5] Obviously, when a problem is assessed by the abductive planner, the state of the problem does play a role, because some actions are executable in certain states and others are not. However, developing a planning heuristic that takes the current state into account is very difficult, if not impossible.

[6] Future research will focus on non-terminating production management problems, in which money is a resource that is produced by one or more *selling* actions. In such problems, online learning becomes an interesting topic for investigation.

ability to deal with non-determinism) with the advantages of a learning method, resulting in a planner that (1) has been optimized for a certain problem without requiring an external agent to analyze the problem and (2) can adapt its behavior to the current state of this problem.

4 Experiments and results

Using a set of experiments, we evaluated the three methods discussed in the previous section. The test set consisted of 5,000 randomly generated production management problems, adhering to the formalism and constraints presented in section 2, with n randomly chosen from $\{1, 2, \ldots, 30\}$.

We compared the performance of the fixed planning heuristics (FPH) and evolutionary planning heuristics (EPH) algorithms with that of the depth-first search (DFS) algorithm to determine whether the heuristic algorithms could lead to acceptable or even optimal solutions, compared to the optimal solution found by extensive searching. For problems with $n = 1$, we were able to compare the solutions obtained by these approaches and concluded that EPH and FPH always find the optimal solution. Unfortunately, DFS was not able to produce solutions for problems with $n > 1$ due to the complexity of these problems. For example, the search tree complexity of a problem with size 2 is already $(2+2)^{16} \approx 4.2 \times 10^9$, leading to predicted running times per problem of approximately 21 days (based on an average running time of 800ms for $n = 1$). Therefore, we could not assess whether the solutions produced by EPH or FPH are optimal for all problem sizes generated. The performance of these algorithms can only be determined by comparing them with eachother.

In the remainder of our experiments, we did not use the DFS algorithm, but only compared the solutions of the FPH approach with those of the EPH approach. The latter approach was allowed to finetune until no improvement had been made for 100 generations in the genetic algorithm. Each problem was offered 10 times to eliminate artefacts due to randomness in EPH. The solutions for each problem were averaged for each of the problem solvers. Next, the solutions were divided in classes; one for each problem size. We analyzed the performance of both algorithms in three ways.

First, we analyzed the average solution for each class, obtained by the two algorithms. For 10 of the 30 classes, this average solution and the average relative percentage $\frac{EPH}{FPH} \cdot 100\%$ are summarized in figure 2. To determine how the solutions were distributed, we constructed a graphical representation of the performance of FPH and EPH. This performance was expressed as $EPH - FPH$. The performances for all problems within a class were calculated, sorted in ascending order, and then plotted in a graph. The graph for the class with $N = 18$ is presented in figure 3, which intuitively shows that EPH performs better than FPH in this class. Similar graphs can be constructed for the other classes, with comparable results.

Problem size	3	6	9	12	15	18	21	24	27	30
FPH average	51.4	75.6	80.3	112.3	115.7	132.1	141.0	169.7	109.0	165.3
EPH average	62.0	91.4	117.7	143.7	152.1	170.3	196.0	200.1	205.0	192.1
(EPH / FPH) * 100% average	120.5	120.7	146.6	128.0	131.4	129.0	138.9	118.2	188.2	116.2
P(EPH > FPH)	1	1	1	1	1	1	1	0.954	0.929	0.889

Figure 2: Results of comparing FPH and EPH on 10 classes of problems, including average solution rewards, percentual difference between the algorithms and probability that EPH is superior.

Second, a randomization test [2] was performed on each of the classes with the hypothesis H_0 : (average EPH = average FPH). Figure 2 shows the resulting probabilities that H_0 should be rejected in favour of the alternative hypothesis H_1 : (average EPH > average FPH), for 10 of the 30 classes. As can be seen from the figure, for problems with a size up to 21, $P(\textit{reject } H_0) = 1$. For larger problems, this probability decreases slowly.

Third, we compared the time complexity of the FPH and EPH algorithms. Results for 10 of the 30 classes are displayed in figure 4. It is clearly visible that the FPH algorithm requires only a small amount of time for any problem size. The EPH algorithm requires more time due to the learning process taking place (up to 15 minutes for the largest problems with $N = 30$, on an Intel Pentium IV 2400). However, the performance of EPH is still very pleasing, because of two reasons: (1) the

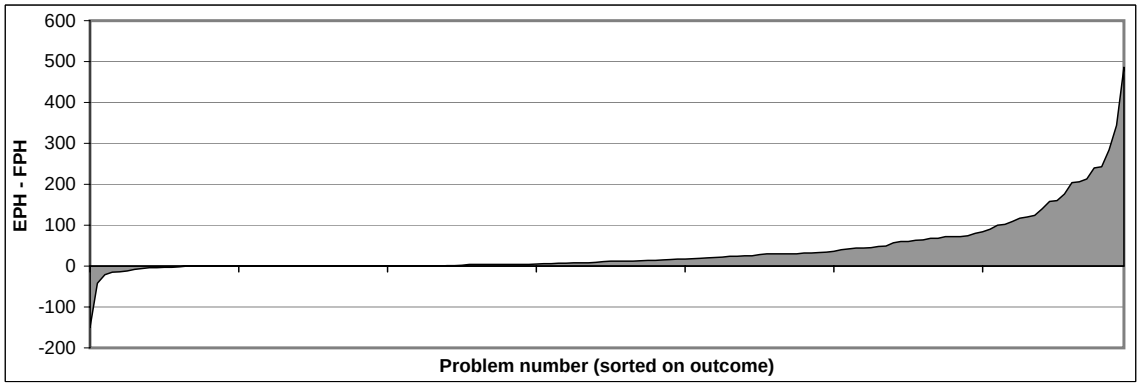

Figure 3: The results of EPH and FPH on 141 problems with $N = 18$, sorted in ascending order.

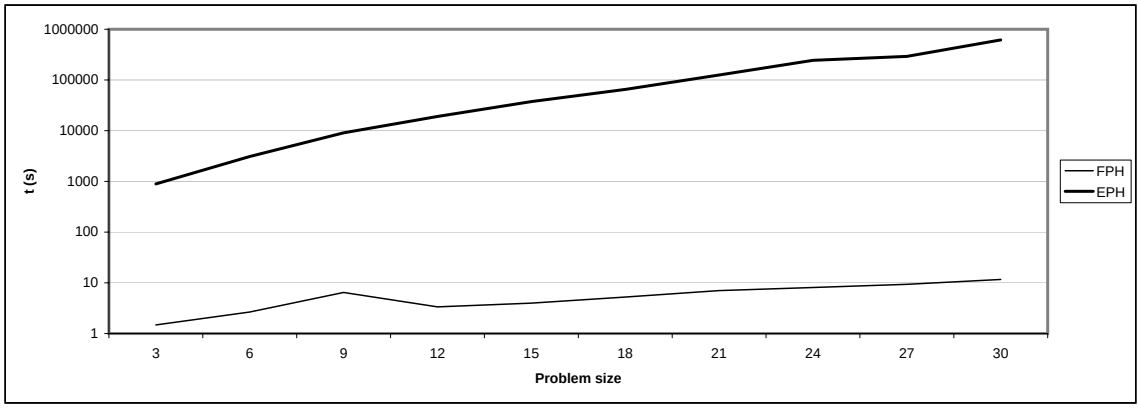

Figure 4: Running times to produce a solution (in milliseconds) of the FPH and EPH algorithms for 10 classes of problems.

problems addressed by the algorithm are NP-hard (and cannot be solved in a reasonable amount of time by traditional search techniques) and (2) the algorithm comes up with good solutions for large problems.

5 Conclusion

The experiments performed indicate that extensive search is not a suitable method for production management problems; it can only solve the smallest problems. Both our planning algorithms did come up with results for each of the production management problems assessed by them, ranging in size between 1 and 30 products. The evolutionary planning heuristics approach (EPH) almost always comes up with better solutions than the fixed planning heuristics approach (FPH) on these problems. The EPH approach has more advantages than just this performance increase over our manually designed heuristic. We will briefly name three of these advantages. First, the approach is designed to handle non-determinism or changes in the problem definitions [3]. Second, it enables us to develop a strong and state-dependent solver for a difficult class of problems without requiring us to manually program the results of an in-depth analysis of such problems. Third, it produces solutions in a much shorter period of time than a traditional search technique and thus enables us to solve intractable (NP-hard) problems up to a reasonably large size.

For future work, many aspects of production management problems, and resource management problems in general, have to be covered by both the problem formalization and the EPH approach. For example, the problems we experimented upon did not include non-determinism or adversaries,

the actions within these problems all took a constant amount of time, problems required a finite number of actions, and the problem solver had no way of generating money during the production process. Furthermore, more approaches should be tested, such as a combination of forward planning and evolutionary planning heuristics, or plan refinement strategies that use evolutionary optimization techniques. Finally, we explicitly terminated the learning process of EPH using a predetermined criterium. The satisfactory results for the largest problems tested ($n = 30$) can definitely increase even further if we allow for longer learning times or more tuned learning. Facilitating this will make the EPH method more scalable.

References

[1] T. Bylander. Complexity results for planning. In *Proceedings of IJCAI-91, Sydney, Australia*, pages 274–279, 1991.

[2] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 2002.

[3] Steven de Jong, Pieter Spronck, and Nico Roos. Requirements for resource management game AI. In D.W. Aha, H. Muñoz-Avila, and M. van Lent, editors, *Reasoning, Representation, and Learning in Computer Games: Proceedings of the IJCAI Workshop* (Technical Report AIC-05-127), pages 43–48. Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, 2005.

[4] Mathijs de Weerdt. *Plan Merging in Multi-Agent Systems*. PhD thesis, Universiteit Delft, the Netherlands, 2003.

[5] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. In *Journal of AI research*, volume 14, pages 253–302, 2001.

[6] Rune Jensen, Manuela Veloso, and Michael Bowling. Optimistic and strong cyclic adversarial planning. In *Proceedings of ECP-2001, European Conference on Planning*, Toledo, Spain, October 2001.

[7] A. Nareyek. Beyond the plan-length criterion. In A. Nareyek, editor, *Local Search for Planning and Scheduling*, pages 55–78. Springer LNAI 2148, 2001.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.

[9] B. Sawyer. Serious games: Improving public policy through gamebased learning and simulation. *Foresight and Governance Project, Woodrow Wilson International Center for Scholars*, 2002.