Nat Comput
DOI 10.1007/s11047-012-9332-y

# Goal distance estimation for automated planning using neural networks and support vector machines

**Benjamin Satzger · Oliver Kramer**

© Springer Science+Business Media B.V. 2012

**Abstract** Many of today's most successful planners perform a forward heuristic search. The accuracy of the heuristic estimates and the cost of their computation determine the performance of the planner. Thanks to the efforts of researchers in the area of heuristic search planning, modern algorithms are able to generate high-quality estimates. In this paper we propose to learn heuristic functions using artificial neural networks and support vector machines. This approach can be used to learn standalone heuristic functions but also to improve standard planning heuristics. One of the most famous and successful variants for heuristic search planning is used by the Fast-Forward (FF) planner. We analyze the performance of standalone learned heuristics based on nature-inspired machine learning techniques and employ a comparison to the standard FF heuristic and other heuristic learning approaches. In the conducted experiments artificial neural networks and support vector machines were able to produce standalone heuristics of superior accuracy. Also, the resulting heuristics are computationally much more performant than related ones.

**Keywords** Planning · Artificial neural network · Support vector machine · Search heuristic · Regression

B. Satzger (✉)
Distributed Systems Group, Vienna University of Technology, Vienna, Austria
e-mail: benjamin.satzger@gmail.com; satzger@dsg.tuwien.ac.at

O. Kramer
Computational Intelligence Group, University of Oldenburg, Oldenburg, Germany
e-mail: oliver.kramer@uni-oldenburg.de

## 1 Introduction

Planning is a concept to enable computer systems to reason about actions. It can be used to build intelligent goal-driven computer systems. In such a way one can specify *what* is expected from the system, while it is able to autonomously determine *how* this can be achieved. Due to the progress in defining heuristics which estimate the goal distance of planning states, forward search within the space of planning states is one of the most successful and prominent planning approaches. Heuristics are used to guide search in state-space. A common approach for heuristic estimation is to construct an easier to solve relaxation of a planning problem and to use the solution to the relaxed problem as heuristic for the original problem. One of the best performing planners, Fast-Forward (FF) (Hoffmann 2001), is based on this idea. It does not guarantee to always find the shortest solution and thus belongs to the category of non-optimal planners.

This work aims at learning goal distance estimates based on methods from natural computing, such as artificial neural networks (ANNs) (Rumelhart et al. 1988) and support vector machines (SVMs) (Cortes and Vapnik 1995), with solved problem instances as training data. In contrast to the FF heuristic that relies on solving relaxed planning problems, the learned heuristics are able to extract relevant knowledge about a domain during the training phase and allow for a fast retrieval of estimates subsequently. ANNs are biologically inspired and SVMs are closely related to ANNs (Swiercz et al. 2008). A SVM model using a sigmoid kernel function is equivalent to a two-layer perceptron neural network. Both are well-known machine learning techniques. To the best of our knowledge, there is no approach for automated planning goal distance estimation using ANNs and SVMs. Also, other researchers have not investigated purely learned standalone heuristics. The

Published online: 26 July 2012

most related approach by Yoon et al. (2006, 2008), which also applies learning to generate search heuristics, is based on augmenting the FF heuristic. Our experiments show that the standalone heuristics based on ANNs and SVMs outperform both FF's heuristic and the heuristic presented by Yoon et al. in terms of accuracy and computational overhead for generating heuristic estimates. Also, the training needs orders of magnitudes less time compared to the Obtuse Wedge[1] (OW) planner that implements the approaches describe in Yoon et al. (2006, 2008).

However, in contrast to most research in the area of learning-assisted planning (including OW), in which the planning problems comprising the training data may be of different size from the planning problems that can be handled by the learned heuristics, in this work the learned heuristics are only applicable if the characteristics of the test and training problems are the same. There are many practical scenarios in which this limitation does not affect the usefulness of our approach. In fact, the application scenario that led to this work is to use intelligent agents for controlling complex computer systems. Agents use automated planning to figure out, which actions need to be executed in order to keep the system well configured. The agent is committed to goals established by the administrator of the computer system. Planning actions are defined by the abilities of the agent to influence and reconfigure the system. Monitoring and sensing gathers the current state of the system. An advantage of this use-case is that humans only need to define high-level goals, while the agent is responsible for complying with them. This can be used for shifting management responsibilities from humans and to create systems with self-management capabilities, as described in Satzger et al. (2008). In such scenarios the managing agent would have to deal with planning problems of the same size, for which the goals do not change (at least over a longer period of time). Motivated by the introduced use-case the heuristic learning approach presented in this paper is based on the assumption that problem sizes and goals do not change. In this use-case, the agent in charge of controlling the computer system could initially be based on a traditional planner, which would be used to figure out necessary management actions by solving planning problems. The solutions can be reused as training input for the heuristics proposed in this work; when the agent has gathered enough experience the planning can be based on learned heuristics, providing a significant performance benefit. In the conclusion section we discuss approaches to support generalization and changing goals.

In previous publications (Satzger et al. 2010; Satzger and Kramer 2010) we have introduced the general idea of using machine learning for generating search heuristics for planning. However, this work uses different machine learning approaches and a different evaluation methodology. In particular, SVMs, one of the most popular and powerful machine learning approaches, have not been considered. Also, previous work does not investigate the scalability of the heuristics with respect to problem size nor the training times. The evaluation presented in the paper is based on comparing results with an oracle, an approach not used in previous work. Finally, it includes a comparison to the learning-augmented heuristics implemented in OW. The main contributions of this paper are:

1. application of ANNs and SVMs to goal distance estimation in automated planning,
2. comparison of purely learned standalone heuristics to a standard heuristic (i.e., the FF heuristic) and learning-augmented heuristics (e.g., OW), and
3. experimental evaluation of all considered techniques.

The remainder of the paper is structured as follows. Section 2 gives a rough overview of automated planning approaches and introduces heuristic search planning. Related work in the area of learning and planning is presented in Sect. 3. Section 4 describes our approach for learning heuristic functions. In Sect. 5, results of conducted experiments comparing different learned heuristics to the FF heuristic are presented. Finally, in Sect. 6, the paper is concluded and opportunities for future work are described.

## 2 Automated planning

The conceptual model of a planning environment can be represented as *state transition system* of the form $\Sigma = (S, O, \gamma)$, where

- $S = \{s_1, \ldots, s_n\}$ is a set of states,
- $O = \{o_1, \ldots, o_m\}$ is a set of actions (also called operators), and
- $\gamma : S \times O \rightarrow S$ is a partially defined state transition function.

Using this notation, a planning problem can be described as follows. Given a description of $\gamma$, an initial state $s_i \in S$, and a set of goal states $S_g \subseteq S$, find a sequence of actions transforming the system from $s_i$ to a state $s_g \in S_g$.

The widely used STanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson 1971; Ghallab et al. 1998) representation of planning problems introduces restrictions resulting in a more practical and compact model. A STRIPS problem is a quadruple $P = (A, O, I, G)$. A finite set of atoms $A$ forms the basis for describing states, goals, and actions. States are subsets of $A$, with an atom $a$ considered being *true* in a state $s$ iff $a \in s$. The initial state $I \subseteq A$ is given by the set of initially *true* atoms. The goal representation $G \subseteq A$ contains all atoms that must be true in a

[1] http://www2.parc.com/isl/members/syoon/obtusewedge.html.

state in order to satisfy the goal, i.e., $s_G \subseteq A$ is a goal state iff $G \subseteq s_G$. The finite set $O$ contains STRIPS actions $o$ of the form

$$o = (pre(o), del(o), add(o)) \quad (1)$$

which consist of three sets

$$pre(o), del(o), add(o) \subseteq A, \text{ with } del(o) \cap add(o) = \varnothing. \quad (2)$$

The precondition $pre(o)$ contains atoms that need to be *true* for the corresponding action to be executable. The delete list $del(o)$ and the add list $add(o)$ describe the action's effects on a state, i.e., action $o$ applied to state $s \subseteq A$ results in a state

$$s' = (s \setminus del(o)) \cup add(o). \quad (3)$$

A planning problem can be solved by searching the space of planning states, starting from the initial state. This results in a search tree or graph, where nodes correspond to planning states and edges to planning actions. Plans are paths within the graph. A path from the initial state $I$ to a state conforming to the goal representation $G$ is a solution to the planning problem. It is possible that multiple paths lead from the initial to a goal state. The shortest path corresponds to the optimal solution.

The most trivial planning algorithm would simply use a breadth-first search to find a solution. Heuristics try to guide the search in order to find a goal state faster by avoiding unpromising states. One way to achieve this is to develop heuristic functions that take a state as input and output the estimated distance to a goal state. A perfect heuristic would assign the shortest distance to a goal for each state, which could be used to implement a perfect planner. However, the computation of such a perfect heuristic is similarly hard to the planning problem itself. Therefore, in order to compute a heuristic value for an intermediate state $s$, many heuristics construct a relaxed planning problem $P' = (A', O', s', G')$ of the original planning problem $P = (A, O, I, G)$, where $s$ (which may be slightly altered to $s'$ due to the relaxation process) is used as initial state. The length of the solution plan to the relaxed problem or approximations thereof are used as heuristic value for state $s$. The following two well known planning algorithms use such an approach. In particular the second one is interesting, because it is used as benchmark for the adaptive heuristics based on machine learning, which are presented later.

(i) HSP (Bonet and Geffner 2000, 2001) is a planner based on the ideas of heuristic search. It generates a relaxed problem $P'$ by ignoring all actions' delete lists. Hence, actions in $P'$ only add but do not delete atoms. Let $h^+$ be a function which outputs the minimum cost from a state to a goal state in $P'$. This function would suit well as heuristic for states of the original planning problem $P$. It would be an *admissible* heuristic for $P$. An admissible heuristic guarantees to find an optimal solution when used in combination with certain search algorithms like $A^*$. However, as the computation of $h^+$ is still NP-hard (Bylander 1994), the actual heuristic $h$ of HSP is an estimate of $h^+$. As further simplification, all subgoals $p \in G$ are assumed to be independent and the costs to achieve them from the initial state $s$ are computed separately. The function $g_s(p)$ estimates the number of actions to achieve subgoal $p$ from state $s$:

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ min_{op \in O(p)}[1 + g_s(Prec(op))] & \text{otherwise} \end{cases} \quad (4)$$

The expression $O(p)$ stands for actions $op$ that add $p$, i.e., $p \in add(op)$. The actual heuristic $h$ is defined by adding up these estimations for all subgoals

$$h(s) = \sum_{p \in G} g_s(p). \quad (5)$$

The resulting heuristic is not admissible and can overestimate the costs, as each subgoal is assumed to be independent. There exists an admissible flavor of the HSP heuristic based on using the max of the subgoal cost estimates instead of the sum. However, the max heuristic is often less informative and proved less useful in planning.

The HSP planner uses a hill-climbing search together with the heuristic described above. Since this search strategy is incomplete, i.e., does not guarantee to find a solution to solvable problems, the successor HSP2 uses a best-first search. It weights states by an evaluation function $f(s) = g(s) + W\, h(s)$, where $g(n)$ is the accumulated cost, h(s) is the estimated cost to the goal, an $W \geq 1$ is a constant. For $W = 1$, HSP2 performs an $A^*$ search (Nilsson 1982), a weighted $A^*$ search (Ebendt and Drechsler 2009) for $W > 1$. Higher values for $W$ usually lead to the goal more quickly but with solutions of lower quality.

(ii) FF (Hoffmann 2001) is a successful planner inspired by HSP. For generating heuristic estimates it also relaxes $P$ to $P'$ by ignoring delete lists of all operators. However, FF extracts an explicit solution to $P'$ rather than estimating the solution length. This extraction is based on the Graphplan (Blum and Furst 1995) planner. The FF heuristic counts the number of actions of the extracted plan. These estimations are usually lower than the HSP heuristic as it is accounted for actions which help to achieve more than only one atom. A relaxed planning graph is built until a layer is reached that contains all goals. Then, the plan is extracted from the graph. Starting from the last layer, at each step some actions from layer $i - 1$ are selected which are needed to achieve subgoals and preconditions from formerly added actions. As search algorithm, FF uses an enforced version of hill-climbing. Given a state $s$, enforced hill-climbing (EHC) uses a full breadth-first search until a state $s'$ is found with better heuristic value than $s$. The actions leading from $s$ to $s'$ are added to the plan and the search continues to evaluate the successors of $s'$. This is done until a solution is found. To further improve planning performance, FF only considers *helpful actions* during search. These are actions which seem most promising in the current state and are also computed using the planning graph. The set $H(s)$ of helpful actions of a state $s$ is extracted as follows:

$$H(s) = \{o | pre(o) \subseteq s, add(o) \cap g_1 \neq \emptyset\}.$$

The set $g_1$ represents the set of goals constructed by relaxed plan extraction at level 1. Actions are considered helpful if they achieve at least one goal at the lowest layer of the relaxed solution. While usage of helpful actions improves planning performance it destroys the completeness of the search. Therefore, if EHC fails a complete search considering all actions is employed. Since the FF heuristic is a non-admissible heuristic like the HSP heuristic, FF is a non-optimal planner as well.

## 3 Related work

This section discusses some selected recent approaches aiming at using machine learning techniques to improve planning. For a more extensive overview we refer to Zimmerman and Kambhampati (2003), Frank (2007), and Fern et al. (2011). One way to use experience gathered during the solution of planning problems is to identify useful macro-actions consisting of an ordered sequence of actions. The idea is to group actions that often occur together in one macro-action. While many older approaches are based on domain analysis, more recent work considers learning of macro-actions from solution plans. Macro-FF (Botea et al. 2005) extends FF by generating macro-actions. It first analyzes a domain and extracts structural information, then it generates macro-operators. The most useful macro-operators are selected by a filtering process. Macro-FF implements two methods for producing macro-operators. The first produces macro-operators from problem formulations and training problems. The generated macro-operators are added to the initial domain, and can be used as input to any planner understanding the planning domain definition language PDDL (Ghallab et al. 1998). The second approach extracts macros from solutions of training problems. These can be used by planners with capabilities to handle macros. Marvin (Coles and Smith 2007) extends the search behavior of FF amongst others by applying a new search strategy called least-bad-first for exploring plateaus and uses plateau-escaping macro-actions. The latter are learned from previous searches of similar plateaus and can be applied in the same way as atomic actions to traverse plateaus in one step. Xu et al. (2009) use weighted rules to make rule-based control of planning more robust. They attempt to learn sets of weighted action-selection rules that are used to assign numeric scores to potential state transitions. These scores are used to guide the search whereby information from multiple rules may be combined. The learning approach itself is based on a combination of a heuristic rule learning and a boosting algorithm called RankBoost.

The probably most similar approach to the one proposed in this work is described by Yoon et al. (2006, 2008) and implemented in the OW planner. In addition to heuristic function learning OW also learns reactive policies. Both learning approaches are based on a relaxed-plan feature space represented in taxonomic syntax, which provides a language for describing sets of objects with common properties. For each search node encountered a database consisting of taxonomic syntax expressions is constructed that includes, amongst others, information about goals, current state and relaxed plan features, e.g., add/delete lists. Taxonomic syntax allows to model more complex features based on these database information. This defines the feature space heuristic functions and reactive policies are learned upon. Heuristic function learning is based on linear regression and aims at augmenting the FF heuristic. The learned heuristic function is represented as weighted linear combination of complex features as described above. The aggregation of the FF heuristic and the regression function is used as heuristic. The training task is to learn a linear function that compensates for misestimations of the FF heuristic. Yoon et al. apply a feature selection procedure trying to identify informative relaxed plan properties. The second approach implemented by OW for incorporating control knowledge into planning are policies or rules that represent a mapping of planning states to actions. Such rules help to reactively select actions during planning as they map states to actions. They have the potential to greatly improve planning performance because if directly applied no search is necessary. However, as such rules tend to be imperfect, their application to the greedy selection of actions may cause planning failures. To overcome this issue Yoon et al. combine learned policies with heuristic search. At each node expansion, in addition to the successors of the currently expanded node, the states encountered by following the policy for some horizon are additionally added to the search queue.

## 4 Adaptive search heuristics

In contrast to the approach of Yoon et al. that uses a trained least squares linear regression model in combination with the FF heuristic for computation of estimates, in this work we investigate ANN and SVM not only to augment the FF heuristic but in particular to generate heuristics that can be used independently from the FF heuristic. Purely learned heuristics have the potential to absorb relevant information and to provide heuristic estimates efficiently, without solving relaxed planning problems. In particular, given a planning problem $P$, the FF heuristic computes an estimate in time polynomial to the length of the longest add list of any action, to the number of actions, and to the number of atoms contained in the initial state; all with respect to $P'$, the relaxed problem of $P$. The heuristics proposed in this work compute an estimate in time linear to the number of atoms. The number of actions in planning is often orders of magnitude larger than the number of atoms.

### 4.1 Learning task

For the case of building standalone goal-distance estimators which are independent from the FF heuristic, the heuristic function $h$ is set to a function $f$ which is to be learned:

$$h(s) \mapsto f(s). \tag{6}$$

Learning is based on training data of the form $\{s_i, d_i\}_{i=1}^{n}$, mapping states $s_i \in S$ to their goal distance $d_i \in \mathbb{N}_0$ for a given domain. Such training data can be extracted from a plan. Consider a planning problem of the form $P = (A, O, I, G)$ and a plan $p$ of length $l$ solving this problem. This can be easily transformed into a representation $\{s_i, d_i\}_{i=1}^{n}$ that maps states $s_i \in S$ to their goal state distance $d_i \in \mathbb{N}_0$. A specific tuple $(I, p)$ maps to $(I, l), (s_1, l-1)\ldots, (s_G, 0)$, where $s_G$ is a state conforming to $G$.

For the case of augmenting FF, the heuristic function $h$ is set to the output of the FF heuristic, aggregated with a function $\Delta$ that has to be learned:

$$h(s) \mapsto \mathrm{FF(s)} + \Delta(\mathrm{s}). \tag{7}$$

For the task of learning a function $\Delta$ to compensate for misestimations of FF, training data of the same form is used. But a specific planning problem and solution plan as above is transformed to the training instances $(I, l - \mathrm{FF}(I)), (s_1, (l-1) - \mathrm{FF}(s_1))\ldots, (s_G, 0 - \mathrm{FF}(s_G))$. Hence, the training data consists of states mapped to the error of the FF heuristic.

### 4.2 Regression algorithms

In the following, the algorithms we have considered for heuristic function learning are reviewed. We assume a set of training instances $\{s_i, d_i\}_{i=1}^{n}$. Any state $s$ of a domain based on atoms $A$ can be described as a binary vector $s = (a_1, \ldots, a_p)$ where $p$ is the size of set $A$.

#### 4.2.1 *Multi-layer perceptron*

The ANN method we use is a multi-layer perceptron (MLP) (Rumelhart et al. 1988). A neuron takes a real-valued vector as input, calculates a linear combination, and puts the output through some activation function $\varphi$.

$$y = \varphi\left(\sum_{i=1}^{m}(\omega_i x_i) + w_0\right) \tag{8}$$

A MLP uses multiple layers of neurons (nodes) and is able to distinguish data that is not linearly separable. In particular, we use a feed-forward network consisting of three fully connected layers: one input layer, one hidden layer and one output layer. The input layer consists of $p$ nodes, and each node represents one atom. The hidden layer contains $\lceil \frac{p}{2} \rceil$ nodes, the output layer contains a single node. Nodes of the input layer are passive and simply relay the inputs to the outputs. A sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is used as activation function for the hidden layer's nodes. Last, the single node of the output layer outputs the predicted distance and no function is used to further change the weighted sum of the inputs, i.e., the activation function is set to the identity function $f(x) = x$. Figure 1 shows an exemplary MLP network for a domain based on an atom set $A = \{a_1, a_2, a_3, a_4, a_5\}$.

The MLP learning is based on the WEKA (Hall et al. 2009) tool, which uses the well-known Backpropagation algorithm (Rumelhart et al. 1988; Widrow and Hoff 1960). This algorithm adjusts the weights of neurons by propagating training instances through the network, comparing the network's output of a particular instance to its label, and back-propagation of the resulting error. Based on the error the perceptrons' weights are modified. Important parameters that affect the Backpropagation algorithm are the learning rate and the momentum. The learning rate influences the extent changes are made to the weights. If the learning rate is set too large, then the algorithm may easily fail the optimum while a small learning rate slows down the learning process. We use a learning rate of 0.3 which is the default of the WEKA tool. Another important constant is the momentum, which helps to avoid the negative effects of too large learning rates by giving stability to the search for the weights resulting in minimal error. The momentum is set to 0.2, which is also the WEKA default.

#### 4.2.2 *Support vector regression*

SVMs (Cortes and Vapnik 1995; Vapnik 1995) find a hyperplane that separates training instances belonging to different classes. Hereby, SVMs try to maximize the

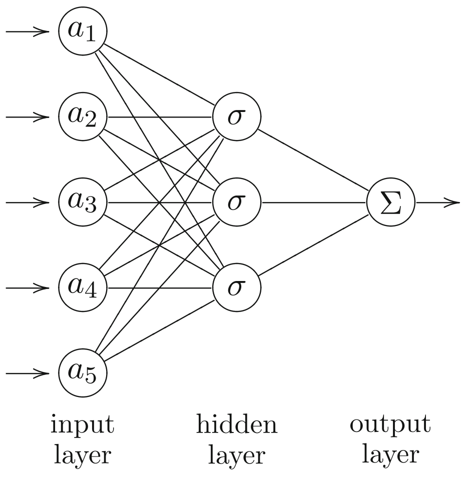

**Fig. 1** Neural network architecture for a domain consisting of five atoms $a_1, \ldots, a_5$. We use a fully interconnected feed-forward network with one hidden layer. The input layer contains one passive node for each atom. The hidden layer consists of $\lceil \frac{5}{2} \rceil = 3$ nodes with sigmoid activation function, while the single output node does not postprocess the weighted sum of the inputs, and outputs the distance estimation. The values in the input to this MLP are binary vectors that represent states

distance of the instances closest to the hyperplane. These instances are called support vectors. SVMs use the *kernel trick* in order to separate data that are not linearly separable. The basic idea is to map instances into a higher dimensional space where the data become linearly separable. Additionally, *slack variables*, which allow but penalize misclassified training instances, are often used to avoid overfitting. SVMs can also be used for regression. The basic idea of support vector regression (SVR) (Drucker et al. 1996) is to find a function that has at most $\varepsilon$ deviation from the actually observed values for the training data, and at the same time is as flat as possible. Similar to classification, the introduction of slack variables may loosen the strict compliance with the maximum deviation defined by $\varepsilon$. SVR learning of a heuristic function $h$ can be expressed by the following optimization problem:

$$\text{Maximize} \quad \begin{cases} \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(s_i, s_j) \\ -\varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} d_i(\alpha_i - \alpha_i^*) \end{cases} \tag{9}$$

$$\text{subject to} \quad \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \text{ and } \alpha_i, \alpha_i^* \in [0, C] \tag{10}$$

$$\text{where } h(s) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)k(s_i, s) + b \tag{11}$$

with Lagrange multipliers $\alpha_i$ and $\alpha_i$*. The SMO algorithm presented in Shevade et al. (1999) is used for solving the above optimization problem, and for computing constant $b$. The SVR is based on WEKA and the defaults it suggests are used for setting the parameters $C$ and the kernel function $k$.

## 5 Evaluation

The evaluation described in this section provides a comparison of standalone adaptive heuristics based on MLP and SVR to the classical FF heuristic and the learned heuristic implemented in the OW planner. Furthermore it is analyzed how the machine learning methods perform when used to augment the FF heuristic, instead of using them for generating standalone heuristics. For evaluation, we use problem domains provided by the learning track of the 6th International Planning Competition (IPC-6) in 2008: Matching Blocksworld, Sokoban, Gold Miner, N-Puzzle, Parking, and Thoughtful. For the first five domains, problem generators, which allow a random generation of planning problems, are available. Therefore, only these domains have been considered. In the following, results for the Matching Blocksworld, Sokoban, Gold Miner, and N-Puzzle domains are presented. The Parking domain has been also omitted, because the oracle we use for evaluation, as explained in the next paragraph, is not able to solve the problems outputted by the Parking domain generator, but often exits with the error message "Initial state is a dead end. Completely explored state space—no solution!", even though the domain generator claims to produce only solvable problem instances.

For evaluation of a heuristic's accuracy we use an "oracle" that is able to tell for any state the true goal distance. This oracle is based on the Fast Downward planner (Helmert 2006) with the LM-cut heuristic (Helmert and Domshlak 2009). During the evaluation phase the planning heuristics' output is compared to the output of the oracle for measuring their accuracy. Figure 2 shows the environment used for conducting experiments. The generation of the training data is also based on the oracle. Each instance of a training data set is generated by creating a random state using a planning problem generator and asking the oracle for a label, i.e., the true minimum goal distance for the state. Every optimal plan of length $l$ can be used to generate $l + 1$ noise-free training instances. The plans found by non-optimal planners are not guaranteed to be optimal with respect to plan length. Thus, when using non-optimal planners for generating training instances, some degree of noise is added to the training data if the planner outputs a non-optimal plan. Misestimations during the planning process, however, do not necessarily add noise to some of the training data instances. This is only the case if the misestimations also materialize in a plan that has non-optimal length. The whole evaluation framework

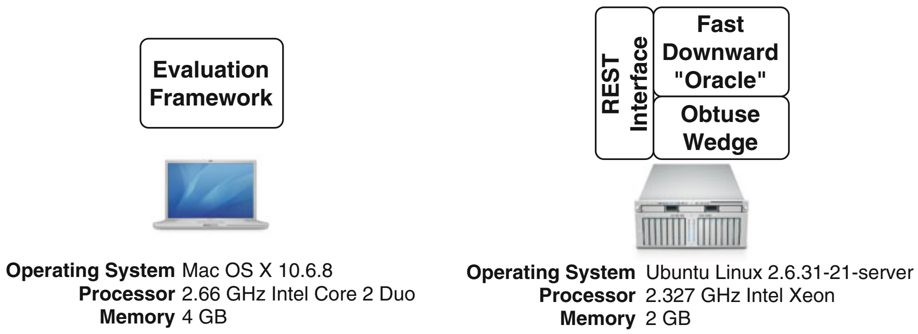
Evaluation Framework
REST Interface
Fast Downward "Oracle"
Obtuse Wedge
Operating System Mac OS X 10.6.8
Processor 2.66 GHz Intel Core 2 Duo
Memory 4 GB
Operating System Ubuntu Linux 2.6.31-21-server
Processor 2.327 GHz Intel Xeon
Memory 2 GB

**Fig. 2** The evaluation environment used for conducting experiments

including all heuristics and machine learning techniques is Java-based and is running on Mac OS X. Since Fast Downward and OW, could not be easily ported to Mac OS X they run on a Linux machine and are made accessible through a REST interface.

The generation of the training set is illustrated in Algorithm 1. A problem generator $G_D$ generating random problems for domain $D$ and the resulting number of training instances are given as input to the procedure. The generation of training data for adaptive heuristics augmenting the FF heuristic is slightly different, i.e., in line 5 not $\{I, d\}$ but $\{I, d - FF(I)\}$ is added to $T$. Algorithm 2 shows how the actual evaluation is conducted. The evaluation procedure takes a heuristic function $h_e$ which is to be evaluated. Adaptive heuristics are first trained using training data set $T$. For evaluation purposes the time it takes to build the heuristic $h_e$ is measured. Then, a series of 100 random problems is generated using the random problem generator $G_D$. The initial state $I$ of this planning problem is fed to the heuristic $h_e$ providing a goal distance estimation and to the oracle which outputs the exact goal distance. For evaluation purposes, the time it takes to generate the estimation is measured and the estimated goal distance as well as correct distance are logged for evaluating the accuracy of the heuristic. Based on the measurements during the evaluation phase (Algorithm 2, lines 1, 4, and 6) we calculate metrics for accuracy and speed of the heuristics. To quantify the estimation accuracy of a heuristic $h_e$ we calculate the root mean square error (RMSE) and the mean absolute error (MAE) of the estimated goal distances $d_e^i$ compared to the true goal distance $d^i$ for each evaluation round $i$ (cf. Algorithm 2, line 6):

$$RMSE = \sqrt{\frac{1}{100}\sum_{i=1}^{100}\left(d_e^i - d^i\right)^2}, \tag{12}$$

$$MAE = \frac{1}{100}\sum_{i=1}^{100}\left|d_e^i - d^i\right|. \tag{13}$$

**Algorithm 1** GenerateTrainingSet($G_D$, $n$)

```
Input: random problem generator $G_D$, number $n$
1: $T = \varnothing$
2: for $i = 1 \rightarrow n$ do
3:     $p_r = (A, O, I, G) \leftarrow G_D.getNext()$
4:     $d \leftarrow$ initial state I's distance to goal according to oracle
5:     $T \leftarrow T \cup \{I, d\}$
6: end for
7: return $T$
```

**Algorithm 2** Evaluate($h_e$, $T$, $G_D$)

```
Input: heuristic $h_e$, training set $T$, random problem generator $G_D$
1: $h_e$.train(T) {measure time, only applicable for adaptive heuristics}
2: for $i = 1 \rightarrow 100$ do
3:     $p_r = (A, O, I, G) \leftarrow G_D.getNext()$
4:     $d_e = h_e(I)$ {measure time}
5:     $d$ = distance according to oracle
6:     {log $d_e$ and $d$}
7: end for
8: return $T$
```

Since the ordering of states by heuristic functions for planning is much more important than correct absolute values we scale the heuristic values in order to avoid taking such errors into account. Especially FF and OW profit from this approach because they are both very much out of scale compared to the actual goal distances. Heuristics based on ANN and SVM do not profit from that preprocessing at all. In order to give information about the execution times of different heuristics we calculate the mean, median, minimum, and maximum of the time the heuristic takes to generate an estimate in milliseconds (cf. Algorithm 2, line 4). For the adaptive heuristics, we also provide the training time (cf. Algorithm 2, line 1).

## 5.1 Matching Blocks World

First, the domain description of the Matching Blocks World domain from IPC-6 is provided.

> This is a simple variant of the blocks world where each block is either positive or negative and there are two hands, one positive and one negative. The twist is that if a block is picked up by a hand of opposite polarity then it is damaged such that no other block can be placed on it, which can lead to dead ends. The interaction between hands and blocks of the same polarity is just as in the standard blocks world.

The evaluation of the Matching Blocks World domain is based on worlds of different sizes, i.e., different numbers of blocks, and the goal is set to a state where the blocks form a sorted tower. We use domains with 5, 6, 7, and 8 blocks to give information about the scalability of our approach. The particular values have been chosen because the oracle is fast enough to solve problems of that size quickly, while for problems with 9 blocks it may take the oracle minutes to solve a single problem. The size of the training data is adjusted to the size of the problem. By default, we train the heuristics for the 5 blocks world with 100 instances, the 6 blocks world with 200 instances, the 7 blocks world with 300 instances, and the 8 blocks world with 400 instances. An instance is not a plan with a solution (containing multiple state/goal distance pairs), but a single state with its corresponding goal distance. We also investigate how modifying the number of training instances affects the prediction accuracy. However, to start, the results provided in Table 1 are based on heuristics with the stated training set sizes. The approach for evaluating the heuristic learning in OW is slightly different. Since OW is able to generate heuristics that can be applied to any problem size we only train it once using 517 training instances.

The values in Table 1 represent the errors of generating heuristic estimates for the FF heuristic (FF), the heuristic used by the OW planner (OW), the standalone MLP heuristic (MLP), the FF heuristic augmented with MLP (FF + MLP), the standalone SVM heuristic (SVM), and the FF heuristic augmented with SVM (FF + SVM), according to the metrics RMSE (cf. Eq. 12) and MAE (cf. Eq. 13). Lower values represent better values and the best values are highlighted in bold. The accuracy of adaptive heuristics is superior to the FF heuristic and the OW heuristic in all cases. OW tends to perform slightly better than FF. The combination of FF with the adaptive heuristics does not provide significantly better results than the standalone heuristics; it rather decreases accuracy in many cases. The MLP based heuristics work slightly better for small Matching Blocks Worlds while the heuristics based on SVMs have an edge on the larger worlds. However, the differences between MLP and SVM are small and may be partly caused by random effects.

Table 2 shows the execution times of the heuristics in milliseconds. Mean, median, min, max stand for the respective values of the execution time for a single call of the heuristic, i.e., how long it takes to generate a heuristic estimate. OW, FF + MLP and FF + SVM are excluded because they generate a heuristic value based on FF. For MLP + FF and SVM + FF the execution time is basically exactly the sum of the FF heuristic and the MLP and SVM heuristic, respectively. For OW there was no simple way to measure the runtime of the single steps that contribute to the overall execution time. Also, since it runs on another hardware the running times would not be directly comparable. The table shows that the adaptive heuristics are able to generate estimates a magnitude faster than the FF heuristic—the SVM heuristic is the fastest one. However, during initialization the adaptive heuristics have to be trained and the respective training times are shown in the Training row.

**Table 1** Errors of heuristic estimates for random states of the Matching Blocks World

| | FF | OW | MLP | FF + MLP | SVM | FF + SVM |
|---|---|---|---|---|---|---|
| 5 | | | | | | |
| RMSE | 2.25 | 1.92 | **0.56** | 0.71 | −0.97 | 0.97 |
| MAE | 1.77 | 1.50 | **0.34** | 0.42 | 0.38 | 0.57 |
| 6 | | | | | | |
| RMSE | 2.48 | 2.39 | **0.82** | 1.25 | 1.23 | 1.36 |
| MAE | 1.97 | 1.96 | **0.51** | 0.94 | 0.88 | 1.06 |
| 7 | | | | | | |
| RMSE | 2.50 | 2.63 | 1.37 | 1.51 | 1.50 | **1.32** |
| MAE | 1.97 | 2.21 | 0.97 | 1.20 | 1.09 | **0.95** |
| 8 | | | | | | |
| RMSE | 2.90 | 2.80 | 1.85 | 2.05 | **1.72** | 1.73 |
| MAE | 2.30 | 2.32 | 1.34 | 1.44 | **1.23** | 1.27 |

The size of the world varies from 5 to 8 blocks. Lower values represent a better estimation accuracy. The best values are highlighted in bold

**Table 2** Mean/median/min/max values for the time it takes to generate a single heuristic estimate for states of the Matching Blocks World

| | FF | MLP | SVM |
|---|---|---|---|
| 5 | | | |
| Mean/median | 7.98/7.99 | 0.26/0.24 | 0.16/0.15 |
| Min/Max | 2.30/15.99 | 0.18/0.70 | 0.10/0.55 |
| Training | × | 217.00 | 31.00 |
| 6 | | | |
| Mean/median | 15.40/16.11 | 0.27/0.27 | 0.18/0.17 |
| Min/max | 6.39/22.84 | 0.24/0.82 | 0.14/0.64 |
| Training | × | 2353.00 | 161.00 |
| 7 | | | |
| Mean/median | 25.74/26.36 | 0.32/0.31 | 0.18/0.17 |
| Min/max | 9.20/47.42 | 0.29/0.86 | 0.13/0.88 |
| Training | × | 3376.00 | 591.00 |
| 8 | | | |
| Mean/median | 39.55/41.50 | 0.39/0.32 | 0.27/0.18 |
| Min/max | 19.68/51.90 | 0.28/3.81 | 0.15/3.27 |
| Training | × | 3594.00 | 1917.00 |

Training refers to the time it takes to train the corresponding adaptive heuristic, which is performed once during initialization. The size of the world varies from 5 to 8 blocks. Lower values are better. All values are milliseconds, except for the training time of OW

OW training: 31.6 h (517 instances)

The training time increases with the size of the domain and the size of the training instances. For the world with 5 blocks we use 100 training instances, 200 for the 6 blocks world, 300 for the 7 blocks world, and 400 for the 8 blocks world. The SVM heuristic has shorter training times than the MLP heuristic; the relative difference seems to decrease for larger problem instances. The training time for the OW planner is extremely long compared to the heuristics based on ANNs and SVMs. It takes almost 32 h to train the planner using 517 instances. For OW the number of instances used for training cannot be exactly defined; we have used a target training instance number of 500 for OW for Matching Blocks World and all subsequent domains.

Figure 3 illustrates the influence of the training set size on the estimation error, i.e., the RMSE. The FF heuristic does not depend on the training set size and for OW we have only used one training sample size of 517. For the adaptive heuristics the experiments are based on different training set sizes of 25, 50, 75, and 100 % regarding the initial training set sizes (i.e., 100 for 5 blocks world, 200 for 6 blocks world, 300 for 7 blocks world, and 400 for 8 blocks world). The figures show that already with few training instances the adaptive heuristics provide better results than the FF heuristic and the OW heuristic. The MLP heuristics benefit more clearly from additional training data.

## 5.2 Sokoban

The domain description of Sokoban provided by IPC-6 reads as follows: "This domain is inspired by the popular Sokoban puzzle game where an agent has the goal of pushing a set of boxes into specified goal locations in a grid with walls."

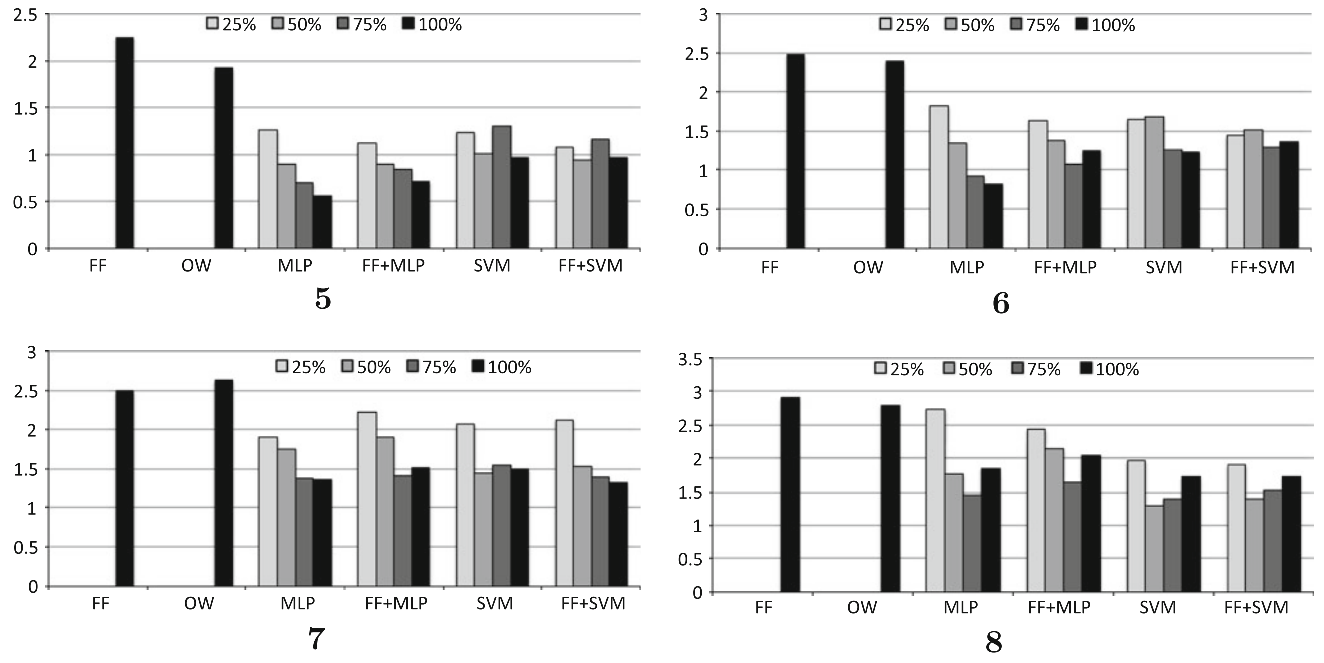

**Fig. 3** Root square mean errors of heuristic estimates for different training set sizes of 25, 50, 75, and 100 % for the Matching Blocks world with 5 blocks (*top-left*), 6 blocks (*top-right*), 7 blocks (*bottom-left*), and 8 blocks (*bottom-right*)

**Table 3** Errors of heuristic estimates for random states of the Sokoban domain

| | FF | OW | MLP | FF + MLP | SVM | FF + SVM |
|---|---|---|---|---|---|---|
| 6 × 6, 1 | | | | | | |
| RMSE | 2.40 | × | 1.64 | 2.78 | **1.24** | 2.87 |
| MAE | 1.95 | × | 1.22 | 2.15 | **0.87** | 2.20 |
| 7 × 7, 1 | | | | | | |
| RMSE | 2.23 | × | **1.97** | 4.79 | **1.97** | 3.34 |
| MAE | 1.85 | × | 1.07 | 3.99 | **1.04** | 2.63 |
| 7 × 7, 2 | | | | | | |
| RMSE | 3.49 | × | 3.03 | 3.81 | **2.22** | 3.11 |
| MAE | 2.63 | × | 2.31 | 2.90 | **1.49** | 2.26 |
| 8 × 8, 2 | | | | | | |
| RMSE | 4.64 | × | 4.82 | 4.57 | **3.05** | 4.98 |
| MAE | 3.32 | × | 3.88 | 3.57 | **2.37** | 3.60 |

The size of the world varies from a 6 × 6 grid with 1 box to an 8 × 8 grid with 2 boxes. Lower values represent a better estimation accuracy. The best values are highlighted in bold. OW could not be considered because training did not finish in reasonable amount of time

For evaluation, again four different problem sizes have been considered: a domain with a 6 × 6 grid and one box, a domain with a 7 × 7 grid and one box, a domain with a 7 × 7 grid and two boxes, and a domain with an 8 × 8 grid and two boxes. Each domain has four wall elements. The size of the training data is again adjusted to the size of the problem, in the same way as for the Matching Blocks World: 100 instances for the smallest world and 400 instances for the biggest one. Again the actual dimensions have been chosen according to the capabilities of the oracle.

Table 3 presents the errors of heuristic estimates for the Sokoban domain. As before, the adaptive heuristics always provide better results than the FF heuristic. The OW heuristic could not be considered because the training based on 500 target instances did not finish within 100 h and was cancelled. For every experimental setting no heuristic is able to beat the SVM heuristic in terms of accuracy. This reflects the theoretical superiority of SVMs over MLP: SVMs do not suffer from multiple local minima, while backpropagation usually converges only to locally optimal solutions. Another advantage of SVMs is that they automatically adjust their model size by selecting a number of support vectors. The combination of FF with adaptive policies tends to decrease the goal distance estimation quality. Also for the execution times, shown in Table 4, the MLP and SVM heuristics clearly outperform the FF heuristic. Both adaptive heuristics are able to generate estimates two orders of magnitudes faster than the FF heuristic, while SVM is faster than MLP. Similar to the previous domain, SVM again needs less time for training than MLP; the relative training speed difference is again higher for small problem instances and seems to converge for larger instances. Figure 4, which illustrates the influence of the training set size on the estimation error in the same way as for the Matching Block world, shows that in most cases, i.e., heuristics based on different numbers of training instances, the standalone adaptive heuristics outperform FF while the heuristics aiming to augment FF actually tend to deteriorate its accuracy.

**Table 4** Mean/median/min/max values for the time it takes to generate a single heuristic estimate for states of Sokoban

| | FF | MLP | SVM |
|---|---|---|---|
| 6 × 6, 1 | | | |
| Mean/median | 35.55/37.79 | 0.40/0.38 | 0.20/0.19 |
| Min/max | 8.64/63.94 | 0.31/1.26 | 0.14/1.07 |
| Training | × | 4289.00 | 97.00 |
| 7 × 7, 1 | | | |
| Mean/median | 58.59/54.78 | 0.69/0.52 | 0.27/0.24 |
| Min/max | 28.55/136.84 | 0.44/8.10 | 0.17/1.50 |
| Training | × | 1185.00 | 182.00 |
| 7 × 7, 2 | | | |
| Mean/median | 135.61/127.50 | 0.73/0.65 | 0.50/0.36 |
| Min/max | 62.99/321.75 | 0.55/3.84 | 0.23/3.88 |
| Training | × | 21683.00 | 1651.00 |
| 8 × 8, 2 | | | |
| Mean/median | 329.00/329.68 | 1.31/0.95 | 0.69/0.42 |
| Min/max | 120.28/701.66 | 0.86/5.66 | 0.35/6.62 |
| Training | × | 6897.00 | 2248.00 |

Training refers to the time it takes to train the corresponding adaptive heuristic, which is performed once during initialization. Lower values are better. All values are milliseconds, except for OW, for which training did not finish within 100 h and was cancelled

OW training: not finished after 100 h (500 instances)

## 5.3 Gold Miner

The following text passage quotes the description of the Gold Miner domain as provided by IPC-6:

> A robot is in a mine and has the goal of reaching a location that contains gold. The mine is organized as a grid with each cell either being hard or soft rock.

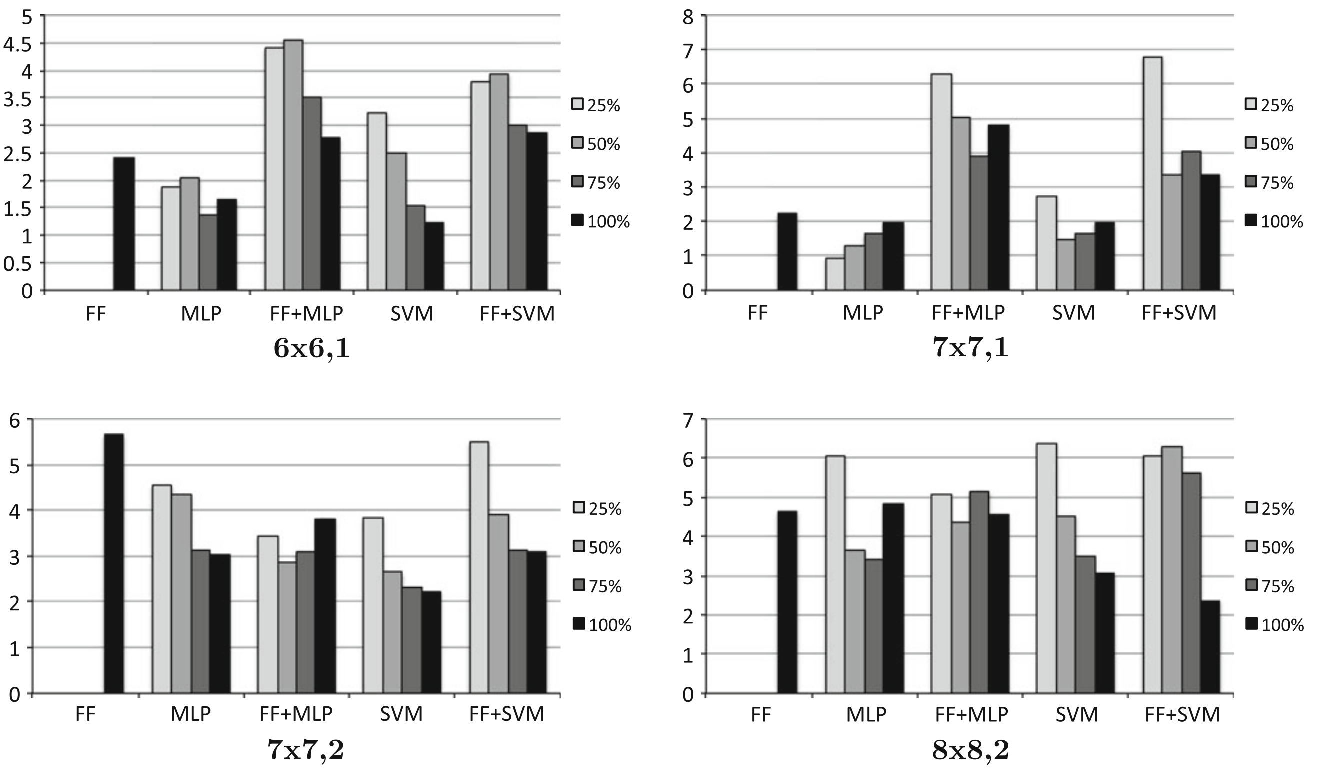

**Fig. 4** Root square mean errors of heuristic estimates for different training set sizes of 25, 50, 75, and 100 % for the Sokoban domain with a 6 × 6 grid and 1 box (*top-left*), a 7 × 7 grid with 1 box (*top-right*), a 7 × 7 grid with 2 boxes (*bottom-left*), and an 8 × 8 grid with 2 boxes (*bottom-right*)

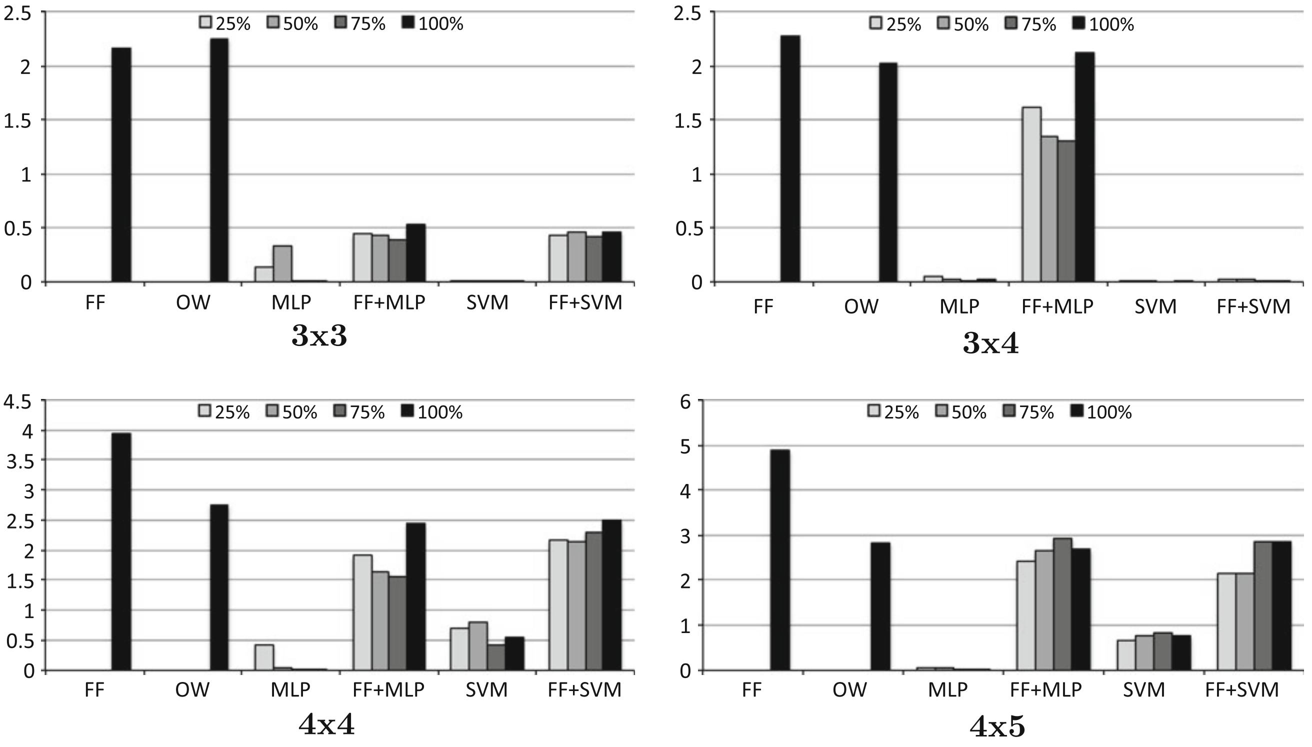

**Fig. 5** Root square mean errors of heuristic estimates for different training set sizes of 25, 50, 75, and 100 % for the Gold Miner domain with a 3 × 3 grid (*top-left*), a 3 × 4 grid (*top-right*), a 4 × 4 grid (*bottom-left*), and a 4 × 5 grid (*bottom-right*)

There is a special location where the robot can either pickup an endless supply of bombs or pickup a laser cannon. The laser cannon can shoot through both hard and soft rock, whereas the bomb can only penetrate soft rock. However, the laser cannon also will destroy the gold if used to uncover the gold location. The bomb will not destroy the gold. The problem difficulty is scaled by increasing the size of the grid. This domain has a simple optimal strategy: (1) get the laser cannon, (2) shoot through the rock until reaching a cell bordering the gold, (3) go and get a bomb, (4) blast away the rock at the gold location, (5) pickup the gold.

The four different world sizes for the Gold Miner domain are: a $3 \times 3$ grid, a $3 \times 4$ grid, a $4 \times 4$ grid, and a $4 \times 5$ grid (Fig. 5). For this domain the standalone adaptive heuristics can almost optimally predict goal distance estimates, as shown in Table 5, and thus seem to have absorbed the optimal strategy from the training data. OW improves over FF in most cases but is still far from the performance of the standalone heuristics. Combinations with FF tend to deteriorate the goal distance estimation accuracy compared to the standalone heuristics. For the problem instances of size $3 \times 3$ and $3 \times 4$, MLP and SVM are basically equally accurate and differences can be attributed to random effects. For the two larger problem instances MLP outperforms SVM in terms of accuracy, which can be attributed to different stopping conditions and longer training times of MLP, as presented in Table 6. Again, the training time of OW, which is based on 429 training instances, is magnitudes longer for OW compared to MLP and SVM. The table also shows that the execution times for computing estimates for MLP and SVM are again

**Table 5** Errors of heuristic estimates for random states of the Gold Miner domain

| | FF | OW | MLP | FF + MLP | SVM | FF + SVM |
|---|---|---|---|---|---|---|
| $3 \times 3$ | | | | | | |
| RMSE | 2.16 | 2.25 | **0.01** | 0.53 | **0.01** | 0.46 |
| MAE | 1.89 | 1.99 | **0.01** | 0.44 | **0.01** | 0.44 |
| $3 \times 4$ | | | | | | |
| RMSE | 2.28 | 2.02 | 0.02 | 2.12 | **0.01** | **0.01** |
| MAE | 2.06 | 1.81 | 0.02 | 1.48 | **0.01** | **0.01** |
| $4 \times 4$ | | | | | | |
| RMSE | 3.94 | 2.75 | **0.02** | 2.44 | 0.54 | 2.49 |
| MAE | 3.26 | 2.42 | **0.01** | 1.85 | 0.32 | 1.94 |
| $4 \times 5$ | | | | | | |
| RMSE | 4.87 | 2.83 | **0.01** | 2.68 | 0.75 | 2.67 |
| MAE | 3.64 | 2.49 | **0.01** | 2.12 | 0.60 | 1.72 |

The size of the world varies from $3 \times 3$ grid to a $4 \times 5$ grid. Lower values represent a better estimation accuracy. The best values are highlighted in bold

**Table 6** Mean/median/min/max values for the time it takes to generate a single heuristic estimate for states of the Gold Miner domain

| | FF | MLP | SVM |
|---|---|---|---|
| $3 \times 3$ | | | |
| Mean/median | 7.55/7.16 | 0.25/0.24 | 0.16/0.16 |
| Min/max | 4.48/22.68 | 0.19/1.23 | 0.12/0.69 |
| Training | × | 2,143 | 34 |
| $3 \times 4$ | | | |
| Mean/median | 15.47/15.02 | 0.33/0.29 | 0.19/0.19 |
| Min/max | 10.58/21.71 | 0.26/3.17 | 0.14/0.89 |
| Training | × | 1,497 | 29 |
| $4 \times 4$ | | | |
| Mean/median | 25.36/25.91 | 0.45/0.38 | 0.20/0.19 |
| Min/max | 18.35/35.11 | 0.36/3.55 | 0.14/0.75 |
| Training | × | 16,146 | 532 |
| $4 \times 5$ | | | |
| Mean/median | 43.21/42.78 | 0.45/0.38 | 0.23/0.22 |
| Min/max | 38.28/53.29 | 0.33/2.47 | 0.18/0.91 |
| Training | × | 37,567 | 1,676 |

Training refers to the time it takes to train the corresponding adaptive heuristic, which is performed once during initialization. Lower values are better. All values are milliseconds, except for the training time of OW

OW training: 4.4 h (429 instances)

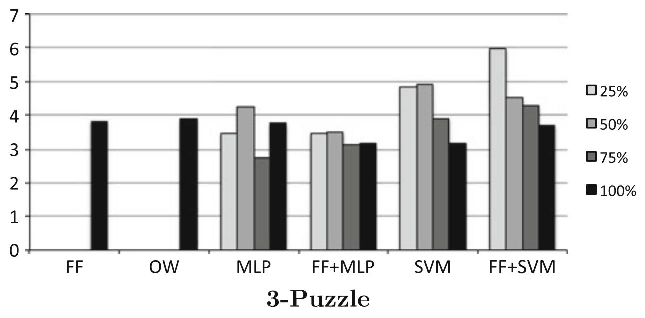

**Fig. 6** Root square mean errors of heuristic estimates for different training set sizes of 25, 50, 75, and 100 % for the 3-Puzzle domain

considerably faster compared to FF, while SVM is faster than MLP for generating estimates. In Fig. 6, which shows the sensitivity of the heuristics related to training size, the values for the standalone heuristics are much better than the values for the combined ones for all training set sizes. Here, more training set instances seem to lead to overfitting and have negative impact on accuracy in many cases.

### 5.4 N-Puzzle

The following text passage quotes the description of the N-Puzzle domain as provided by IPC-6:

> This is the classic N × N sliding puzzle domain. This has been a common testbed for search algorithms and in particular work on macro learning.

The N-Puzzle domain could only be tested for one domain size: In case of N = 2, the generated problems were very easy to solve; i.e., solvable by plans of size 2 or 3. In case of N = 4, the oracle was not able to generate solutions in a reasonable amount of time. Therefore, only N = 3 has been considered. By default, we train the heuristics with 50 instances (545 for OW). Table 7 shows that adaptive heuristics improve the predication accuracy compared to FF. However, the quality improvement is not as high as for previous domains. All adaptive heuristics perform similarly well, while standalone SVM shows the best results in terms of accuracy. OW performs slightly below FF. The adaptive heuristics again win clearly in terms of the time it takes to compute a single heuristic estimate (cf. Table 8). The training time for SVM heuristics is shorter than the MLP training time, but the difference in training duration is smaller compared to experiences in other domains presented above.

**Table 7** Errors of heuristic estimates for random states of the 3-Puzzle domain

| | FF | OW | MLP | FF + MLP | SVM | FF + SVM |
|---|---|---|---|---|---|---|
| 3-Puzzle | | | | | | |
| RMSE | 3.82 | 3.87 | 3.75 | **3.18** | 3.19 | 3.69 |
| MAE | 3.01 | 3.14 | 2.94 | 2.54 | **2.51** | 2.99 |

Lower values represent a better estimation accuracy. The best values are highlighted in bold

**Table 8** Mean/median/min/max values for the time it takes to generate a single heuristic estimate for states of the 3-Puzzle domain

| | FF | MLP | SVM |
|---|---|---|---|
| 3-Puzzle | | | |
| Mean/median | 21.94/20.76 | 0.33/0.3 | 0.22/0.19 |
| Min/max | 3.94/74.43 | 0.28/1.06 | 0.12/2.88 |
| Training | × | 4,713 | 774 |

All values are milliseconds, except for the training time of OW

OW training: 2 h (545 instances)

## 6 Conclusion

In this work we elaborate on how to learn heuristic estimates for state-space non-optimal planning using ANNs and SVMs, two well-known methods from natural computing. All conducted experiments proved that these methods are able to generate standalone heuristics able to significantly outperform the well-known FF heuristic in terms of estimation accuracy. Additionally, the learned heuristics are able to compute estimates magnitudes faster. While both ANNs and SVMs can be used to augment the FF heuristic, standalone heuristics showed better estimation accuracy and lower overhead. Compared to the Obtuse Wedge planner, which also learns heuristics based on solved plans, our standalone heuristics provide superior accuracy, have lower execution time complexity for generating estimates, and can be trained in a fraction of its training time. However, the heuristics presented here are less general and can only be applied to static goals and fixed domains sizes.

In the future we will work on overcoming these two limitations of our approach. For incorporating varying goals, the learning problem can be reformulated. Instead of learning a function $h(s)$ mapping a state $s$ to its distance from the constant goal, a function $h(s, g)$ could be derived, mapping $s$ and goal description $g$ to the corresponding distance. Training examples will be extended from $\{s_i,d_i\}_{i=1}^{n}$ to $\{s_i, g_i, d_i\}_{i=1}^{n}$. As mentioned above, the second limitation of the presented heuristic learning approaches is that they cannot be trained using small problem instances and used for solving larger instances, subsequently. In the future we will work on incorporating such generalization capabilities into our adaptive heuristics, e.g., by selecting relevant features at predicate level rather than directly taking the atoms' values into account.

**Acknowledgments** This work was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

## References

Blum A, Furst M (1995) Fast planning through planning graph analysis. In: IJCAI'95: proceedings of the 14th international joint conference on artificial intelligence, pp 1636–1642

Bonet B, Geffner H (2000) HSP: heuristic search planner. Entry at AIPS-98 planning competition. AI Mag 21(2)

Bonet B, Geffner H (2001) Planning as heuristic search. Artif Intell 129(1–2):5–33

Botea A, Enzenberger M, Müller M, Schaeffer J (2005) Macro-FF: improving AI planning with automatically learned macro-operators. J Artif Intell Res 24(1):581–621

Bylander T (1994) The computational complexity of propositional STRIPS planning. Artif Intell 69:165–204

Coles A, Smith KA (2007) Marvin: a heuristic search planner with online macro-action learning. J Artif Intell Res 28:119–156

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297

Drucker H, Burges CJC, Kaufman L, Smola AJ, Vapnik V (1996) Support vector regression machines. In: Mozer M, Jordan MI, Petsche T (eds) NIPS. MIT Press, Cambridge, pp 155–161

Ebendt R, Drechsler R (2009) Weighted A* search—unifying view and application. Artif Intell 173:1310–1342

Fern A, Khardon R, Tadepalli P (2011) The first learning track of the international planning competition. Mach Learn 84:81–107

Fikes R, Nilsson NJ (1971) STRIPS: a new approach to the application of theorem proving to problem solving. Artif Intell 2(4):189–208

Frank J (2007) Using data mining to enhance automated planning and scheduling. In: Proceedings of the IEEE symposium on computational intelligence and data mining. IEEE, pp 251–260

Ghallab M, Howe A, Knoblock C, McDermott D, Ram A, Veloso M, Weld D, Wilkins D (1998) PDDL—the planning domain definition language. Technical report. Yale Center for Computational Vision and Control

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD Explor 11(1):10–18

Helmert M (2006) The fast downward planning system. J Artif Intell Res 26:191–246

Helmert M, Domshlak C (2009) Landmarks, critical paths and abstractions: what's the difference anyway? In: ICAPS'09: proceedings of the 19th international conference on automated planning and scheduling. AAAI

Hoffmann J (2001) FF: the fast-forward planning system. AI Mag 22:57–62

Nilsson NJ (1982) Principles of artificial intelligence. Springer, Berlin

Rumelhart DE, Hinton GE, Williams RJ (1988) Learning representations by back-propagating errors. In: Neurocomputing: foundations of research, pp 696–699. http://www.nature.com/nature/journal/v323/n6088/abs/323533a0.html

Satzger B, Kramer O (2010) Learning heuristic functions for state-space planning. In: CI'10: proceedings of the 5th international conference on computational intelligence, pp 36–43

Satzger B, Kramer O, Lässig J (2010) Adaptive heuristic estimates for automated planning using regression. In: International conference on artificial intelligence, pp 576–581

Satzger B, Pietzowski A, Trumler W, Ungerer T (2008) Using automated planning for trusted self-organising organic computing systems. In: ATC'08: proceedings of the 5th international conference on autonomic and trusted computing. Springer, pp 60–72

Shevade S, Keerthi S, Bhattacharyya C, Murthy K (1999) Improvements to the SMO algorithm for SVM regression. IEEE Trans Neural Networks. doi:10.1109/72.870050

Swiercz M, Kochanowicz J, Weigele J, Hurst R, Liebeskind D, Mariak Z, Melhem E, Krejza J (2008) Learning vector quantization neural networks improve accuracy of transcranial color-coded duplex sonography in detection of middle cerebral artery spasm—preliminary report. Neuroinformatics 6:279–290

Vapnik V (1995) The nature of statistical learning theory. Springer, New York

Widrow B, Hoff ME (1960) Adaptive switching circuits. IRE WESCON Conv Record 4:96–104

Xu Y, Fern A, Yoon S (2009) Learning weighted rule sets for forward search planning. In: Workshop on planning and learning, ICAPS-2009

Yoon SW, Fern A, Givan R (2006) Learning heuristic functions from relaxed plans. In: ICAPS'06: proceedings of the 16th international conference on automated planning and scheduling. AAAI, pp 162–171

Yoon S, Fern A, Givan R (2008) Learning control knowledge for forward search planning. J Mach Learn Res 9:683–718

Zimmerman T, Kambhampati S (2003) Learning-assisted automated planning: looking back, taking stock, going forward. AI Mag 24(2):73–96