Applications of Macro Learning to Path Planning

Shaul Markovitch

S. Markovitch is with the Computer Science Department, Technion - Israel Institute of Technology, Haifa, Israel. E-mail: shaulm@cs.technion.ac.il .

Abstract

Many robotics motion-planning techniques use common heuristic graph search algorithms for finding a path from the current state to the goal state. The research described here studies the application of macro-operators learning to the domain of map-driven robot navigation for reducing the search cost. We assume that a robot is placed in an environment where it has to move between various locations (for example, an office-like environment) where it is given tasks of delivering items between offices. We assume that a navigation task consists of a search for a path and movement along the found path. The robot is given a map of the environment ahead of time and has computation resources available. We suggest using this computation time for macro-learning. Macro-learning is a technique for speeding up the search process. Applying macro-learning for navigation brings interesting research problems. Using macros has positive effect on the search cost but negative effect on the solution quality (and therefore the movement cost). The paper describes an extensive set of experiments that study the tradeoff between these two effects and test their combined effect on the total navigation cost.

Keywords

Robot-navigation, Path-planning, Macro-learning, Speedup-learning, Heuristic-search.

I. Introduction

Navigation and motion-planing have been studied throughly in the robotics research community. Most of the research efforts were concentrated on representation and map-building techniques [1], [2], [3], [4]. Many of these works assumed a stage where, given the representation of the map, the robot uses one of the common heuristic graph-search techniques to find a path that is used by the robot for moving to the goal location [5], [6], [7] .

The research presented here tries to answer the following questions:

1. How the choice of a particular search technique affects the total navigation cost (i.e., the search cost plus the traversal cost).
2. How can we apply learning techniques to accelerate the search in order to reduce the total navigation cost.

Assume that a point-like robot is placed in a complex environment and is fed with an accurate map of the environment. The map is represented by a graph and can be built using any of the common road-map building techniques that generates graph representation.

Assume that the robot gets a sequence of tasks to execute, each involves traversing from one location to another. For each task, the robot uses the map to compute a path from the source location to the target location, and then moves along that path. For example, the robot may be placed in an office-like environment and be given tasks of delivering items between offices.

Assume that the robot is given the map in advance and has some computation resources to spend before it starts executing tasks. How can this computation time be used?

To be more specific, we assume the the following setup:

1. The robot is expected to receive a stream of *tasks* drawn from a distribution D, where each task is a pair of locations.
2. The robot is given an accurate *map* of the environment. The map consists of a set of locations and a transition table that tells what are the immediately accessible neighbors of each location.
3. The robot knows its current location at any moment and has no uncertainties associated with this knowledge.
4. When the robot receives a task $\langle s_i, s_g \rangle$, it first performs a computation of a *search* for a path that connects s_i to s_g, and then performs physical movement along that path.
5. The performance of the robot is evaluated by the *navigation cost* – the total time it spends per task.This is the sum of the computation time and the movement time.
6. The computation resources needed for searching the map are not trivial. If one procedure finds the shortest path and another a longer path, it is quite possible that the additional computation time needed by the shortest path procedure outweigh the saving in movement time.
7. The robot is given the map "the night before" it will start getting tasks, i.e., the robot has some computation resources that it can spend before it starts performing tasks.

The goal of the research presented here is to study methods for utilizing the off-line computation time to decrease the expected navigation cost of future tasks.

At first look the above setup looks like a typical expression of a *speedup learning* problem [8]. The goal of a learning program is to improve the performance of a system (a problem solver) with respect to some criterion. When the criterion is execution speed, we call the process *speedup learning*. However, for robot navigation, the search speed is only one of the two factors determining the total navigation cost. In this work we study the application of a well-known speedup-learning technique – *macro-learning* [9], [10], [11], [?], [?] – to the domain of robot navigation. We will primely concentrate on the ways that macro learning affect the total navigation cost.

To have a good understanding of the tradeoffs involved the paper reports the results of an extensive set of experiments in randomly-generated artificial environments. While the artificial domains used are less complex than many real domains, they provide us a carefully controlled experimental testbed that is simple enough to allow statistically-valid results. We believe that the trends observed are applicable to domains with higher complexity, although this belief deserves further investigation.

In Section II we describe the architecture of *MACROSCOP* – a general macro learning system that was used for the experiments described here. In Section III we describe the algorithm used for controlled generation of random domains used for the experimentation. In Section IV we describe a set of experiments that were performed to study the application of macro-learning for map-driven robot navigation. Section V concludes.

II. *MACROSCOP*: a general macro-learning system

A search space consists of a set of states and a set of basic operators that connect states. A macro-operator is a sequence of basic operators acquired during learning. A macro-operator is applied to a state by applying its basic operators consequently. Macros are commonly acquired using *learning by experimentation* methodology. The learning program solves a set of training problems. The solution traces are then used to extract macro-operators. It is very easy to generate a large number of macro-operators. Even if the learning program considers only the solution path, it can generate an order of n^2 macros for each solved problem, where n is the solution length. The benefit of a macro is the reduced search due to the large jumps over areas in the search space. The main cost associated with macros is the increased branching factor. When the number of macros increases, the costs may outweigh the benefit. This is a manifestation of the *utility problem* [12], [13]. In a previous work [13] we discuss the problem in details and offer the *information filtering* methodology as a framework for solving it. The framework considers a learning program as information processing system where information flows from the experience space through some attention mechanism to the learning component which generates knowledge that is used by the problem solver. The framework identifies five logical location for placing filters to increase the utility of the learned knowledge.

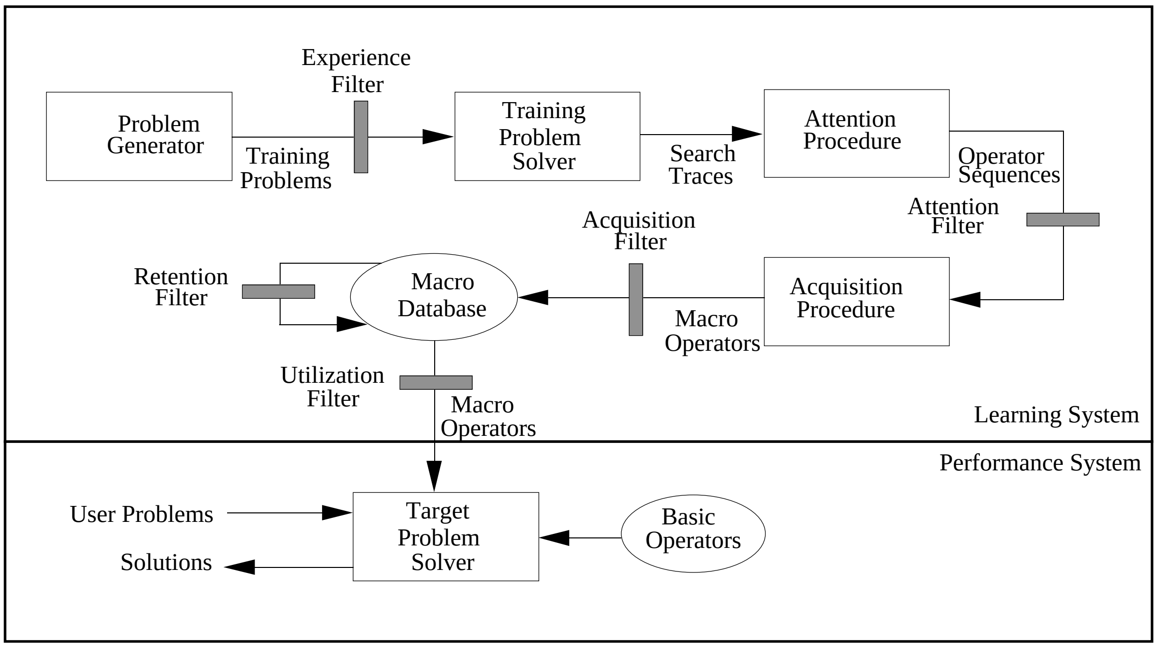


Fig. 1. The architecture of a macro learning system.

Figure 1 shows a macro-learning system in the context of the framework. The problem generator generates training problems. The experience filter selects informative problems that are solved by the training problem solver. The traces of the problem solution are then passed to the attention filter which selects sub-paths and transfers them to the acquisition procedure which builds macros and passes them through the acquisition filter to the macro knowledge base. The retention filter deletes macros that has negative utility. The utilization filter selects relevant macros and passes them to the problem solver which uses the macros just as basic operators.

The architecture described in Figure 1 was implemented in a modular macro learning system called *MACROSCOP*. The *MACROSCOP* system was designed to allow comparative factorial experimentation with selective macro learning. There are various alternatives built for each of the component and the experimenter can change them easily. There are many factors that affects macro learning. However, in this work we will concentrate on the factors that are related to our evaluation criterion – the total navigation cost. In the following subsections we will describe the instantiation of *MACROSCOP* that was used for the experiments described Section IV.

A. The problem generator

A problem generator should generate problems that are relevant and informative. We used the simplest generator that generates random problems according to the task distribution D. Without further knowledge about this distribution, the generator uses uniform distribution. The only restriction was to allow only *solvable* problems to avoid the "censored data" problem described in [14], [15].

B. The problem solver

There are many known heuristic search methods for finding a path between two states. The main dichotomy is between *optimizing* methods that search for the shortest path, and *satisficing* methods [16] that try to find a solution path as fast as possible ignoring its cost. This dichotomy is very relevant to the map-driven navigation problem. *Optimizing* methods, such as A^*, reduce the travel cost but many times carry high search cost. *Satisficing* methods, such as *best-first search*, may carry a lower search cost, on the expense of higher travel costs. Both methods were used in the context of the path-planning for robots [5], [6], [7].

To be able to shift smoothly between the two extremes, we used the *Weighted-*A^* algorithm [17]. *Weighted-*A^* uses $(1 - W)g(n) + Wh(n)$ as its evaluation function, where $g(n)$ is the cost of the path from the start node to node n, $h(n)$ is the heuristic estimate of the cost to the goal, and $W \in [0, 1]$ is the weight parameter that controls the weight of each of the factors. When $W = 0.5$ the procedure reduces to A^*. When $W = 1.0$, the *Weighted-*A^* procedure performs a greedy search and is equivalent to *best-first.* Hence we sometime call W the *Greed* parameter.

There are two instances of the problem solver. The one used during learning and the one used while performing external tasks. We can use different W for the two. We call the first W_{learn} and the second W_{test}. In previous work [18] we showed that using optimizing search ($W_{test} = 0.5$) during testing does not allow us to utilize macros.

C. The attention filter

The attention filter selects the sub-paths that will be used for macros. In previous works [19] we showed that careless acquisition of macros leads to macro sets with negative utility. Iba [11] developed the *peak-to-peak* attention filtering method that selects sub-paths between heuristic peaks in the solution path. Finkelshtein and Markovitch [**?**] developed the *minimum-to-better* method which selects paths that leads from local minima on the solution path to the first state with a better heuristic value. This method proved to be very efficient in producing good macros and was used for the experiments described in this paper.

D. The acquisition and retention filters

The acquisition filter decides whether to add a new macro to the current macro knowledge base. The retention filter deletes harmful macros. Both filter types were not used for the experiments described here.

E. The utilization filter

The utilization filter determines which of the available macros will be tried in a given state. One approach allows using a macro only in the same state where it was acquired. The problem with that approach is the lack of generalization which yields slow learning rates, large macro knowledge bases, and low macro usage. The other extreme is to try all macros in all states. This approach was used in combination with hill-climbing search in the Hillary system [**?**] and was very successful in solving problems such as the $N \times N$ sliding tile puzzle. Here we used the former approach that tries only macros that where acquired in the same state.

III. A controlled generation of artificial navigation domains

To allow a thorough experimentation of the described methodology, a domain class with the following properties was sought for:

1. It should allow random generation of domains to allow repetition of experiments for collecting sufficient statistics.

2. The generation procedure should be controllable. The properties of the generated domain should be set-able by the experimenter to allow using them as independent variables.
3. The generated domains should be realistic, i.e., they should be similar to real-world domains.
4. The domains should allow an intuitive heuristic function for the map search.
5. The generated domains should be fully connected, to avoid problems with censored data [14], [15]. Also fully-connected domains allow us to use the same set of test problems for several domains.

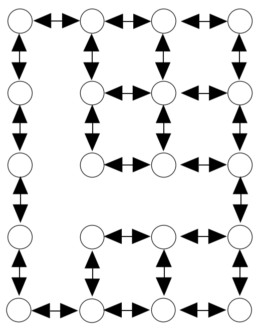


Fig. 2. An example of two walls: A south-north wall of length 3 and a west-east wall of length 2.

We have developed an algorithm for generating grid domains with walls that satisfy the above requirements. A grid domain consists of a set of $n \times m$ states. Each state i is identified by its pair of coordinates, $\langle x_i, y_i \rangle$. Each state (except those on the edges) is initially connected to four neighbor states. A random grid domain is generated by starting with a fully connected grid and disconnecting some of its links. The states are disconnected by inserting *walls*: a consequent sequence of unconnected links. Figure 2 shows an example of two walls: A south-north wall of length 3 and a west-east wall of length 2.

The complexity of the grid is controlled by two parameters:

1. *Wall density*: The portion of links that are disconnected. A grid of $n \times n$ with k disconnected links has a wall density of $k/4(n^2 - n - 1)$.
2. *Wall length*: The maximal length of a wall. To make this parameter more meaningful it is given as a fraction of n.

Figures 3 and 4 shows a set of grids generated with various settings for the wall density and wall length parameters. To make sure that the resulted domain is fully connected, walls are inserted using a special algorithm. The starting point of a wall is selected randomly. Then the algorithm disconnect a consequent sequence of links with length equal to the maximal length. The wall creation is stopped if the wall is running into another wall. This guarantees that every wall will have at least one opening and the state space will remain connected.

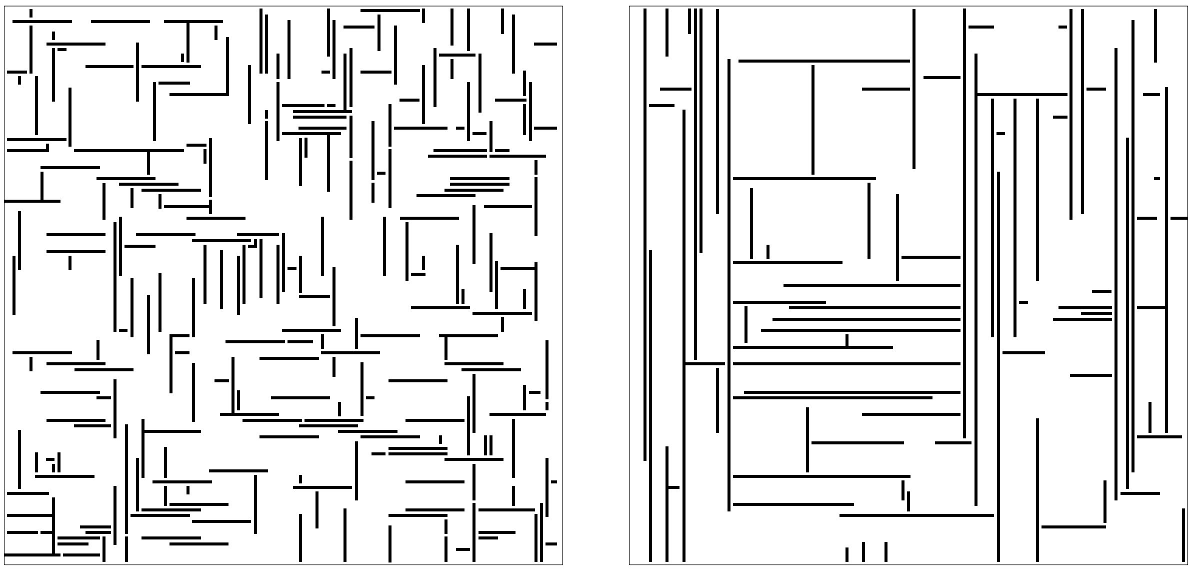


Fig. 3. Two grids with wall density of 0.1. The left grid was generated with a maximal wall length of 0.1. The right grid was generated with a maximal wall length of 0.8.

IV. Experimentation

This section describes a set of experiments performed with the *MACROSCOP* system in the robot navigation domain. The next subsection describes the experimental methodology. The following subsections describe the specific experiments and their results.

A. Experimentation methodology

There are basically two types of experiments. The first set of experiments studies the properties of the search algorithms used for navigation. The second set of experiments tests the learning algorithm in the context of robot navigation. The following subsections

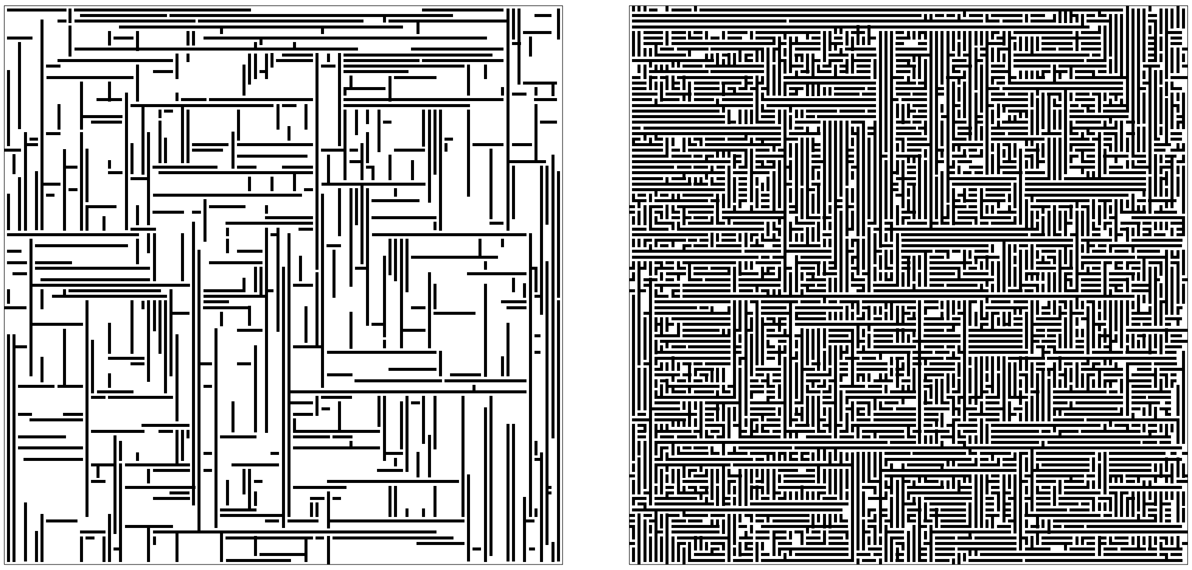


Fig. 4. The left grid has wall density of 0.2 and maximal wall length of 0.5. This is the default setting used for the experiments described here. The right grid has wall density of 0.5 and maximal wall length of 0.8.

describe the independent and dependent variables used for the experimentation.

A.1 Independent variables

There are many independent variables involved in the experimentation. We were able to study the effect of only few of them. However, we will list also the variables that were kept fixed. There are basically three sets of independent variables. Those associated with the domain, those associated with the search procedure used for testing and those associated with the learning process. For each variable we give the default values used for the experiments.

A.1.a Independent variables associated with the domain.

1. *Grid size:* Grids of size 100×100 were used for all the experiments.
2. *Wall density:* A default value of 0.2.
3. *Maximal wall length:* A default value of 0.5.

A.1.b Independent variables associated with testing.

1. *Search procedure: Weighted-A*.*

2. *Weight:* Possible values of $0.5 - 1.0$.
3. *Test set size:* Test sets of 100 problems.
4. *Utilization filter:* The only macros applied are those acquired in the current state.
5. *Cost ratio:* Determines how many moves can the robot make in the time it takes to generate one search node (usually a number which is much smaller than 1).

A.1.c Independent variables associated with learning.

1. *Search procedure: Weighted-A**.
2. *Weight:* Possible values of $0.5 - 1.0$.
3. *Training set size:* Training sets of 500 problems.
4. *Experience filter:* Problems were generated randomly from the whole space by selecting two random states.
5. *Attention filter:* Local minimum to better state.

A.2 Dependent variables

1. *Domain complexity:* We measured the domain complexity by the expected search resources spent for solving a randomly selected problem. We approximate this value by the number of generated nodes averaged over 100 randomly generated problems.
2. *Search cost:* The obvious candidate for measuring the search resources during search is the CPU time it consumes. However, CPU time depends on too many irrelevant features such as hardware, software and programming skill. The number of nodes expanded during search is a better measurement but is not appropriate for search procedures that use macros. Such procedures are likely to consume more time per expanded node due to the larger branching factor. Therefore we use the number of *generated* nodes.
3. *Movement cost:* We measure the movement cost by the length of the produced solution. The movement cost and the search cost are measured in different units. The *cost ratio* translates between the two.
4. *Navigation cost:* The navigation cost is computed by movement cost plus the search cost multiplied by the cost ratio. This dependent variable is the principle variable measured in this paper.

5. *Search cost difference:* The difference between the number of generated nodes after learning and the number of generated nodes before learning. Negative numbers mean improvement (since the cost is reduced).
6. *Movement cost difference:* The difference between the length of solution after learning and the length of solution before learning. Negative numbers mean improvement (since the cost is reduced).
7. *Navigation cost difference:* The difference between the navigation cost after learning and the navigation cost before learning. Negative numbers mean improvement (since the cost is reduced).

B. The effect of grid-generation parameters on the complexity of the domain

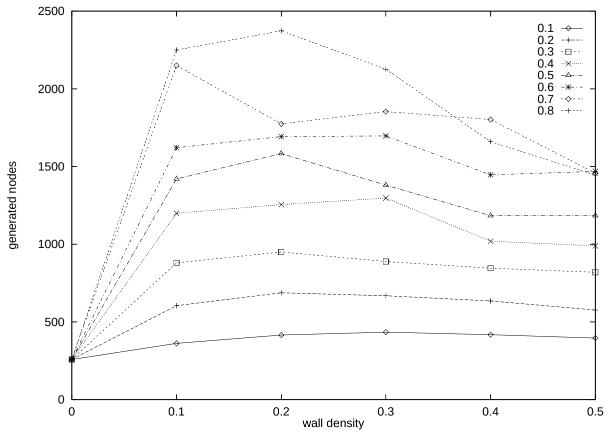


Fig. 5. The effect of wall density and wall length on the complexity of the grid domain.

To understand the behavior of the grid generation algorithm with different parameters we generated domains using wall length values of $0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8$ and wall density values of $0, 0.1, 0.2, 0.3, 0.4, 0.5$. For each of the 48 combinations, a set of 10 random grids was generated. A set of 100 random test problems was generated and solved using best-first search with the Manhattan distance heuristic function. Figure 5 shows the results of this experiment. Each point in the graph is the average result over 10 runs. It is clear that the wall length parameter is a good predictor of the difficulty of search in a domain. Longer walls make the search more difficult. The wall density parameter has a less obvious effect on the complexity of search. The complexity of search indeed increases with the increase in the wall density up to a certain point where it starts to decline. The reason for this decline is the decrease in the number of links in the space which reduces the average branching factor of the search graph.

C. The effect of search strategy on navigation cost

The cost of map-driven navigation is the sum of the search cost and the movement cost along the found path. We conducted an experiment to test the effect of the greediness parameter, W, of the *Weighted-A** algorithm on the cost of search:

1. A random grid with wall density of 0.2 and wall length of 0.5 was generated.
2. A set of 100 random test problems was generated.
3. The *Weighted-A** algorithm was used to solve the test set with W_{test} values of $0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
   Recall that $W = 0.5$ yields A* search and $W = 1.0$ yields best-first search.

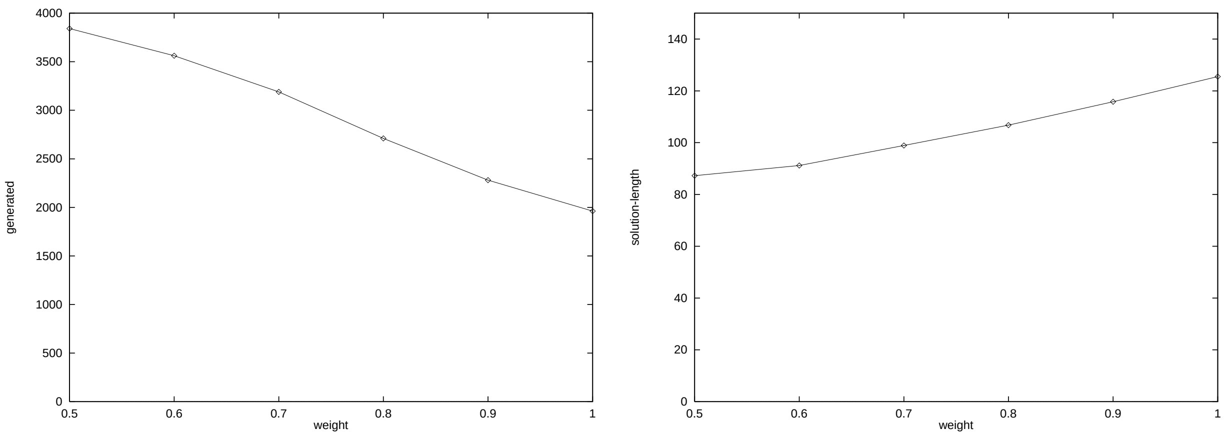


Fig. 6. The effect of the greediness parameter W on search cost and solution cost in the grid domain.

Figure 6 shows the search resources (expressed in the number of generated nodes) and the solution length as a function of the greediness of the search (expressed by W). As expected, the solution length increases with the increase in W. The cost of search decreases with the increase in W. While this behavior was the motivation led to the development of *Weighted-A** in the first place, there are domains where the cost of the search *increases* with the increase in W. Figure 7 shows similar graphs for the eight-puzzle domain. In this domain, increasing W from 0.9 to 1.0 yielded longer solutions *and* less efficient search.

Figure 8 shows the total navigation cost as a function of W and as a function of the cost ratio (the ratio between the cost of a search unit and the cost of a traversal unit). For small cost ratios the increase in W yields an increase in the total-cost due to the

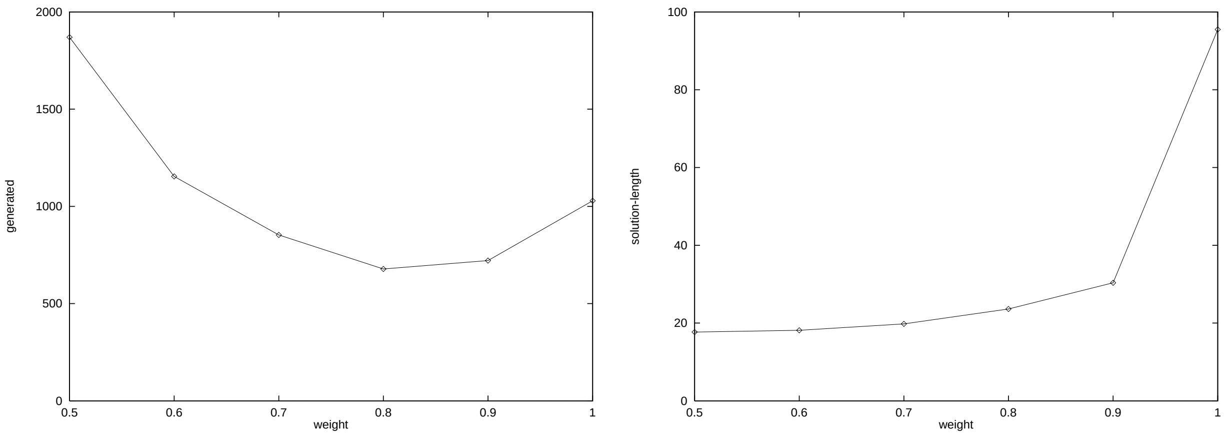


Fig. 7. The effect of the greediness parameter W on search cost and solution cost in the eight-puzzle domain.

larger impact of the solution path length. For large cost ratios the increase in W yields a decrease in the total cost due to the larger impact of the search cost. For cost ratios of about 0.02, W has no effect on the total cost. The increase in solution length is exactly offset by the decrease in the search cost.

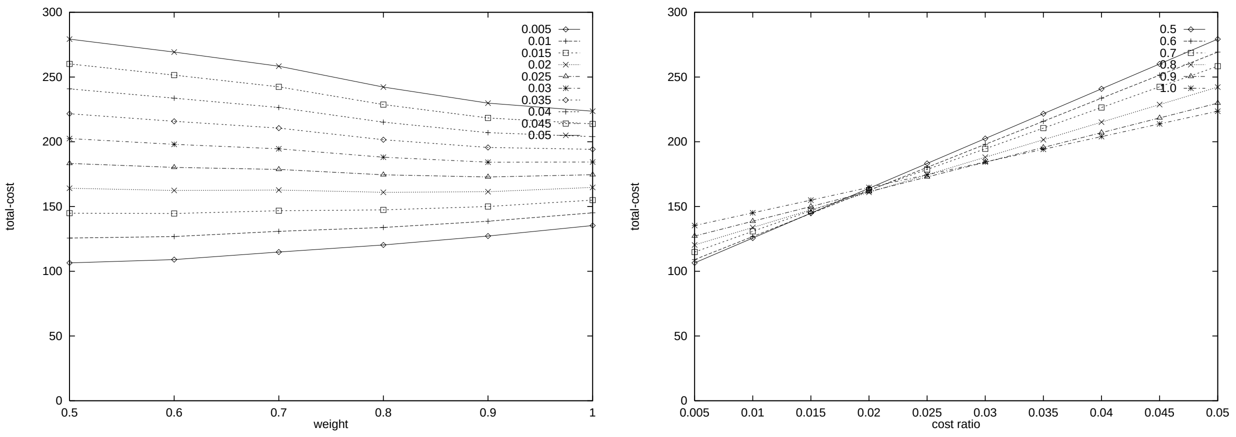


Fig. 8. The effect of the greediness parameter W on the total navigation cost. The left graph shows the total cost as a function of W for various cost factors. The right graph shows the total cost as a function of the cost ratio for various values of W.

As can be seen in Figure 6, both the search cost and solution length graphs are almost linear. Assume that the search cost is indeed a linearly decreasing function of W

$$SearchCost = -a_1 \cdot W + b_1$$

and that the solution length is increasing linearly with W

$$SolLength = a_2 \cdot W + b_2.$$

Assume that the the cost ratio is r. The total navigation cost is then

$$\begin{aligned} NavCost &= SearchCost + MoveCost = r \cdot SearchCost + SolLength = & (1) \\ &= r(-a_1 \cdot W + b_1) + a_2 \cdot W + b_2 = (a_2 - ra_1) \cdot W + (rb_1 + b_2). & (2) \end{aligned}$$

Thus the navigation cost is a linear function of W. When $a_2/a_1 > r$ the function will decrease linearly with W and when $a_2/a_1 < r$ the function will increase linearly with W. Therefore, when both the search cost and solution length are linear functions of W, the lowest navigation cost is achieved either with the lowest possible W, 0.5, or the highest possible W, 1.0, depending on the slopes of the graphs and the cost ratio.

The left graph in Figure 8 indeed shows that the total cost is a linear function of the weight. For cost ratios of 0.005, 0.010, 0.015 the total cost linearly increases with W, for cost ratio of 0.020 the graph is flat, and for cost ratios above it the total cost decreases with the increase in W. That means that for most ratios, one should either use weight of 1.0 (best first) or weight of 0.5 (A*) to get the best performance. The left part of Figure 9 emphasizes this phenomenon. It shows the W that yields the minimal navigation cost as a function of the cost ratio. The graph is indeed almost a step function. The right part of the figure shows the same graph for the eight-puzzle domain. Since the search cost and the solution length are not linear functions of W, the graph for the best W increases gradually.

D. The effect of learning resources on the utility of learning

The basic learning experiment was conducted on a fixed random grid domain with wall density of 0.2 and maximal wall length of 0.5. During each learning session the program generated a sequence of 500 training problems. A testing set of 100 testing problems was generated and used for measuring the program's performance. After processing each 50 training problems, learning was switched off and the program was given the testing set. Both training and testing were conducted with best-first search ($W = 1.0$).

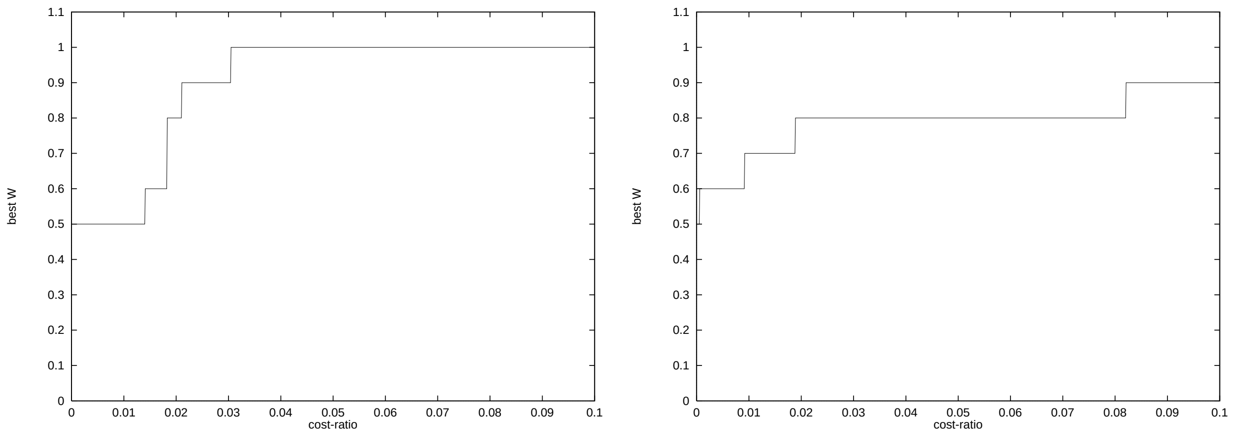


Fig. 9. The optimal W as a function of the cost ratio for the grid domain and the eight-puzzle domain.

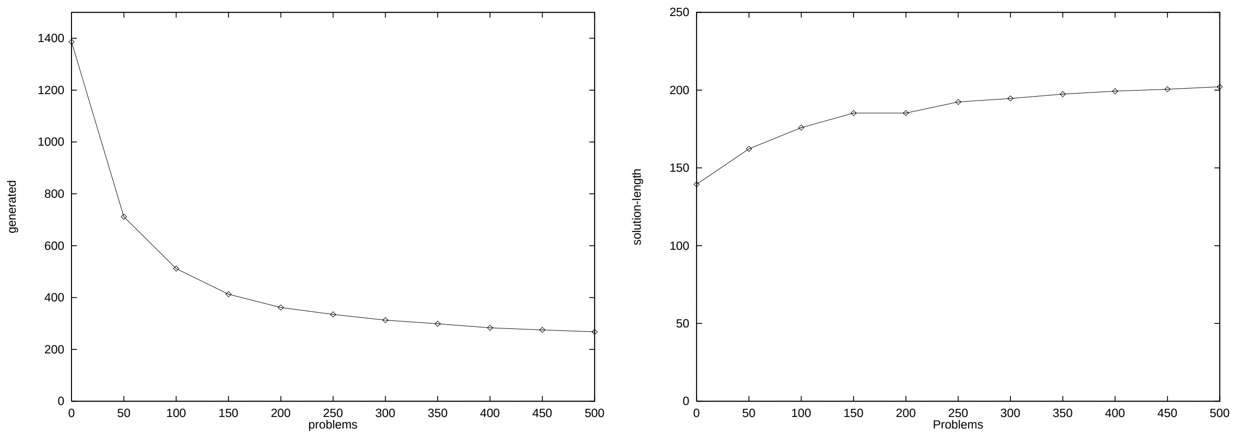


Fig. 10. The learning curves of the basic macro-learning experiment. The left graph shows the number of generated nodes as a function of the learning resources. The right graph shows the solution length as a function of the learning resources.

Figure 10 shows the learning curves averaged over 10 learning sessions. The left graph shows the number of generated nodes as a function of the number of training examples processed. We can see that macro learning was indeed successful by reducing the search cost by a factor of 5. However, for robot navigation, we must take into consideration also the length of solution produced. The right graph shows the solution length as a function of the training problems. We can see that the solution length increases with the usage of macros.

Is macro learning beneficial for robot navigation domain? That depends on the cost ratio. Figure 11 shows the total navigation cost as a function of the training problems

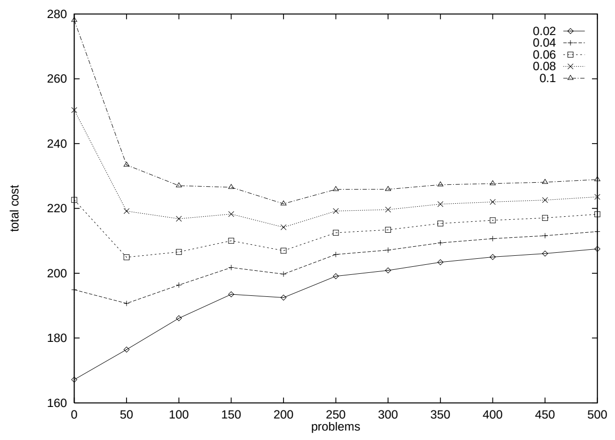


Fig. 11. The total navigation cost as a function of the learning resources for various cost ratios.

processed. We can see that for small ratios, macro learning is harmful and yields an increase in the total navigation cost. For large rations, learning is beneficial, but the improvement is modest - about 20%.

E. The effect of the search strategy used during testing on the utility of learning

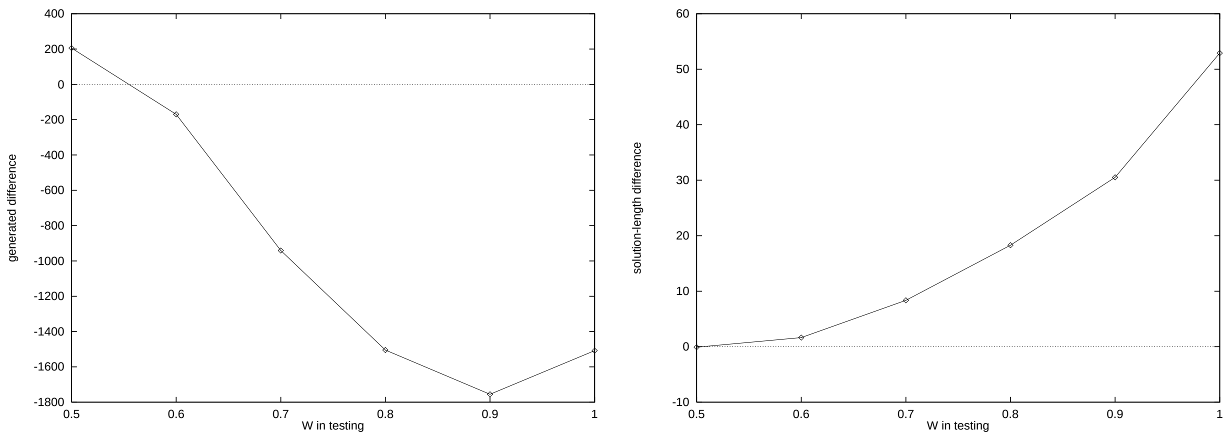


Fig. 12. The left graph shows the difference in the number of generated nodes after and before learning as a function of the weight used during testing. The right graph shows the difference in the solution length after and before learning as a function of the weight used during testing. The weight used during learning was 1.0.

In the previous experiment we showed that for small cost ratios, the total navigation cost increases with learning. In Section IV-C we showed, for a search procedure that does not use macros, that with small cost ratios the total navigation cost decreases with a decrease in the greediness parameter W. In this section we would like to test the effect of W_{test} for a search procedure that uses macros. We stored the macro data bases generated in the previous experiment and tried testing them with various W values. For each value of W, we tested each of the 10 macro data bases by solving the test set once without the macros and once with the macros. We then computed the difference in the number of generated nodes and the difference in solution length.

Figure 12 shows the results obtained. We can see that for all weight values except 0.5 the macros yield a decrease in the search cost. The harmful effect of macros, when used in *Weighted-*A^* with $W = 0.5$ (which is equivalent to A^*), is a known phenomenon that was reported by Markovitch and Rosdeutscher [18]. The reason is that even when A^* uses a macro that leads to the goal, it must validate that there is no shorter path, and performs the search that generated the macro in the first place.

The right graph of Figure 12 shows the difference in the solution length after and before learning as a function of W_{test}. We can see that using macros always yields longer solutions. This increase can be controlled by using smaller W during testing, but those small W will also yield more expensive search.

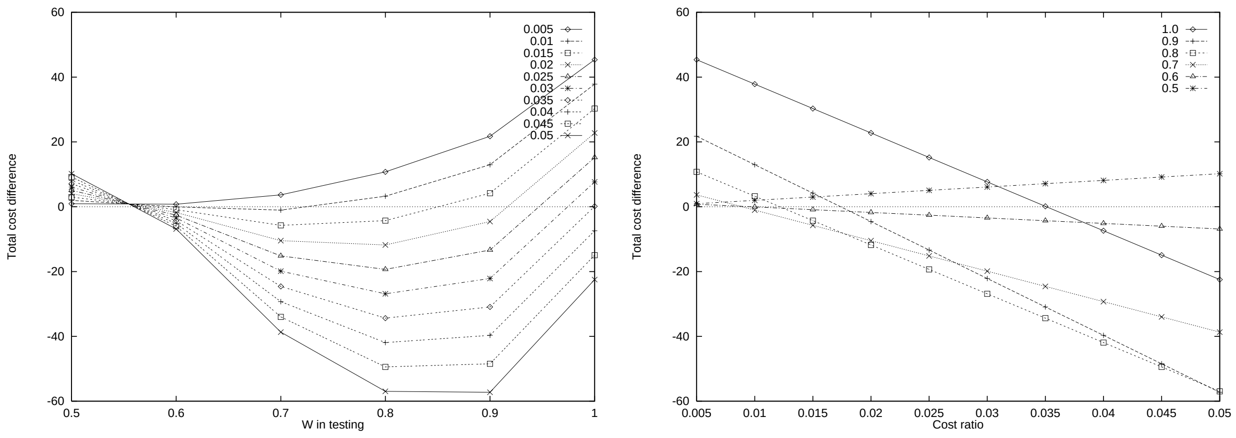


Fig. 13. The reduction in navigation cost after learning as a function of the weight used during testing for various cost ratios.

Figure 13 shows the difference in the total navigation cost after and before learning. Negative numbers indicate improvement (reduction) in navigation cost as a result of learning. For small cost ratios, the weight of the search cost is negligible, and therefore macro usage is harmful for all W values. For larger cost ratios, and for $W > 0.5$, macro learning is beneficial in reducing the total navigation cost. The optimal W for these cases is either 0.8 or 0.9. This values enable the problem solver to exploit the search cost reduction obtained by using macros without paying too much in increased solution length.

The right part of Figure 13 shows the same data with the cost ratio on the X axis. We can see that the graph for $W = 0.5$ is always positive (hence, harmful). The graph for $W = 0.8$ shows the best behavior for most cost ratios.

F. The effect of the learning search strategy on the utility of learning

The search strategy used during learning is not necessarily the same as the one used during testing. We repeated the previous experiment with various values for the weight W_{learn} used during learning. Totally, 36 combinations of W values used in leaning and in testing were used in the experiment.

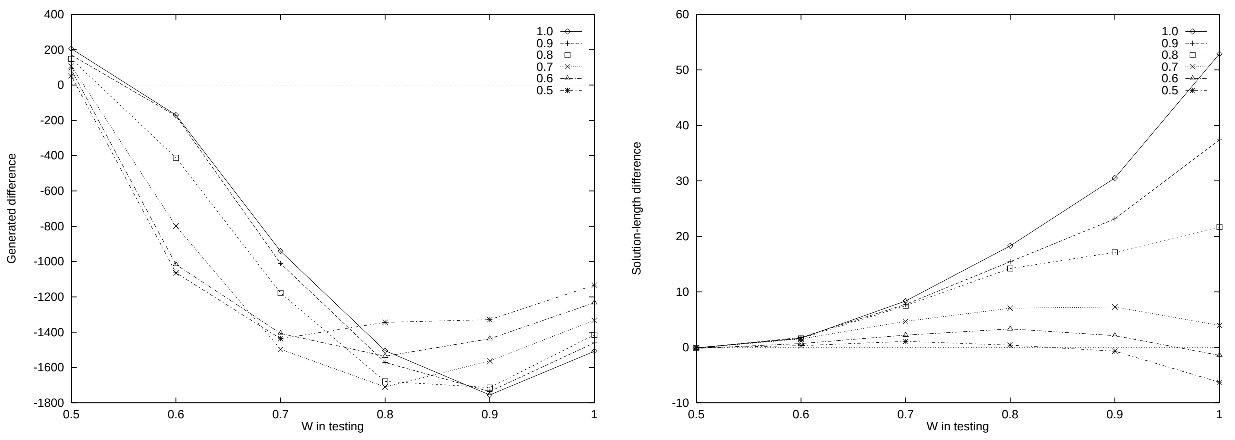


Fig. 14. The left graph shows the difference in the number of generated nodes after and before learning as a function of the weight used during testing. The right graph shows the difference in the solution length after and before learning as a function of the weight used during testing. Both graphs are shown for different values of weight used during learning.

The left part of Figure 14 shows the difference in generated nodes after and before learning. The difference is shown for all combinations of W_{learn} and W_{test}. Each graph shows, for a particular W_{learn}, the difference as a function of W_{test}. All the graphs show similar behavior. Smaller W_{learn} values have advantage when the macros are used with smaller W_{test}. Larger W_{learn} values have advantage when used with larger W_{test} values. The reason for this behavior is that larger W_{learn} produce longer macros which are not trusted by the more optimizing search procedures.

The right part of Figure 14 shows the difference in solution length as function of W_{test} for various W_{learn}. These results are much more interesting. We can see that using optimizing search during learning ($W_{learn} = 0.5$) eliminates the harmful effect of macro learning on solution length. For $W_{test} = 1$, using macros learned with $W_{learn} = 0.5$ in fact yielded a *decrease* in the solution length.

In previous subsections we showed that macro learning was sometimes harmful, causing an increase in the total navigation cost. That experiment was performed using satisficing search for learning ($W_{learn} = 1.0$). The graphs in Figure 14 indicate that optimizing search during learning has beneficial effects on both search cost and movement cost (solution length). We therefore repeated the experiment described in Figure 13 using optimizing search for learning ($W_{learn} = 0.5$). Figure 15 shows the results obtained. We can see that indeed using optimizing search for macro learning yields improvement in the total navigation cost.

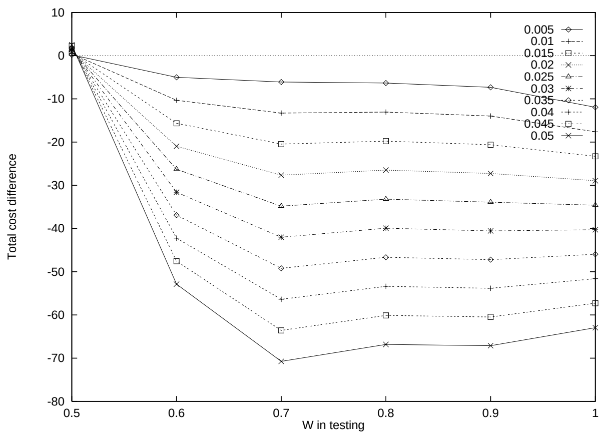


Fig. 15. The difference in the total cost after and before learning as a function of the weight used during testing. The graph is shown for different values of cost factors. All the graphs are for weight of 0.5 used during learning.

G. The effect of the domain navigation complexity on the utility of learning

Since we expect realistic domains to be more complex than those used for the experiments described here, we are interested in testing the effect that the domain complexity has on the benefit of macro-learning.

The experiments described in Section IV-B showed that the *maximal wall length* parameter has the best correlation with the domain search complexity. Therefore, to generate domains with increasing complexity, we generated 10 random grids for each of the wall-length parameters: 0.1, 0.2, 0.3, 0.4 ,0.5, 0.6, 0.7, 0.8 (total of 80 domains). For each of the domains we performed a learning session (with $W_{learn} = 0.5$) and measured the difference in performance after and before learning.

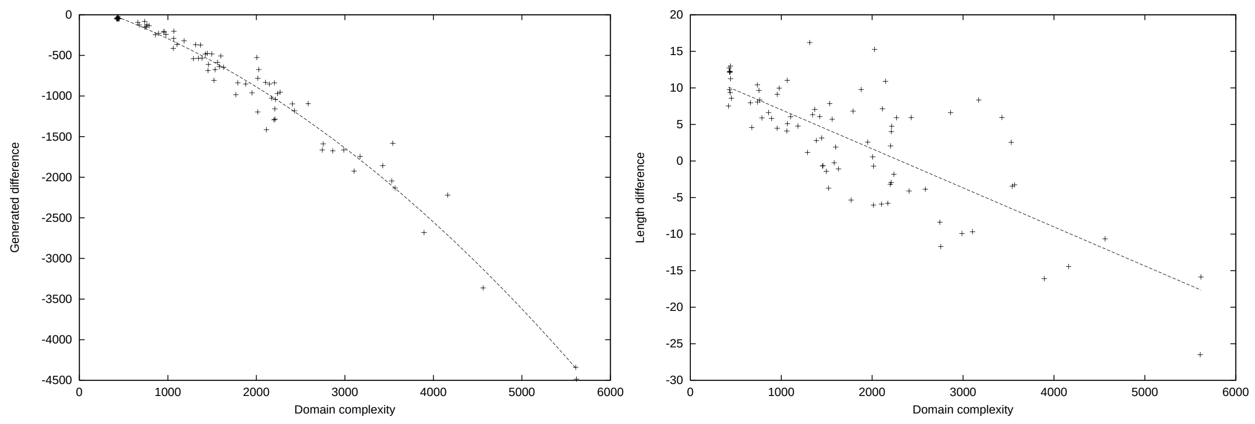
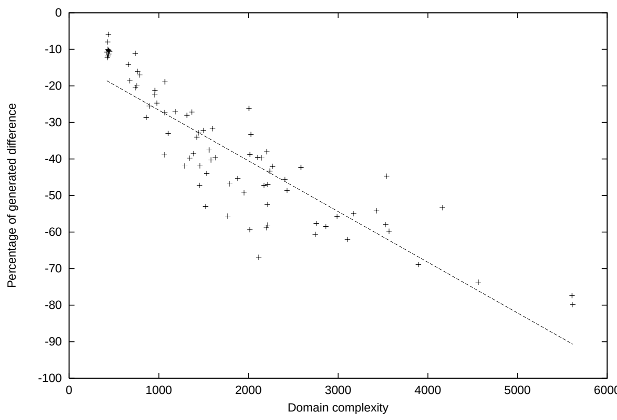


Fig. 16. The difference in the number of generated nodes and the solution length as a function of the navigation space complexity. The complexity of the space is measured by the mean number of generated nodes for solving problems before learning. The curve in the left figure is a least-square-fit to a second degree polynom (with RMS=153.62). The line in the right graph is a linear least-square-fit (with RMS=5.15).

Figure 16 shows the difference in the number of generated nodes and the solution length as a function of the navigation space complexity. The complexity of the space is measured by the mean number of generated nodes for solving problems before learning. The curve in the left figure is a least-square-fit to a second degree polynom (with RMS=153.62). The line in the right graph is a linear least-square-fit (with RMS=5.15).

The graphs indicate that indeed, when the search complexity increases, so does the saving obtained from macro learning. Another way to view the data is to look at the saving as *percentage* of the initial value. Figure 17 shows the differences expressed as percentage. We can see that the benefit of macro-learning for the search cost increases even percentage-wise while the increase in the solution cost slows down.

The main question to ask is what is the combined effect of the two parameters, i.e., what



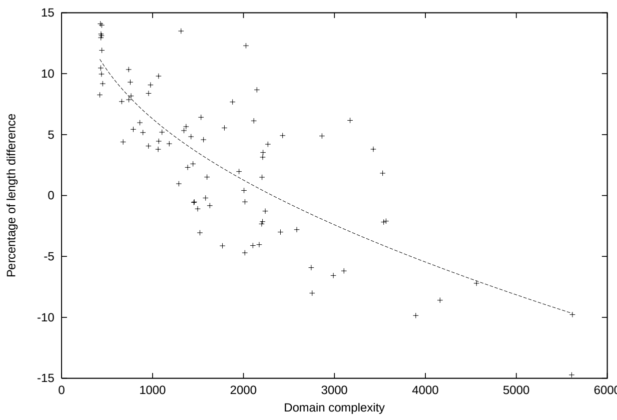


Fig. 17. The difference in the number of generated nodes and the solution length as a function of the navigation space complexity. Both differences are expressed as percentage of the initial value. The complexity of the space is measured by the mean number of generated nodes for solving problems before learning. The line in the left figure is a linear least-square-fit (with RMS=8.73). The line in the right graph is a least-square-fit to an exponentially decreasing function (with RMS=3.91).

is the effect of the space complexity on the difference in total navigation cost. Figure 18 shows the results expressed once as absolute improvement and once as a relative improvement (in percentage). These graphs show that when the space complexity increases, we can expect increased benefit of macro-learning to the total navigation cost.

V. Conclusions

The research described in this paper studies the possibility of using macro-learning techniques for reducing the search cost in map-driven robot navigation. We assume that the robot has an access to a map of its navigation environment and that a navigation task is performed by first searching for a path on the map and second moving along the found path. The total navigation cost is the search cost plus the movement cost. Macro-learning affects both the search cost and solution cost (movement cost). The experiments described in this paper study the tradeoff between these two effects.

The most important conclusions are:

1. Even without learning, it is important to select the right method of search when doing map-driven navigation. We fixed the search method to be *Weighted-A** so that the weight will determine the search strategy. A weight of 0.5 yields optimizing

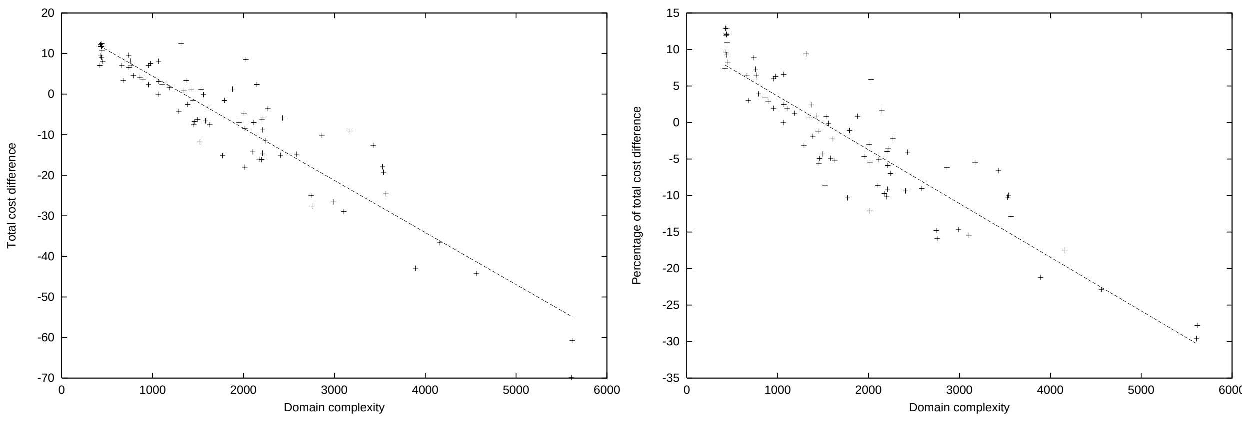


Fig. 18. The difference in the number of generated nodes and the solution length as a function of the navigation space complexity. Both differences are expressed as percentage of the initial value. The complexity of the space is measured by the mean number of generated nodes for solving problems before learning. The line in the left figure is a linear least-square-fit (with RMS=6.0). The line in the right graph is a least-square-fit to an exponentially decreasing function (with RMS=3.85).

strategy (A^*) and a weight of 1.0 yields satisficing strategy (best first). The most important factor in determining the strategy is the ratio between the search cost and the movement cost. When the movement cost becomes extremely dominant, it is always better to use optimizing search. As the ratio increases (and movement cost becomes less dominant), the search should be less optimizing. We have shown that for the robot navigation domain, when the search cost decreases linearly and the solution cost increase linearly with the increase in the weight, the optimal weight behaves as a step function of the cost ratio. One should either use $W = 0.5$ or $W = 1.0$. For other domains it is not necessarily so.

2. When using greedy search ($W = 1.0$) during learning and during testing, macro-learning has a positive effect on the search cost and a negative effect on the solution cost. The effect on the total navigation cost depends on the cost ratio. For small ratios (when the movement cost is highly dominant), macro-learning is harmful – it yields an increase in the total navigation cost. For larger ratios, macro-learning is beneficial – it reduces the total navigation cost.
3. The weight used during testing affects the benefit of macro-learning. Using larger weights (less optimizing search) increases the saving in search cost. Using optimizing

search during testing makes macro-learning harmful by increasing the search cost. Using larger weights in testing has the reverse effect on the movement cost. Larger weights makes the macro-usage causing an increase in the movement cost. When testing the effect of the weight during testing on the combined navigation cost, the benefit of macro-learning depends on the cost ratio. For smaller cost ratios, macro-learning is harmful and yields an increase in the total navigation cost regardless of the weight used during testing. Larger weight causes larger increase. For larger cost ratios, macro-learning is beneficial and yields a reduction in the total navigation cost. The best savings is achieved for weights between the optimizing and satisficing ones.

4. For robot navigation, it is always prefered to use optimizing strategy during the learning phase. Optimizing search during the learning phase yields shorter macros which reduce the negative effect of macro-learning on the solution cost (and the movement cost). Since optimizing search during learning has minor effect on the benefit of macros to search cost reduction, its effect on the total navigation cost is positive. Indeed, a combination of optimizing search during learning and mixed search during testing yields the best results.
5. The benefit of macro-learning for robot navigation increases with the increase in the domain complexity. This is true when considering absolute improvement and holds even when considering relative improvement.

In this work we allowed several simplifying assumptions which allowed us performing extensive experimental study. In future work we plan to remove some of these assumptions and work in more complex configuration spaces. In such spaces we expect the search cost to be more dominant and macro-learning to be more effective. Even in its present form, this work can help robotics researchers who use heuristic search methods for motion planning to get better understanding of the tradeoffs involved in using search and macro-learning for map-based navigation.

References

[1] N. J. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," in *Proceedings of the International Joint Conference on Artificial Intelligence*, Washington, D. C., May 1969, pp. 509–520.

[2] O'Dunlaing, Sharir, and Yap, "Retraction: A new approach to motion-planning," in *STOC: ACM Symposium on Theory of Computing (STOC)*, 1983.

[3] Rodney A. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 3, pp. 190–197, Mar. 1983.

[4] J. F. Canny, *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

[5] Joan Ilari and Torras Carme, "2D path planning: A configuration space heuristic search," *The International Journal of Robotic Research*, vol. 9, pp. 75–91, 1990.

[6] A. M. Thompson, "The navigation system of the JPL robot," in *Proceedings of the fifth international joint conference on Artificial Intelligence*. 1977, pp. 749–757, Cambridge, Mass: MIT press.

[7] T. Lozano-Perez and M. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacls," *Comminications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[8] P. Tadepalli and B. K. Natarajan, "A formal framework for speedup learning from problems and solutions," *Journal of Artificial Intelligence Research*, vol. 4, pp. 419–443, 1996.

[9] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, pp. 251–288, 1972.

[10] R. E. Korf, "Macro operators: A weak method for learning," *Artificial Intelligence*, vol. 26, pp. 35–77, 1985.

[11] Glenn A. Iba, "A heuristic approach to the discovery of macro-operators," *Machine Learning*, vol. 3, no. 4, pp. 285–317, Mar. 1989.

[12] S. Minton, *Learning Search Control Knowledge: An Explanation-Based Approach*, Kluwer, Boston, MA, 1988.

[13] Shaul Markovitch and Paul D. Scott, "Information filtering: Selection mechanisms in learning systems," *Machine Learning*, vol. 10, no. 2, pp. 113–151, 1993.

[14] Alberto Segre, Charles Elkan, and Alexander Russell, "A critical look at experimental evaluation of EBL," *Machine Learning*, vol. 6, pp. 183–195, 1991.

[15] Oren Etzioni and Ruth Etzioni, "Statistical methods for analyzing speedup learning experiments," *Machine Learning*, vol. 14, pp. 333–347, 1994.

[16] H. A. Simon and J. B. Kadane, "Optimal problem-solving search: All-or-none solution," *Artificial Intelligence*, vol. 6, pp. 235–247, 1975.

[17] Ira Pohl, "The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, Nils J. Nilsson, Ed., Standford, CA, Aug. 1973, pp. 12–17, William Kaufmann.

[18] Shaul Markovitch and Irit Rosdeutscher, "Systematic experimentation with deductive learning: Satisficing vs. optimizing search," in *Proceedings of the Knowledge Compilation and Speedup Learning Workshop*, Aberdeen, Scotland, 1992.

[19] Shaul Markovitch and Paul D. Scott, "The role of forgetting in learning," in *Proceedings of The Fifth International Conference on Machine Learning*, Ann Arbor, MI, 1988, pp. 459–465, Morgan Kaufmann.