University of Alberta

AUTOMATED ACTION SET SELECTION IN MARKOV DECISION PROCESSES

by

Greg Lee ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2004

Your file Votre référence
ISBN: 0-612-95792-6
Our file Notre référence
ISBN: 0-612-95792-6

"If I have been able to see further, it was only because I stood on the shoulders of giants."
Isaac Newton

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Vadim Bulitko for being incredibly involved and helpful as a supervisor. Our weekly meetings not only helped guide my research, but also led to my introduction to many different facets of Computing Science, and other sciences.

Thanks to Ilya Levner for extensive assistance with one of the testbed domains, as well as his contributions during general discussions in our meetings. Thanks to Lihong Li for his interest and input, as well as his courtesy in sharing resources.

I would also like to thank Calvin Thomas for extensive remote discussion about my work, his work and science in general, which helped make the less exciting days of research seem more meaningful. Thanks to Rene Malenfant, Wes Mackay, Jonathan Newton and Dave O'Connell for their coding contributions during preliminary experiments.

To my family, thanks for never pushing me an in any direction academically and allowing me to find my own way. Thanks to Melissa for her love and support during both the easy and hard times of research, and for helping in any way she could with work not at all in her area of expertise.

Lastly, I would like to thank NSERC, iCORE and the University of Alberta, without whom none of this research would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Choice: One of the chief advances in modern society is the vast number of choices people have in every aspect of life, be it what to make their profession, where to reside, where to travel, how large a family to have, what to make their priorities or even which computer operating system to use. Only a few decades ago our choices were severely limited and there was a constant struggle for more options in life. With many societal developments (feminism, affirmative action, globalization) and technological advances (television, internet), humans now have myriad options to choose from in every aspect of life. But with the gift of many choices comes the burden of making the *correct* choice, the *best* choice - the *optimal* choice. Often being given more choices comes hand in hand with being given better choices, but intelligence is required to be able to tell the good choices from the bad choices. Thus it may be sometimes better for a person to be presented with fewer choices, even at the cost of the *optimal choice*, since the person would have fewer chances of making a bad decision.

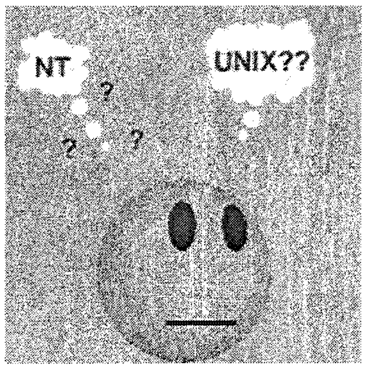


Figure 1.1: A typical problematic decision faced by computer users [Bod Host 2004].

Take for example a carpenter with a toolbox. The ideal toolbox for this carpenter is the one that maximizes his or her performance on a given task. Most carpentry tasks involve a series of steps, thus requiring a decision as to what tool to use at each step. The carpenter must follow a *policy* to decide which tool is the best to use, given the current *state* of the construction. Intuitively, we would think that providing this carpenter with every possible tool would be the best scenario, because the optimal choice for any given situation would then be in his or her toolbox. But this method of selection would also include countless useless tools which, if chosen, would lead to poor outcome. This would not matter if the carpenter had an *oracle* to tell him or her which tool to use at all times. Unfortunately, in the real world there is no such oracle, thus we need to *optimize* the toolbox for a given carpenter and a given task. A carpenter with little experience should probably be given only a few general purpose tools, in order to minimize the chances of a highly suboptimal choice. On the other hand, a more experienced carpenter could be a given a broader set, with obscure tools good only for one state. His or her policy would ensure proper use of more precise tools. Also, since tools cost money, the carpenter's company would be interested in supplying toolboxes that provide him or her tools that will allow for optimal (or near-optimal) performance while drastically reducing the total expense of the tools in the toolbox. A big toolbox is also hêavy and cumbersome, and if some less useful tools can be removed from the toolbox, it can only help the carpenter.

Psychological studies further reinforce the idea that more choice can be detrimental to "performance". In [Schwartz, 2004], human happiness is examined in the context of the amount of choice available to people. Individuals are divided into two groups: *maximizers* and *satisficers*. Maximizers seek to always make the *optimal* choice, and tend to evaluate as many choices as possible before ultimately making a decision. Satisficers seek to make a *good* decision, but do not spend too much time evaluating their choices, thus often settling for a *sub-optimal* result.

Satisficers tend to be happier people, because they do not spend much time deliberating about what choices they could have (or should have) made. They do not suffer from regret as do maximizers, since they do not even know about many of the options that were present at the time of the decision. Maximizers exert a significantly larger amount of effort in making their decision, since they force themselves to make so many more comparisons with other choices (and can be "weighed-down", much like a carpenter). Then, if the choice they have made turns out to not be as satisfying as hoped, they feel worse than satisficers since they committed so much more to the decision. In addition to this, since they are seeking the optimal choice, they can only be satisfied by a small set of decisions (often just one), the *correct* decisions (or what they perceive to be the correct decisions). Maximizers also tend to continue to compare their choice with others long after the final decision is made, often leading to further depression. And again, like the carpenter, they may have to spend more resources to evaluate more choices.

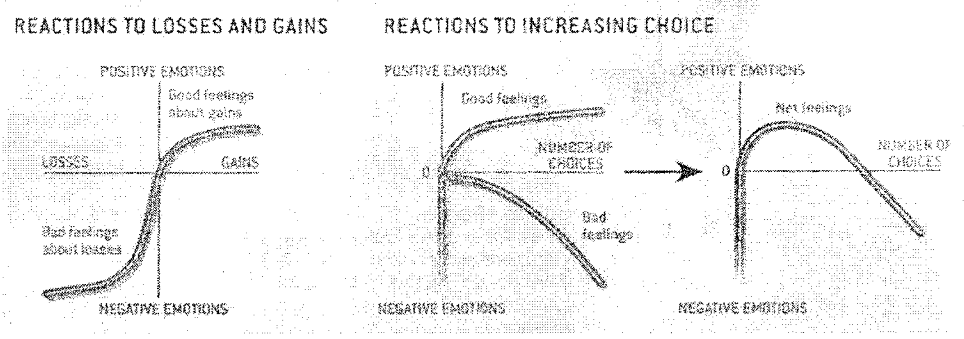


Figure 1.2: More choice is not always better for human happiness. On the left we see that bad feelings about losses outweigh good feelings about gains. The two graphs on the right show how increasing choice is only beneficial to a point [Schwartz, 2004]. After a certain threshold, the chooser is overwhelmed by the number of choices both before and after the decision is made.

Further psychological research shows that while more choice is logically better, it is only empirically better to a point. After so much choice is added, the negative factor of comparing options before and after the decision is made becomes overwhelming. Also, making a good decision is not as satisfying an outcome as making a bad decision is depressing. These findings are summarized in Figure 1.2.

One model of decision-making is the Markov Decision Process (MDP) [Sutton and Barto, 1998]. MDPs describe any situation where an agent is made to choose an action based on its current state. The decision is made by the agent's policy, that maps states to actions. The action leads the agent to another state, and the cycle continues with actions being chosen until a final state is reached. The agent rarely if ever has an oracle telling it which action to choose next. Instead, the agent uses its control policy to select its actions. Thus a convoluted action set can lead to poor and slow performance. Conversely, an undersized action set can often lack the proper actions for optimal performance. Performance of an MDP and optimality of an action set will be formally defined in Chapter 2.

It is due to this phenomenon that a need for a method to choose an *optimal* action set for a given MDP arose. This *optimal* action set is dependent upon the agent, as differing agents will require different actions to optimize their performance. Some agents will be led astray by certain states, thus if we can eliminate actions which lead to these states, we increase the performance of the agent. The optimal action set is also dependent upon the task at hand or the environment in which the agent is placed.

A most common approach is to have a human *domain expert* handpick an action set for a given domain [Draper *et al.*, 2000]. This is not desirable, since it involves much human expert time, and also is a deterrent in the automation of intelligent systems. In addition to this, it is difficult to manually construct an optimal action set for an agent and if we tweak the agent even a little, a whole new action set would need to be designed by the domain expert. In the end, one would need a domain expert on hand at all times to choose a new action set if any changes are ever to be made to the system.

In order to eliminate the need for a domain expert, we propose *automating* the action set selection for Markov Decision Processes. We then test our automated method on two vastly different domains and evaluate the results.

1.2 MDP Applications

An agent in a maze domain provides a simplistic example of a Markov Decision Process. This agent moves about a maze with a given set of *actions*, which transport it from one *state* to another. The agent's task is to discover the *goal state* of the maze. A convoluted action set could lead to the agent wasting time trying similar actions. Too few actions could prevent the agent from ever reaching its goal. Optimizing this action set leads to better performance by the maze agent. Performance is based on the state of the agent when it decides to quit (goal state or non-goal state), how many moves the agent makes before it quits (the fewer moves, the better) and how much the agent deliberates before making a move.

While the maze domain is a toy problem, there exist real-world Markov Decision Process problems whose current solutions could greatly be enhanced by optimizing the action set involved. One modern day computer vision system called MR ADORE (Multi-Resolution ADaptive Object Recognition) [Levner *et al.*, 2003; Levner and Bulitko, 2004] models vision as a partially observable Markov decision process (POMDP), with states being images (or parts of images), and actions being vision processing operators that manipulate these images. Interpreting an image can be done by applying successive operators, making object recognition a decision-making process. Having more operators in the system's set provides for more flexibility in interpretations, and strictly equal or better final labelings (if one tries all possible combinations of operators). A larger operator set also lengthens learning and execution time, since having more operators in a set leads to more possible combinations of operators, and to more useless labelings. If the agent's policy is sub-optimal, this would increase its chances of choosing a bad labeling for a given image.

In psychology terms, MR ADORE is a maximizer. It always seeks to make the optimal choice, and considers all possible options when it is making its decision. If we allow it to try all possible combinations of vision operators (which we do in practice), then the number of choices increases exponentially with the number of action applications we allow, as seen in Table 3.1. While this cannot possibly diminish the optimal choice, it does present the policy with many more choices,

Table 1.1: offline state space exploration in MR ADORE. All action sequences up to a fixed length are applied to an image. The number of nodes (intermediate images) and sequences, explored state space physical size (GBytes), and the expansion time on a dual Athlon MP 1600+ shown were averaged over 10 images. The operator set used is a greatly downsized version of the full operator set

Sequence Length	# of Nodes	# of Sequences	Size (GBytes)	Time
4	269	119	0.038	30 sec
5	7,382	3,298	1	10 min
6	192,490	86,037	26	8 hrs

with most of them being nowhere near optimal. And much as bad decisions are more detrimental to performance (happiness) in humans than good decisions are beneficial, in MR ADORE a bad labeling is much more unsatisfying to return to the user than a good labeling is satisfying.

While we could statically use a *good* sequence (as determined from our training runs), this would eliminate the *adaptability* of the system. We also run the risk of choosing a static sequence that simply fits the training data, and is not good on new images. Thus the best method to prevent bad decisions is to eliminate bad choices from possibility. A reduced domain and agent specific action set would accomplish this, all the while preserving enough actions to ensure good performance.

In the real world, the best set of choices is dependent upon the person being given the choices, and the person's task. In an MDP, the best set of actions is dependent upon the agent being made to act, and the domain in which it is acting.

The most straightforward way to find the best possible action set for a given domain and agent is to test every possible set of actions. Unfortunately, evaluating all possible action sets involves searching the powerset of the N actions (2^N). For any large N, this number practically infinite.

Since exhaustive search through all practically encountered action sets is often impossible, we turn to heuristic search methods to sample and search through the space of action sets intelligently. Any search method requires an evaluation function to determine the quality of searched instances. In some cases (such as in MR ADORE), the evaluation of just one action set (or vision operator set) can take hours, thus making a thorough search impossible in a reasonable amount of time, even with a heuristic search. To this end, we use a machine-learned evaluation function to score and rank the currently searched action sets. These *meta-models* ([Jin *et al.*, 2001]) are generalized over training data to provide our heuristic search with a quick and accurate ranking for any given action set.

1.3 Contributions

In this thesis we make three contributions to the field of Artificial Intelligence:

- We investigate the issue of 'How much choice is enough?' both implicitly (in our experiments to reduce action set size) and explicitly. We test whether increasing choice for MR ADORE aids or hurts its performance by starting with a small, but well-performing operator set. Results show that adding various operators helps performance with an oracle telling us which interpretation to choose (which is necessarily true), but with any more than 10% randomness added to the decisions, only serves to slow down the operation of the system, without changing the interpretation quality.

- We automate the selection of the *action set* for a Markov Decision Process(MDP) agent. This process is traditionally done by hand, which deters from the full automation of MDP-based systems. Our method combines the strength of wrapper [Kohavi and John, 1997] and filter approaches [Kira and Rendell, 1992; Pudil *et al.*, 1994; Bins and Draper, 2001] (formally defined in Section 4.1), providing a fast yet thorough search through possible action sets. We use both genetic algorithms [Holland, 1962; Goldberg, 1989] and simulated annealing [Metropolis *et al.*, 1953] as heuristic search methods, and artificial neural nets (and perceptrons) [Haykin, 1994], naïve Bayes classifiers [Jensen, 1996], decision trees [Quinlan, 1993], and decision lists [Rivest, 1987] as objective functions inside these search methods. We test our method against pure filter and random methods, as well as action sets chosen by a domain expert. Our method outperforms the filter and random approaches and matches the performance of the domain expert.

- We empirically evaluate the novel automated action set selection method in both the maze domain and the vision domain. We reduce the action set for a maze domain agent, improving its fitness (defined in Section 3) by increasing its likelihood of solving the maze, while reducing the number of moves it takes to reach the goal state and the number of moves considered at each state. This domain is an example of a domain where no expert is available. In these situations, our method can be especially helpful, since it outperforms other automatic selection methods. In a state of the art adaptive objection recognition system (MR ADORE), we first determine the best search methods, machine learners for evaluation functions, as well as parameters for both of these key elements in the method. We then reduce the vision operator set improving the image interpretation accuracy on novel images while reducing the learning and execution cost of the system twenty-five-fold.

As in psychology, adding more choice in an MDP is beneficial, but only to a point. Our novel method of action set selection *learns* to provide the agent at hand with an adequate, but not superfluous, number of actions for a given domain, so as to improve performance over hand-engineered implementations of such systems.

1.4 Thesis Organization

The remainder of the thesis is organized as follows: In Chapter 2 we formally define Markov Decision Process framework, as well as the applications used in the thesis. In Chapter 3, the action selection task is described. Chapter 4 outlines related work in the similar field of feature selection, heuristic search and machine-learned evaluation functions. Chapter 5 describes our four step method for choosing action sets. Our empirical evaluation is given in Chapter 6, with discussion following in Chapter 6.5. Concluding remarks are given in Chapter 7.

Chapter 2

Markov Decision Processes

A Markov Decision Process involves an agent that can be in a set S of distinct states, and who can perform a set A of distinct actions to move between these states [Sutton and Barto, 1998]. In this thesis, we concern ourselves with *finite-horizon Markov Decision Processes*, where S and A are finite. At each time step t, the agent perceives its current state $s_t \in S$ and selects an action $a_t \in A$ to perform. Each action is given a reward r_t based on the current state and action performed.

Definition 2.1 *For any state and action, the probability of reaching each possible next state s' is:*

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$

In a similar fashion, we can predict the value of the reward for any given state and action:

Definition 2.2 *For any state and action, the expected value of the next reward r_{t+1} is:*

$$\Upsilon^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

where E denotes the expected value

In this thesis, we experiment with applications where the rewards are deterministic and undiscounted (i.e. later rewards are as important as immediate rewards):

Definition 2.3 *The* return *of a finite-horizon undiscounted MDP from time t is defined as*

$$\mathcal{R}_t = \sum_{t=0}^{N} r_t$$

where N is the finite-horizon.

The *solution* to an MDP is the agent's *policy* that tells the agent which action to take in each state.

Definition 2.4 *A deterministic policy π $S \rightarrow A$ mapping states to actions is defined as:*

$$\pi(s_t) = a_t$$

The *optimal* solution to an MDP is thus the *optimal policy* $\pi^*(s_t) = a_t$, which maximizes R. The *value* of a state is dependent upon π:

Definition 2.5 *The value of a state V under policy π is:*

$$V^{\pi}(s) = E_{\pi}[\mathcal{R}_t(s)]$$

where E_{π} is the expected value if the agent follows policy π.

Actions are also given values, dependent upon the state:

Definition 2.6 *The value of an action a under policy π in a finite-horizon Markov Decision Process is:*

$$Q^{\pi}(s,a) = E_{\pi}\{\mathcal{R}_t | s_t = s, a_t = a\}$$

where E_{π} is the expected value if the agent follows policy π. The term Q comes from Q-learning [Watkins, 1989].

With these quantities defined, we can now describe the domains used in our experiments. We use two significantly different MDP-based systems in our research to test the ability of our method to effectively choose an action set for a broad array of MDP problems.

2.1 Maze Domain

Maze domains have been used in many different studies including general reinforcement learning [Sutton and Barto, 1998], lookahead pathologies [Madani *et al.*, 2002], the study of partially observable Markov decision processes [Miyazaki and Kobayashi, 1995], studies in neural information processing [Dayan and Hinton, 1993], geometric navigation [Blum *et al.*, 1991], genetic programming [Soule *et al.*, 1996], and overcoming incomplete perceptions [McCallum, 1993].

In the maze domain used in our experiments [Madani *et al.*, 2002], an agent is allowed to roam within the boundaries of a square maze with a randomly defined goal state. Figure 2.1 shows a typical maze. The MDP state of the agent is its current position within the maze. The agent has perfect information regarding its current state. The agent's percept at any given time is described by the tuple $\{x, y, s_0....s_m, g\}$ where x and y are the agent's coordinates, $s_0....s_m$ are the agents m sensor readings for the blocks it can reach from its current state (one of $\{empty, wall, goal\}$) and g is a binary sensor telling the agent whether or not it is in the goal state.

The agent moves between states with the MDP actions. These actions are defined in terms of a radius δ. The agent can move to any state on the border of the defined δ. The agent thus has $8 * \delta + 1$ possible actions (+1 for the "quit" action) in its action set A, and increasing the radius

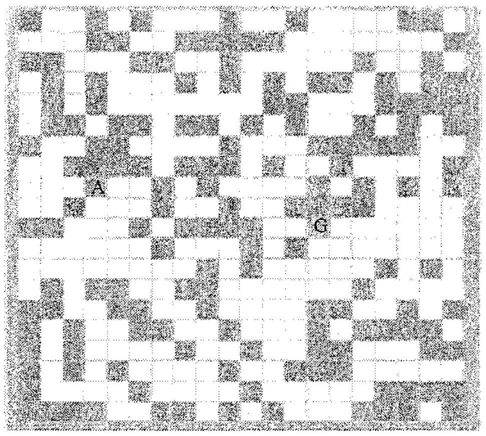


Figure 2.1: A 20x20 maze with walls shown as unlabeled squares, the agent's position shown by the red A square, and the goal state shown in green G square.

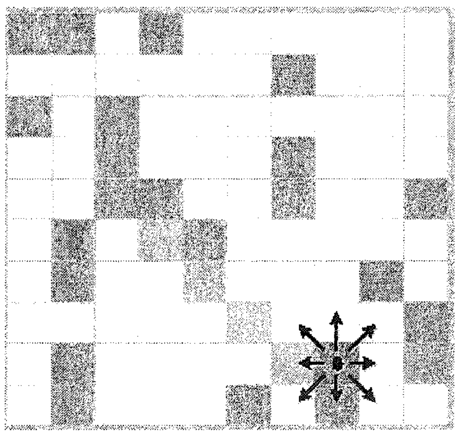


Figure 2.2: Actions available to a maze agent with radius size 1. The agent is in the magenta square, its path is described by the diagonal blue line of squares and the goal is in the red square .

causes a linear increase in the number of possible actions available to the agent. Figure 2.2 shows the available moves to an agent with actions of radius size 1.

The agent is capable of "jumping" over walls, but cannot "land" on a wall. Thus, as long as there is no wall in the state mapped to by an action (and as long as this move doesn't leave the boundaries of the maze), it is a legal action. Note that the agent cannot move in distances smaller than (or greater than) δ.

The agent is given a maximum number of action applications with which to find the goal state. A non-discounted reward function is used such that every reward incurred is of equal importance. Every move incurs a negative reward equivalent to the floor of the Manhattan distance (the distance between two points measured along axes at right angles) between the current state and the destination state. If the agent quits in the goal state, it receives a large positive reward, if it quits in any other state it receives a large negative reward.

We use this domain as a testbed for our action selection methods as it is a "traditional" MDP problem (as all the related research would attest to), so it is interesting to see if our methods can find an action set that is less convoluted than the full action set.

2.2 MR ADORE

Image interpretation is an important and highly challenging problem with numerous practical applications. Hand-crafted image interpretation systems suffer from expensive design cycle, a high demand for expertise in both subject matter and computer vision, and the difficulties with portability and maintenance. Over the last three decades, various *automated* ways of constructing image interpretation systems have been explored. The following brief account is based on [Draper, 2003].

One of the promising approaches to automatic acquisition of image interpretation systems lies with treating computer vision as a control problem over a space of image processing operators. Early attempts used the schema theory [Arbib, 1972; 1978]. While presenting a systemic way of designing image interpretation systems, the approach was still *ad-hoc* in its nature and required extensive manual design efforts [Draper *et al.*, 1996].

In the 1990's the second generation of control policy based image interpretation systems came into existence. More than a systematic design methodology, such systems used theoretically well-founded machine learning frameworks for automatic acquisition of control strategies over a space of image processing operators. The two well-known pioneering examples are a Bayes net system [Rimey and Brown, 1994] and a Markov decision process (MDP) based system [Draper *et al.*, 2000].

The latter system (called ADORE for ADaptive Object REcognition) learned dynamic image interpretation strategies for finding buildings in aerial images. As with many vision systems, it identified objects (in this case, buildings) in a multi-step process. The input data were raw images, and the output was an interpretation which identified buildings in the image; in between, the data could be represented as intensity images, probability images, edges, lines, or curves. ADORE modeled image interpretation process as a Markov decision process, where the intermediate representations were continuous state spaces, and the vision procedures were actions. The goal was to learn a dynamic control policy that selects the next action (i.e., image processing operator) at each step so as to maximize the quality of the final interpretation.

ADORE, which was a pioneering system, left several exciting directions for future work and improvement. These directions are investigated in a project titled MR ADORE (Multi Resolution ADORE) [Levner *et al.*, 2003; Levner and Bulitko, 2004]. *Multi Resolution* means it is possible to change the resolution of the images during the interpretation process.

MR ADORE begins with the Markov decision process (MDP) as the basic mathematical model by casting Image Processing Library (IPL) operators as the MDP actions and the results of their applications as the MDP states (Figure 2.3). An example of an IPL operator would be a procedure for converting a colour image to a greyscale image. The system operates in two modes as follows.

During the *offline training stage* (Figure 2.4), available subject matter expertise is encoded as a collection of training images with the corresponding desired interpretation (the so-called ground truth). Figure 2.6 demonstrates an example of such a pair (input image, ground truth label). Offline training continues by invoking an off-policy reinforcement learning algorithm that uses deep backups without bootstrapping ([Sutton and Barto, 1998])to acquire its value function [Sutton and Barto, 1998]. Specifically, at first, all feasible length-limited sequences of IPL operators are applied to each training image. The resulting interpretations are evaluated against the ground truth provided by the user. MR ADORE uses a pixel-level similarity scoring metric defined as the ratio of the number of pixels labeled as the target class (e.g., spruce) by both the system and the expert to the total number of pixels labeled as the target class by either one of them. According to such a metric, an interpretation identical to the user-supplied label scores 1 while a totally disjoint interpretation will get a score of 0.

The interpretation scores are then "backed up" along the IPL operator sequences using dynamic programming. As a result, the value function $Q : S \times A \to \mathbb{R}$ is computed for the expanded states $S' \subset S$ and applied actions $A' \subset A$. The value of $Q(s, a)$ corresponds to the best interpretation score the system can expect by applying operator a in state s and acting optimally thereafter. In reinforcement learning terms, MR ADORE represents the task as a finite horizon non-discounted problem wherein all intermediate rewards are zero except these collected by outputting an image interpretation. The latter is a positive reward proportional to the quality of the interpretation.

The collected training set of Q-values $\{[s, a, Q(s, a)]\}$ samples a tiny fraction of the $S \times A$ space. Correspondingly, function approximation methods are used to extrapolate the value function onto

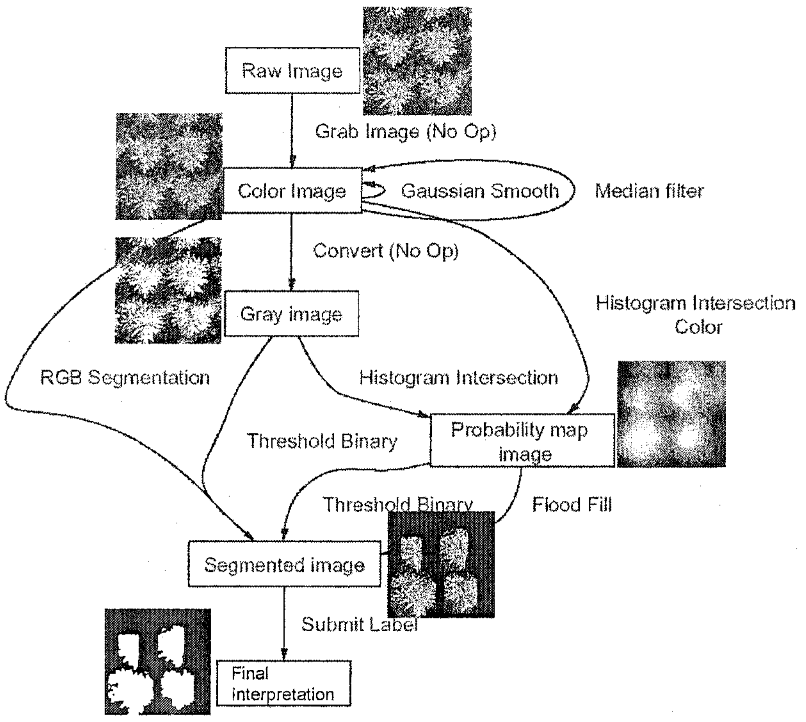


Figure 2.3: A fragment of the state-action graph used in our experiments. States are labeled with their vision data types and have forest samples shown next to them. Image processing operators are shown as the arcs.

the entire space. To make approximation tractable, raw multimegabyte states are currently distilled down to a 192 features and are presented to Artificial Neural Networks, which in turn act as function approximators.

During the *online interpretation stage*, the system receives a novel image and proceeds to interpret it, as depicted in Figure 2.5. The value function, that was learned offline, now guides the control policy to apply vision operators from the IPL library.

In MR ADORE a "least-commitment" [Levner *et al.*, 2003; Levner and Bulitko, 2004] control policy is used which first applies all limited feasible sequences of operators to the input image s_0. Once the set of possible image interpretations $\{s_1, \dots, s_N\}$ is computed, the policy uses the label of each interpretation s_i to extract features from the original input image s_0. The resulting composite feature vectors $f_{s_i}(s_0)$ are used with the machine-learned value function to select the most promising interpretation s_{i^*} as follows: $i^* = \arg\max_i Q(f_{s_i}(s_0), submit)$. In other words, the policy selects the interpretation s_{i^*} that is expected to bring the highest reward when submitted (i.e., output as the system's interpretation of the input image).

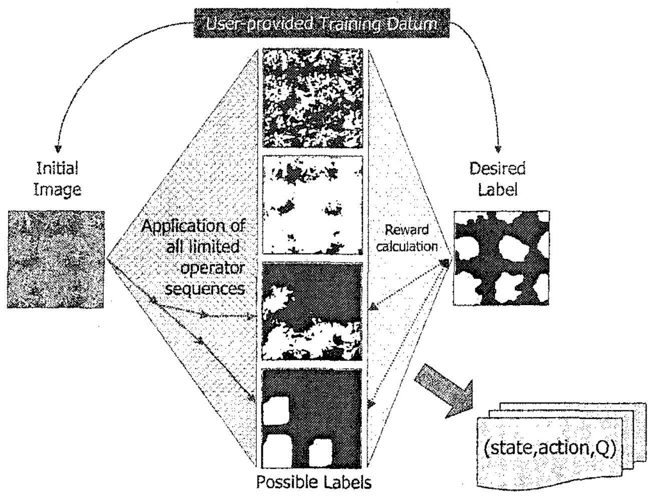


Figure 2.4: offline training stage: all limited-length operator sequences are applied to each training image. The resulting image interpretations are evaluated against the desired label. Action-state rewards are then computed.

This technique eliminates ADORE's need to design high-quality features for every processing level as they are now required for the initial colour image and the final binary interpretation only. Additionally, extracting features from the initial image provides a context for the features extracted from a candidate interpretation thereby addressing ADORE's loss of performance due to history-free Markov features.

Finally, before interpreting a novel image, it is partitioned into regular rectangular tiles. Each tile is processed independently by the control policy. The resulting interpretations (one per tile) are then assembled into a single interpretation of the original image. This technique greatly increases flexibility of the system by allowing it to use different operators on different parts of a non-uniform image. The size of tiles is determined arbitrarily.

Vision operators within MR ADORE often take parameters (such as a threshold operator, for what intensity at which to perform the thresholding). We consider different instantiations of an operator as different operators, since they perform different tasks, and thus the operator set selection process amounts to selecting not only which operators, but which instantiations of these operators to include in the set.

In this thesis, MR ADORE is used for the task of recognizing spruce trees within forestry images. The task is to label only the pixels that belong to spruce trees. Scoring is done at the pixel level. Foresters are interested in counting the number of trees in an image (and learning the details for each tree), which is not provided by MR ADORE. It is the responsibility of another module to interpret the labels provided by MR ADORE.

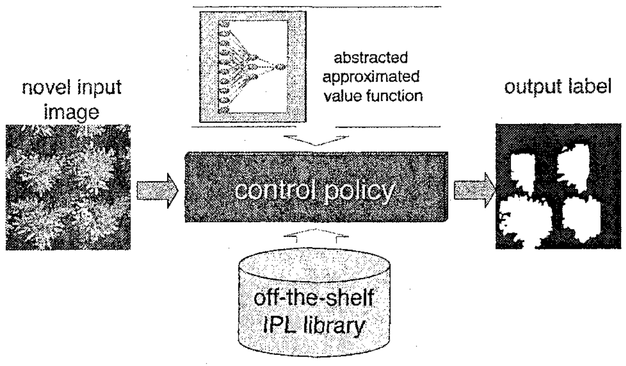

novel input image
abstracted approximated value function
output label
control policy
off-the-shelf IPL library

Figure 2.5: online operation: the control policy uses an approximate value function to select the best sequence of operators from the IPL library. As the result, an image interpretation label is produced.

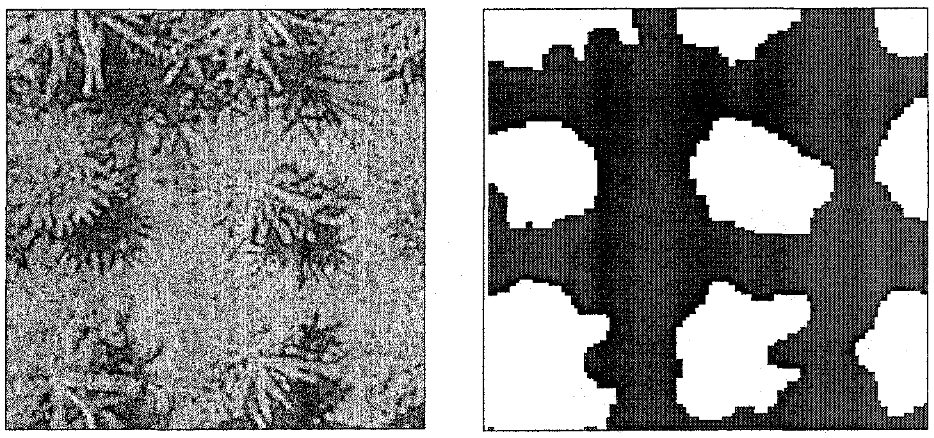


Figure 2.6: Original aerial forestry image (left) and its labeling (right) superimposed over the original image, provided by the user as a part of the training set.

Chapter 3

Problem Formulation

The task at hand in the thesis research is to automatically select a subset of actions from a presumably oversized full action set for use in an MDP. This subset should be the optimal action set for a given task, be it object recognition (vision operators), maze navigation (movement operators), or some other MDP domain.

We consider three factors with respect to the *optimality* of an action set A used by agent executing policy π. First comes the *learning cost* incurred by the agent while learning a policy for the task.

Definition 3.1 *The learning cost is defined as the time in CPU cycles taken to learn the policy π used on novel data:*

$$C_l(A) = c_l(\pi)$$

where c_l is the number of CPU cycles used in the learning process.

Another penalty is the *run-time cost* incurred by the agent while performing the task.

Definition 3.2 *The run-time cost is defined as the number of CPU cycles taken to accomplish the task:*

$$C_{rt}(A) = E_{s_0}[\sum_{i=0}^{n} time(\pi(s_i))]$$

where E_{s_0} is over the distribution of starting states and $time(\pi(s_i))$ is the deliberation time of policy π in states_i.

Combining these two, we have the full cost $C(A)$ of an action set A

Definition 3.3 *The cost is defined as the total number of CPU cycles spent learning the policy and then executing the policy:*

$$C(A) = C_l(A) + C_{rt}(A)$$

The third factor is the *reward* value of the final state where the agent ceases to act.

Definition 3.4 *The reward in an undiscounted finite-horizon MDP (with M action applications) is defined as the sum of the scores given to the agent upon completion of its task:*

$$R(A) = E_{s_0}[\mathcal{R}(s_0)] = E_{s_0}[E_\pi[\sum_{i=1}^{M} r_i]]$$

where M is the finite-horizon and E_π is the expected value if the agent follows policy π.

The *fitness* F of an action set A is a combination of this *cost* and *reward*, typically defined in a linear equation:

Definition 3.5 *The fitness of an action set A is a linear combination of reward and cost:*

$$F(A) = \alpha R(A) - \beta C(A) + \sigma \tag{3.1}$$

where $\alpha + \beta = 1$ and σ is a constant. The values for α and β are provided by the user, depending upon how the importance of accuracy relative to cost. The value of σ simply determines the scale of the fitness values.

Thus, the optimal action set (A) is the set that maximizes $F(A)$:

Definition 3.6 *The optimal action set A^* maximizes the fitness measure $F(A)$:*

$$A^* = \arg\max_A F(A)$$

The optimal set varies, depending upon how much the user values reward over cost, or vice-versa.

For instance, in the maze domain, the reward is based upon whether or not the agent terminates in the goal state, and upon how many moves the agent makes before it terminates. If the agent terminates in a non-goal state, a negative reward is incurred for failing to solve the maze.

Ideally, one would like to search through all possible subsets of the given action set, and choose the optimal set (A^*) based on the predefined balance of reward and cost. This is, however, not possible for two reasons. Firstly, since the set of subsets of an action set of size N is the powerset of N, even a modest-sized set of 100 actions generates a number of subsets that is practically infinite.

Secondly, even if one could search through all the possible subsets, there is still a need for a means of evaluating each subset of actions. The most accurate method is to evaluate each action set's performance on the actual system, which in the case of MR ADORE means to learn the policy π, execute it on all training images and rank it based on its average performance. Unfortunately, performing such a task for just one action set at depth four (applying at most four actions in all sequences) can take multiple hours, making searching through thousands of action sets infeasible.

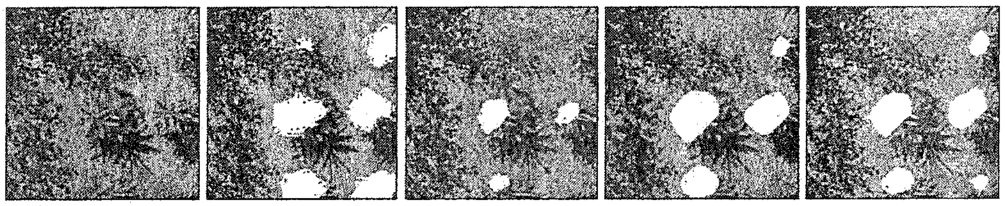


Figure 3.1: Longer operator sequences lead to better labeling. From left to right: the original image, desired user-provided labeling, the best labelings with an operator sequence of length 4, 5, and 6.

3.1 Desired Attributes of Ideal Solution

The desired end result is to have a completely portable system for choosing an *optimal subset* A^* of a provided action set in any MDP, based on the evaluation metric ($F(A)$) provided by the user.

We want the mechanism to match human experts in speed and reward. If the system takes years to choose an operator set, then it is likely not of much use, and if the chosen operator set does not perform well enough, it is again quite useless. We would also like to outperform more "naïve" methods, such as choosing an operator set based on highest ranked individual operators.

The selected operator set should hopefully be much smaller (number of actions in the set) when compared to the full set (of course this depends upon how many redundant operators are present in the full set), and thus reduce the total cost of the set C. Despite its smaller Cis, the selected set should share comparable final rewards R to the full operator set, and hopefully, equal final rewards. Having fewer operators should provide the system with fewer choices at each decision point, thus reducing the likelihood of error.

3.2 Example: Library Selection in MR ADORE

During the offline phase, MR ADORE explores the state space by expanding the training data provided by the user. In doing so it applies all operator sequences up to a certain length (Figure 2.4). Longer sequences are preferred for better image interpretation since more operators can be applied for more precise transformations of the input image into the desired labeling. Even the modest increase from 4 to 6 operators, shown in Figure 3.1, is clearly beneficial.

On the other hand, the size of the state space being explored increases exponentially with depth and therefore quickly becomes prohibitively expensive. Currently in MR ADORE, a greatly downsized (expert-selected) operator set is used, to ensure that the learning and run-time costs of the system are reasonable. With this downsized operator set used by MR ADORE for the tree canopy recognition task, the effective branching factor is approximately 26.5 which results in the sizes and timings shown in Table 3.1 (as seen in Chapter 1).

Table 3.1: offline state space exploration. All operator sequences up to a fixed length are applied to an image. The number of nodes and sequences, explored state space physical size (GBytes), and the expansion time on a dual Athlon MP 1600+ shown were averaged over 10 images.

Sequence Length	# of Nodes	# of Sequences	Size (GBytes)	Time
4	269	119	0.038	30 sec
5	7,382	3,298	1	10 min
6	192,490	86,037	26	8 hrs

There are three conflicting factors at work: (i) large off-the-shelf image processing operator libraries are required to make MR ADORE cross-domain portable, (ii) long operator sequences are needed to achieve high interpretation quality, and (iii) combinatorial explosion during the learning phase can impose prohibitive requirements on the storage and processing power. Fortunately, most domain-independent operator libraries almost invariably contain numerous redundant or ineffective operators when a specific domain is considered. Thus, the feasibility of the policy learning phase as well as subsequent online performance critically depends on the selection of an efficient operator subset for the domain of interest.

Previous systems such as [Draper *et al.*, 2000] relied on *manual* selection of highly relevant non-redundant operators thereby keeping the resulting IPL small and the offline state space exploration feasible. Unfortunately, such solutions defeat the main objective of MR ADORE-like systems: their automatic construction for a given domain. Note that the source of the original library is irrelevant. In particular, one may engage genetic programming to create such a library first [Teller and Veloso, 1996] and then run automatic operator set selection on it.

Chapter 4

Existing Methods

Action selection in Markov Decision Processes is done by the agent's policy π. Action *set* selection has been a human-expert task in the past, and automation of this task has not been seen in the literature. Feature selection is a closely-related field, thus we review various feature selection methods. A feature is a significant aspect or property of a state that can be used to compare different states. More importantly features can be *measured*, whereas many states (such as an image) cannot. [Gilmore and Hillston,]. Features give us a compact way of describing a complex state, which allows for comparisons between and operations on these complex states

4.1 Feature Selection

Selecting an action set is similar to selecting an optimal set of features in as much as the individual actions/features are interdependent, possibly redundant, and their performance can be fully evaluated only within the target system. Thus, we will first briefly review representative feature selection literature and then discuss the differences.

There are two important dimensions to consider: the type of search in the space of feature sets and the optimization criteria. Two primary approaches have been studied. Wrapper approaches [Kohavi and John, 1997] measure the actual performance of the target system with a candidate feature set. This means that whenever the search mechanism wants to evaluate a feature set, the actual system is invoked in order to evaluate how well the feature set performs. While being accurate, such optimization criteria can be prohibitively expensive in practice. For instance, in the context of MR ADORE measuring the fitness of a typical operator set on a test suite of 35 images takes around 12 hours on a dual AMD Athlon MP+ 2400 Linux server, since both the learning cost C_l and the execution cost C_{rt} are significantly large due to full expansions done in labeling novel images (with the "least-commitment" policy). Filter feature selection methods [Kira and Rendell, 1992; Pudil *et al.*, 1994; Bins and Draper, 2001] use system-independent criteria such as feature redundancy,

Algorithm 1 The RELIEF algorithm ([Kononenko, 1994]). **Input**: m training data. **Output** Optimized feature set.

```
1: set all weights $W[A]$: = 0.0;
2: for $i = 1$ to $m$ do
3:     randomly select an instance $R$;
4:     find nearest hit $H$ and nearest miss $M$;
5:     for $A = 1$ to $all\_attributes$ do
6:         $W[A] = W[A] - \frac{diff(A,R,H)}{m} + \frac{diff(A,R,M)}{m}$;
```

relevance, and other similar metrics. While such criteria are frequently less expensive to compute, they are decoupled from the actual target performance function and may not always account for the influence of domain specifics on the performance of a feature set.

The number of feature sets is usually exponential in the number of features and therefore incomplete heuristically guided search methods are typically preferred. For feature selection, greedy algorithms have been used. In [Pudil *et al.*, 1994] a Floating Search Algorithm is used to select features. This algorithm is from the family of sequential search procedures involving backtracking. The Floating Search Algorithm is shown to be effective in selecting features from both monotonic and nonmonotonic feature sets. A monotonic feature set is one where if a feature from that set is added to the current set, the performance of the system cannot be degraded.

In [Kira and Rendell, 1992] the RELIEF algorithm was introduced. RELIEF uses how well attributes distinguish among instances that are near each other to estimate the attributes' values [Kononenko, 1994]. The full algorithm is given in Algorithm 1. RELIEF takes each instance and searches for its *nearest hit* (its nearest neighbour in the same class), and its *nearest miss* (its nearest neighbour in a different class). In this way, attributes are ranked highly if they successfully differentiate between instances from different classes and give instances of the same class the same values.

Another filter method involves ranking features based on the average performance (defined by the user) of feature sets containing them, on a given system. This is similar to the $+/-$ statistic in hockey, where a player is given a $+$ if his team scores while he is on the ice, and is given a $-$ if the opposing team scores while he is on the ice. These $+$s and $-$s accumulate, and a player with a highly positive $+/-$ is considered a valuable asset to his team. In a similar fashion, a feature with a high rank would be considered an asset to a feature set. The algorithm for computing the optimal feature set with this method is given in Algorithm 2.

Once features are selected, all of them are applied to the data token at hand *simultaneously*. This is in contrast to image processing operators which are applied to the initial image *sequentially* with one operator's output being the next operator's input. Furthermore, operator application is guided by a dynamic control policy (e.g., best-first ANN-guided policy in [Draper *et al.*, 2000]) and can

Algorithm 2 "Top" , a method for selecting feature sets. **Input:** Training data, desired number of features d in set **Output:** Domain specific feature set(s)

```
1: for Each feature $a$ in the full set do
2:   for Each training datum {attributes,fitness} do
3:     if $a$ is present in the training datum then
4:       Add this fitness to $a$'s total
5:       Increment count
6:   Calculate $a$'s fitness by dividing total by count
7: Sort features by their fitness
8: Output top $d$ features
```

involve loops, back-tracking, and early termination. These additions complicate the mapping from operator sets to the resulting target system performance thereby possibly limiting the applicability of filter methods, a phenomenon that we explore in our experiments.

4.1.1 Genetic Algorithms (GAs) in Feature Selection

In [Vafaie and Jong, 1993], GAs were shown to be more robust than traditional greedy methods for selecting features to be used by the AQ15 incremental learning system to determine texture classification rules in images. A sequential backwards search (SBS) algorithm is used as the greedy method comparison to the GA. SBS starts with the full feature set and discards one feature at a time until the desired number of features are present in the set. Features to be discarded are determined by evaluating the feature set in each feature's absence, and removing the feature with the least use. In situations where there are many interdependencies between features, GAs are shown to outperform the SBS algorithm. When there are fewer interdependencies, GAs are shown to be less efficient. Since vision operators are known to be substantially interdependent (each operator depends upon the output of another operator), GAs should be suitable for the vision operator selection task. In our MDP domains, it is quite often not known how many actions are *desired* in the set, thus it would be difficult to apply SBS.

Multi-criterion optimization (e.g., minimize the feature set size *and* maximize the overall recognition accuracy) is possible with GAs as well. In [Sun *et al.*, 2002], a GA balancing accuracy and the number of features is used, by applying a linear weighting formula for the two criteria, shown in Equation 4.1.

$$fitness = 10^4 I + 0.4 \times Z \tag{4.1}$$

where I is the interpretation accuracy of the system, and Z is the number of features present in the current set. The task at hand is to determine the gender of people in images based on an eigenvector describing the image. In this task, having fewer features (i.e. only those features which encode gender information) actually increases the accuracy of the system. Comparing this to vision operators in MR ADORE, having more vision operators in offline training *always* increases accuracy, since this only gives more possible operator combinations, and we know which image interpretation is the best. In contrast to this, in the online operation of MR ADORE, having fewer operators can be beneficial, since the machine-learned decision making system is then given fewer poor image interpretations.

[Sun *et al.*, 2002] first use Principal Component Analysis (PCA) to store each image as a feature vector of low dimension. The multi-criteria GA is then used to choose features to be used by a neural net to determine the gender of the person in each image. With the reduced feature set provided by the GAs, the interpretation accuracy of the neural net is improved from 82.3% to 88.2%.

In [Vafaie and Jong, 1992] a weighted multi-criteria GA is again used. The authors use genetic algorithms as a tool to select features for a rule induction system, again for the AQ15 system. The fitness function for the GA is a weighted sum of the number of testing examples identified correctly and the number of testing examples identified incorrectly (with the GA selected features).

A similar weighted approach is used in [Oliveira *et al.*, 2001] to help select features for recognizing handwritten digits. Two types of GA are used: a standard GA (SGA), which, as its name describes, is simply a GA with the standard crossover, mutation and selection process, and an iterative GA (IGA) which is known to converge faster than SGAs [K.F.Man *et al.*, 1999] by simply restricting the search space at each iteration. The best solution found at each iteration is used as a guideline for the next generation. In this task, the SGA outperforms the IGA.

In [Oliveira *et al.*, 2002], a Non-Dominated Sorting GA (NSGA) with elitism is used [N.Srinivas and K.Deb, 1995]. A ranking selection method is used to make good solutions more prominent, while a niche method is used keep the subpopulations of good solutions stable. The NSGA outperforms traditional GAs on the task of handwritten digit recognition.

4.1.2 Pareto-optimal Genetic Algorithms in Feature Selection

Pareto-optimal GAs present an alternative method for producing solutions that must balance between objectives. A *pareto-optimal* solution is one that dominates all other possible solutions in each criterion of the problem. In [Emmanouilidis *et al.*, 1999] a *niched Pareto GA* with random sampling tournament selection is used. It is described in Algorithm 3. The authors use the Pareto GA to select features for neurofuzzy modeling of cancer data and vibration analysis data. In the former, nine features are used, while in the latter 56 features are used. In both cases Pareto GAs are able to produce reasonable solutions for different situations - where different criteria are given different weights.

Algorithm 3 Niched Pareto-Optimal GA ([Emmanouilidis *et al.*, 1999]). **Inputs**: Evaluation function, Number of bits in chromosome. **Output**: Fit population of solutions.

```
1: Randomly select Individuals from the population to create a dominance tournament group
2: Form a dominance tournament sampling set by again selecting individuals from the population.
3: Check each individual in the tournament group for domination by the dominance sampling set
   (to see it is dominated by at least one individual).
4: If only one individual in the tournament group is non-dominated, this individual is copied into
   the mating pool.
5: If more than one individual is non-dominated, or all the individuals are dominated, the individual
   copied to the mating pool is the that which will be maintain diversity, which is the individual
   with the smallest niche count, which is based on the Hamming distance between the individual
   in question and all the individuals alreadypresent in the mating pool.
6: Iterate until mating pool is full. When this is true, start the genetic algorithm with the created
   mating pool.
```

4.2 Meta-Models: A Crossover between Filter and Wrapper Approaches

Evaluating possible solutions to a problem on the system at hand can be prohibitively expensive. In [Dahm and Ziegler, 2002], genetic algorithms are used to select walking patterns and speeds for robots commonly used in robot soccer. Clearly, evaluating thousands of possible combinations is not feasible on these machines due to time and power restrictions, as well as wear and tear on the robots. Thus, alternate evaluation methods must be considered.

One common approach is to use a *simulation* of the actual system. In [Robert H. Kewley and Embrechts, 2000] the authors perform fuzzy genetic decision optimization (FGDO) on a complex stochastic system. They use a stochastic simulation model to estimate the results of parameter settings for the system, and a fuzzy ordinal preference model to aggregate the results of the simulation model into a fitness value for the parameter set. Genetic algorithms are then used with these fitness evaluations to search for high performance parameter sets for the actual system. The task the authors use to test their model is a tactical military attack route planner. They test the FGDO method against human expert attack planners, and outperform them. It took seven hours to compute 40 iterations of their GA, which they deem unacceptable, and which makes this method seem too costly in terms of C_{rt} to implement for our purposes.

Unfortunately, building a simulation model involves domain-expert knowledge and may not even be possible in certain domains. With robots for instance, it may be impossible to mathematically model the conditions of all the joints in the robots. With MR ADORE, it is impossible to model the effects of executing a vision operator on a given image (except in trivial cases) without actually executing the vision operator.

There is thus a need to replace costly evaluation via running the actual system with a *surro-*

gate evaluation function. These surrogate functions are commonly known as *meta-models* in the literature, and are used in many different applications. The idea of meta-models as evaluation functions has been evaluated in [Jin *et al.*, 2001]. The authors discuss the convergence of GAs with approximate fitness models, and use a covariance matrix method (CVA). CVAs converge quickly and are capable of self-adaptation. New data points that lie along the direction in which the GA is proceeding are given larger weight in online learning.

The authors use a *generation-based evolution control* method, meaning certain generations within the GA have their fitness evaluated by the actual mechanism (rather than by the meta-model). These generations can then be used to train the meta-models further. Another form of controlled evolution discussed in the paper is *individual based*, where certain individuals from each generation are evaluated with the true fitness function. Using the *best strategy*, the top m ranked individuals are evaluated on the actual system. Using the *random strategy* m random individuals are evaluated on the actual system. Both methods of controlled evolution can be inapplicable, depending upon the application (i.e. when the fitness evaluation is extremely expensive). *Control Frequency* determines how many generations use the true fitness function for evaluation, and is determined by how accurate the meta-model is deemed to be. The authors evaluate the methods on two benchmark data sets (Ackley Function and Rosenbrock function) as well as an aerodynamics problem. Results show that properly estimating the control frequency leads to improved results over simply employing the true evaluation function.

Periodically using the actual system to calculate fitness values within our heuristic search methods could greatly slow down the process of selecting action sets for MDP domains and agents. In MR ADORE, if a costly action set is to be evaluated on the actual system, the process could be delayed by several hours.

In [Yan and Minsker, 2003], genetic algorithms are used with artificial neural nets (ANNs) the meta-models. The authors use the early iterations of the genetic algorithm to train the ANN fitness function, by periodically retraining the ANN with data obtained from recent runs of the GA. This makes the entire process dynamic, rather than static. Empirically their method achieves satisfactory performance on the task of the risk-based remediation design model for groundwater management. Traditionally, linear programming has been used for this task, but new combinations of different submodels in groundwater management cause such an approach to yield suboptimal results. Simulated annealing has also been used for this task [Dougherty and Marryott, 1991], and has shown similar performance to GAs. The dynamic GA has adaptability as its advantage, but the meta-models must be retrained in order to adapt, which can be expensive with ANNs.

Another type of meta-model used is the kriging method [Willmes *et al.*, 2003]. The kriging method models a system as a localized, stochastic Gaussian process with a covariance matrix ($\sum$)

and an expected value (μ). The kriging method is most often used in geostatics, where data is spatially correlated [Journel and Huijbregts, 1978]. In experiments, neither neural nets nor the kriging model emerged as the best method to use for a meta-model.

It is important to remember that in order for a meta-model to be effective, it is not necessary for it to be completely *accurate*. Its main purpose is to *select the correct individuals*, no matter what *fitness* it gives these individuals. This is why traditional methods for evaluating methods do not generally work for evaluating meta-models [Jin *et al.*, 2003]. Evaluating meta-models based on their mean squared error:

$$E^{mse} = \frac{1}{n}\sum_{j=1}^{n}(\phi_j^{(model)} - \phi_j^{(orig)})^2$$

(for n examples and ϕ_j is the score given to solution j) often leads to poorly ranking the best meta-model, since lowering MSE *does not necessarily* lead to choosing better solution. Here the error is calculated by comparing the fitness value given to an individual by the meta-model and the fitness value given by the true fitness function. A more effective method of evaluating meta-models is to evaluate how often the meta-model *correctly selects* individuals. The authors suggest several metrics for measuring how often a meta-model selects the correct individuals, but these are dependent upon being able to determine what are the correct individuals to select, a task that is not possible within either the maze domain (except when exhaustive search is possible) or MR ADORE

4.3 Genetic Algorithms versus Simulated Annealing

Since we use genetic algorithms and simulated annealing in our experiments, we reviewed their comparison in the literature.

The performance of genetic algorithms and simulated annealing has been compared in mapping processes in parallel programming [Talbi and Muntean, 1993]. The authors also use hill-climbing as a third method to solve the problem. Mapping processes involves determining the optimal static placement of communicating processes on the processors of a distributed memory parallel machine, an NP-complete problem [Garey and Johnson, 1979]. Simulated Annealing and Genetic Algorithms show similar performance, with GAs having a search time comparable to hill-climbing (faster than simulated annealing). The intrinsic parallelization of genetic algorithms is shown to give them an advantage over simulated annealing.

Chapter 5

Heuristic Search with Meta-Models Method

5.1 Novel Approach

We now describe in detail our novel approach to solving the action set selection problem in Markov Decision Processes.

As mentioned in the Existing Methods section, wrapper approaches use the correct optimization criteria but can be prohibitively expensive. This is because the actual system is invoked in order to evaluate all potential solutions encountered during the search. Filter approaches are computationally feasible but do not necessarily deal well with complex interdependencies among operators since they have no access to performance of the actual system. We combine the best of wrapper and filter approaches by using a wrapper-like search in the space of action sets. Unlike traditional wrapper methods, we guide the search with a *fast* but at the same time *domain-specific* fitness function (meta-model). Similar approaches have been used in [Jarmulak and Craw, 1999]

We call our method Heuristic Search with Meta-Models. We will refer to this method as HSMM for the remainder of the thesis. HSMM involves four major steps which operate as follows:

Step 1: we evaluate a small collection of selected action sets via running each of them on the actual MDP system as shown in Figure 5.1. Each action set A, is assigned a reward and a cost, based on the average reward and cost over several runs of the system using it. The action set's *fitness* $F(A)$ is then $F(A) = \alpha R(A) - \beta C(A) + \sigma$ (as defined in Definition 3.5). The number of random action sets evaluated depends upon the amount of time available for the training process and the number of actions in the set.

Step 2: step one results in a collection of action sets and their fitness values $\{A, F(A)\}$. In the second step, we generalize this collection using machine learning (ML) techniques (Figure 5.1). As a result, an approximate fitness function F_{ML} is acquired.

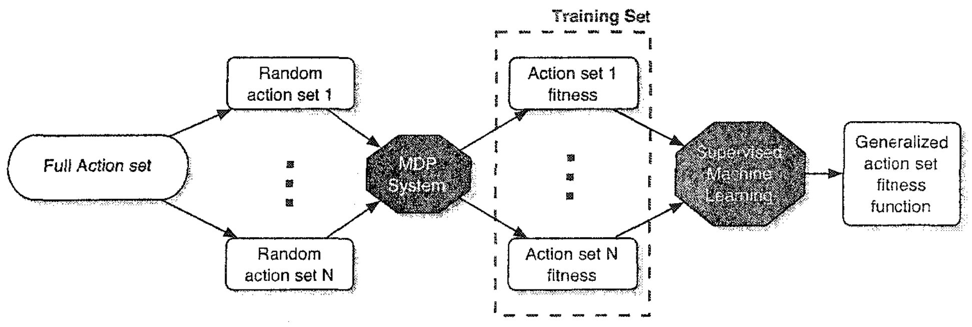


Figure 5.1: Steps 1 and 2 of the proposed methodology. Supervised machine learning methods are used to generalize fitness of sampled operator sets into an approximation to the actual fitness function.

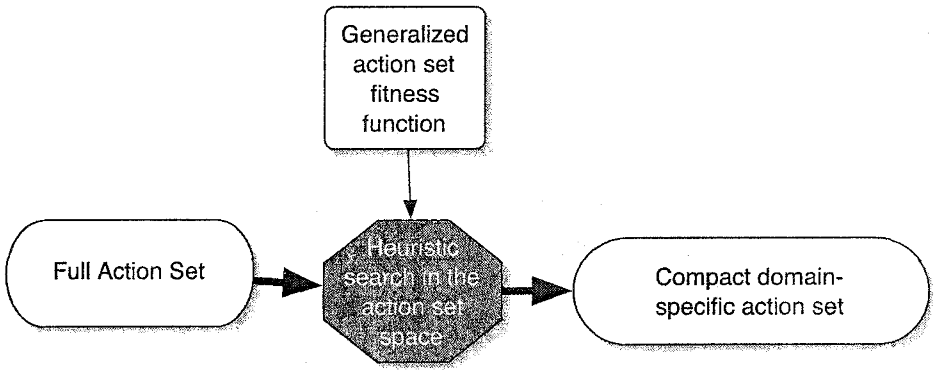

Generalized action set fitness function
Full Action Set
Heuristic search in the action set space
Compact domain-specific action set

Figure 5.2: Step 3 of the proposed methodology. Our approach for automated operator selection: heuristic search is conducted in the space of operator sets. It is guided by a machine-learned approximation to the performance function of the actual system learned in steps 1 and 2.

Step 3: once machine learning is over, we use the approximate fitness function as the optimization criteria in a heuristic search (e.g., genetic algorithms and simulated annealing (SA)) in the space of action sets (Figure 5.2). Numerical classifiers provide the search mechanism with actual fitness values. Symbolic classifiers label an action set with a class, which can be used in the same way as a fitness value by the search mechanisms, by simply choosing action sets labeled with the highest class, since these classes are defined as buckets for fitnesses (i.e class *12* could be for fitnesses between 0.2 and 0.25, class *13* could be for fitnesses of between 0.25 and 0.3, and so on...).

Step 4: the action sets found by the search are then evaluated against a set of validation images. The best m action sets are output to the user to be used in the domain of interest.

In all experiments, we compare the performance of the MDP agents with the action sets chosen by the proposed method to the performance of the MDP agents with action sets chosen by the following standard techniques.

First, we use the Top method (detailed in Algorithm 2 in Chapter 4) to choose action sets to test whether a wrapper/filter approach is necessary. We use the same training data as is used to train HSMM, but use it instead to *rank* actions. An action's score ω is defined as the average *fitness* (Definition 3.5) of the action sets in which it was present. This treats operators as individuals, abolishing the assumption that operators must be considered as a *team*, and that certain operators are useless in the absence of other operators (such as operators that work with grayscale images in the absence of an operator that converts colour images to grayscale images). The method lacks the knowledge of how many actions to put in its chosen sets.

Second, we compare against randomly generated operator sets and third, we compare against a set chosen by a domain expert. In the vision domain, this expert has three years of expertise using the MR ADORE system on forestry data. Since there is no domain expert in the maze domain, we compare against an action set with one half of the actions removed, but all the general movement directions still present.

It is important to note that if the full action set is the best set for a given agent and domain, then the HSMM method is capable of choosing this set. That is, HSMM does not *necessarily* reduce the action set, it only does so if a reduced set has a better fitness than the full set.

It should also be noted that our heuristic search methods are not required to perform multi-objective optimization, since the reward $R(A)$ and the cost $C(A)$ (C_l+C_{rt}) are already incorporated into the fitness $F(A)$ before the meta-models are trained. Thus the search methods are required to optimize one objective, the fitness of an action set.

5.2 Algorithms

We will now present the search algorithms used in step 2.

5.2.1 Heuristic Search Methods

Recall that the number of subsets of an action set of size N is 2^N, an intractable number for exhaustive search, even for modest values of N. In order to intelligently explore the space of action sets we turn to well-known heuristic search methods. Heuristic search is known to have a loose requirement on gradient information and better global searching ability [Yan and Minsker, 2003].

Genetic Algorithms

Genetic Algorithms (GAs) are a vital part of our HSMM method. As such, we present an brief overview of GAs, specific to the type of GA we use in our experiments.

Introduced in the 1960s by John Holland of the University of Michigan [Holland, 1962], and greatly furthered by David Goldberg in the 1980s [Goldberg, 1989], genetic algorithms are a method for parallel search which use biological abstractions to represent data. Data is encoded into a bit-string called a *chromosome.* A position in the chromosome can be either active (represented by a '1') or inactive (represented by a '0'). An initial *population* of random chromosomes is generated and ranked according to each chromosome's *fitness*. The fitness function is the most important part of any genetic algorithm in that it must accurately rate the chromosomes according to some given criteria.

Once the initial chromosomes are ranked, evolution begins. In simple GAs, to create the next generation of chromosomes, two *parents* are chosen from the present pool. A *flip* function selects the parents in such a way that it is more likely to pick higher ranking parents, but will sometimes pick lower ranking parents.

Once parents are selected, they are *mated* to form *children.* Two parents form two children, with the children being formed as opposites. This is done by giving *child1 parent1*'s qualities half the time (and giving *child2 parent2*'s qualities) and the opposite the other half of time (giving *child1 parent2*'s qualities,etc . . .). This is called uniform crossover, and is used when the order of the bits in the bitstring is not important. Giving qualities of a parent is implemented by assigning the bit in the child active if it is active in the parent, or inactive if it is inactive in the parent.

Once the children are formed, they are mutated by flipping random bits. This is done to prevent a population from having individuals that are too similar, which could lead to convergence to a local maximum. This is also the reason why it is not always the case that the top-ranked parents are selected for mating. This is much the same as allowing exploration in maze-type applications; we may find a better solution by taking seemingly sub-optimal steps in the present because they may lead to improvements in the future. The probability with which the children's' bits are flipped during mutation is determined by the similarity of the parents that produced these children. The more similar the parents, the greater the chance any bit is flipped.

The children chromosomes replace two other chromosomes in the population, which are selected in the same way as parents were, except that it is more likely that poorer ranking solutions will be chosen and replaced. Thus in every evolutionary step two new chromosomes are created and two are replaced. The population size is kept constant. It is an arbitrary matter to decide how many evolutionary steps to take (how many generations to produce), but generally after a few thousand iterations convergence to a maximum will occur.

For our specific application, chromosomes represent action sets. Each bit in a chromosome represents an action that can be either present or absent from any given set. Obligatory actions (such as the quit action in the maze domain) are included in every set and are not part of the chromosome, since we do not want to risk them not being in the set. The length of the chromosome is thus determined by the number of actions that we allow to either be present or absent in an action set. An action is present if the bit at its position is set to '1'. Otherwise (if the bit is set to '0'), the action is not included in particular set. For instance, bitstring '100101' represents an action set containing actions number one, four and six.

Simulated Annealing

Simulated Annealing is used as an alternative heuristic search in some of our earlier experiments and is described in this section.

Annealing is a process in which a solid is heated, and then allowed to cool. In order to maintain a satisfactory molecular arrangement throughout the process and guarantee a "good" end-state, the cooling must be slow, as this allows the arrangement to propagate through the solid.

Simulated annealing is the simulation of this thermodynamic system by a computer [Metropolis *et al.*, 1953]. Given a fitness function, which returns the suitability, the fitness of the system is calculated at every state. Given a current optimal solution $S_1$and a candidate optimum solution S_2, simulated annealing chooses to accept the candidate (i.e. replace $S_1$with S_2, given the probability in Equation 5.1:

$$P(accept) = \min\left(1, e^{-\frac{f(S_1)-f(S_2)}{kT}}\right) \tag{5.1}$$

where T is the system temperature and k is a constant. It should be clear then that if the candidate solution performs better than the current optimal solution it is always accepted. Otherwise, it is conditionally accepted, based on the current temperature of the system, and how much "worse" the candidate is than the current optimal solution. Thus, the probability of acceptance can be substantial, if the temperature is high. As the number of iterations increases, the temperature decreases, according to a manually-selected annealing schedule. The initial temperature is also selected manually.

There are numerous stop criteria for the simulation. The simulation may stop when a set number of iterations is reached, the temperature has become too low, there has been an "adequately long" time without finding a candidate that is better than the current optimum, etc. We used a pre-defined number of iterations as the terminating criterion.

This method is primarily an improvement on the hill-climbing algorithm, in which better solutions are always selected, and worse solutions are never selected. The simple hill-climber can fail when there are multiple maxima in the system; it may get trapped at a local maxima. Simulated annealing addresses this problem by allowing the system to back out, based on how much worse the candidate solution is, and the current temperature, as described above. The current best solution is always stored.

This is the general process of simulated annealing, which has been successfully applied to numerous combinatorial problems as a heuristic method [Kirkpatrick *et al.*, 1983]. We have now applied this technique to the problem of action selection. For this case, the function that is to be maximized is to find the action set (represented by a bitstring) that returns the maximum fitness, as determined by any one of the machine learning techniques described in Section 5.2.2.

At each iteration, a new candidate solution is created by taking the current optimal solution and randomly flipping a few bits in the string. This solution is then compared with the current optimal, and optionally accepted, as described above. The search terminates after the preset number of iterations. As a result, the best found bitstring is output to the user.

Both higher and lower probabilities of flipping bits have advantages. A higher probability allows the system to abandon less-than-optimal solutions more quickly. Lower probabilities mean that the candidate has more in common with the current optimal; more is learned from one iteration to the next. Clearly, if a probability of 0.5 is used, then nothing is learned from one iteration to the next, and the system is no more than a simple random search. Also, probabilities of 0.0 and 1.0 ensure that no real learning is possible, since either nothing changes, or everything learned changes at each iteration. Selecting the annealing schedule is a complicated task, often done by trial-and-error.

5.2.2 Machine Learning Algorithms

Any informed search mechanism requires an evaluation function in order to be able to score and rank individuals in the search. In order to obtain the true score for a candidate, invocation of the full system is required. If this invocation is too costly, the search can become intractable for a large search space. For this reason we turn to machine learning to *approximate* the true evaluation function within our heuristic search methods. These *meta-models* ([Jin *et al.*, 2001]) enable us to effectively search through the space of action sets. Different types of machine learning were used including symbolic and numeric learners, in an attempt to discover the best meta-model for each task.

Naïve Bayes Classifiers

As the name implies naïve Bayes is based primarily on Bayes theorem, which gives us a way of relating conditional probabilities to each other:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

where D is the training data and h is the hypothesis [Jensen, 1996; Mitchell, 1997].

In our case the attribute values are whether a given bit in the bitstring is on or off (1 or 0), and our target value is one of the discrete bucketed accuracy values. When a new instance is provided as input to the naïve Bayes classifier, it assigns the most probable target value to it, based on the attribute values describing the instance:

$$v_{MAP} = \underset{v_j \in V}{argmax} P(v_j | a_1, a_2, \ldots a_n)$$

where v_{MAP} is the most probable target value and $a_1, a_2, \ldots a_n$ are the attribute values of the instance.

Using Bayes theorem, we rewrite this equation as:

$$v_{MAP} = \underset{v_j \in V}{argmax} P(a_1, a_2, \ldots a_n | v_j) P(v_j)$$

With this equation we have two values to estimate. $P(v_j)$ is calculated simply by counting how often each target value occurs in the training data. Estimating $P(a_1, a_2, \ldots a_n | v_j)$ would require seeing the entire instance space, in order to obtain a reliable probability for how often these attributes appear together. The naïve Bayes classifier makes the simplifying assumption that each attribute a_i is independent of all other attributes, reducing the approach to:

$$v_{NB} = \underset{v_j \in V}{argmax} P(v_j) \prod_i P(a_i | v_j)$$

where v_{NB} is the target value output by the naïve Bayes classifier.

All the probabilities were trained using the same training data as the other methods. The difference is that no search is performed, instead we need only frequency counts from within the training data.

Artificial Neural Nets and Perceptrons

Artificial neural networks (ANNs) are a neurologically inspired function approximators. ANNs are built with many simple units, called *neurons,* that can accept many real-valued inputs, and output a single real-valued number. *Layers* are composed of one or more *neurons.* ANNs are composed of an input layer, optional hidden layer(s), and an output layer, as can be seen in Figure 5.3. Each neuron in a layer employs a *squashing* function, that maps a very large input domain to a small range of outputs. One common squashing function is the *sigmoid* unit that first computes a linear combination of its inputs, then applies a threshold to the result. The output o is computed as:

$$o = \sigma(\vec{\omega}.\vec{x})$$

where

$$\sigma(y) - \frac{1}{1 + e^{-y}}$$

$\vec{\omega}$ is the weight vector, and $\vec{x}$ is the input vector.

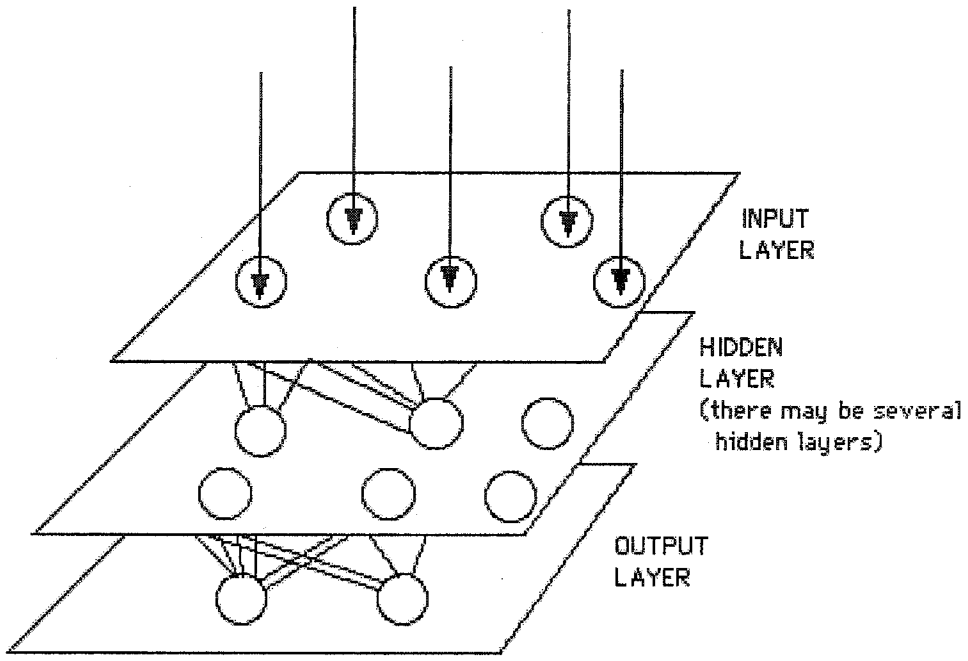


Figure 5.3: A basic artificial neural network [[University, 2004]].

A commonly used algorithm for training an ANN is the *backpropagation* algorithm [Haykin, 1994; Mitchell, 1997], which learns the weights for a multilayer network (input, hidden, and output nodes), provided that network has a fixed set of units and interconnections. All neurons are initially given a small random weight. In the *forward pass*, an instance is fed through the network, and the output is compared against the target output (provided in the training data).

A commonly used mathematical tool, *gradient descent*, is used to adjust the weights after errors are calculated. Errors are the difference between the predicted output produced by the neural network, and the actual output given in the training data:

$$E(\vec{\omega}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$$

where t_{kd} and o_{kd} are the target and actual output values associated with training example d and the kth output unit. The full backpropagation algorithm is given in Algorithm 4

ANNs accept real-valued inputs and output real-valued numbers as well. Scores and rewards in our experiments are (and generally will be) real-valued numbers, and thus not having to discretize these values should give the ANNs an advantage over other machine learners (such as decision trees), since discretizing may cause a loss of information when two different values are placed in the

Algorithm 4 Backpropagation algorithm. **Inputs**: (*training_examples*, η,n_{in}, n_{out}, n_{hidden}) ([Haykin, 1994; Mitchell, 1997]). **Outputs**: New weights for neurons

```
1: Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units and $n_out$ output units.
2: Initialize all network weights to small random numbers (e.g. between -.05 and .05).
3: while Not Termination Criteria do
4:   for each $\{\vec{x}, \vec{t}\}$ in $training_examples$ do
5:     Input the instance $\vec{x}$ to the network and compute the output $o_u$ of every unit $u$ in the network
       (Propagate the input forward through the network).
6:     For each network output unit $k$, calculate its error term $\delta_k$
           $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
7:     For each hidden unit $h$, calculate its error term $\delta_h$
           $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} \omega_{kh} \delta k$
8:     Update each network weight $\omega_{ji}$
           $\omega_{ji} \leftarrow \omega_{ji} + \triangle\omega_{ji}$
       where
           $\triangle\omega = \eta \delta_j x_{ji}$
       (Propagate the errors backward through the network)
```

same *bucket*.

Perceptrons are a special type of ANN that are composed of an input layer and an output layer, with no hidden layer(s). They are thus the equivalent of a linear separator, and are often too simple to properly partition the complicated space of action sets.

Decision Trees

A decision tree approximates discrete-valued target functions in a tree representation [Mitchell, 1997; Quinlan, 1993] At each node in the tree, a decision is made by testing a condition on a single input *attribute*, and the corresponding decision branch is followed to the next node in the tree, with the process continuing at the next sub-tree. This continues until we reach a *leaf node*, which contains a constant value returned by the decision tree. Decision trees can be represented as a set of *if-then* rules, for human readability. They can also be represented as a disjunction of conjunctions of the form:

$$(X = a \wedge Y = b) \vee (Z = b \wedge W = c)$$

Decision trees are considered appropriate for problems where instances are represented by attribute-value pairs, which is the case in our task. Each *action* is represented in a bitstring, taking on values $\{0,1\}$. Even if the number of disjoint possible values for an attribute is not two, decision trees work

Algorithm 5 *Generate_decision_tree.* Generate a decision tree from the given training data ([Quinlan, 1993; Han and Kamber, 2001]). **Inputs**: samples, attribute-list. **Output**: decision tree.

```
1: create a node $N$
2: if $samples$ are all of the same class $C$ then
3:     return $N$ as a leaf node labeled with class $C$
4: if attribute-list is empty then then
5:     return $N$ as a leaf node labeled with the most common class in $samples$
6: select test-attribute, the attribute among attribute-list with the highest information gain
7: label node $N$ with test-attribute
8: for each known value $a_i$ of test-attribute do
9:     grow a branch from node $N$ for the condition test-attribute = $a_i$
10:    let $s_i$ be the set of samples in $samples$ for which test-attribute = $a_i$
11:    if $s_i$ is empty then then
12:        attach a leaf labeled with the most common class in $samples$
13:    else
14:        attach the node returned by Generate_decision_tree($s_i$,attribute-list − test-attribute)
```

well as long as this number is not excessively large. Even real-valued attributes can be handled, with slight modifications to the algorithm (i.e. discretizing). In our work, the output values are fitnesses that take on real values. We thus discretize the data into *buckets*, limiting the number of possible output values.

A well-known decision tree induction algorithm is *ID3* (and subsequently *C4.5* and *C5.0*) that uses *information gain* to decide which attribute to test at each node. Information gain depends greatly upon a measure called *entropy*:

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i log_2 p_i$$

where S is a collection of examples, and p_i is the proportion of the examples belonging to class i. The information gain of an attribute $\mathcal{A}$ relative to $\mathcal{S}$ is defined as:

$$Gain(\mathcal{S}, \mathcal{A}) = Entropy(\mathcal{S}) - \sum_{v \in Values(\mathcal{A})} \frac{|\mathcal{S}_v|}{|\mathcal{S}|} Entropy(\mathcal{S}_v)$$

where $Values(\mathcal{A})$ is the set of all possible values for $\mathcal{A}$ and $\mathcal{S}_v$ is the subset of $\mathcal{A}$ that has value v. The *ID3* algorithm ([Quinlan, 1993]) is given in Algorithm 5

Decision Lists

A decision list is an ordered set of rules where each rule has a conjunction of possibly negated literals as the precedent and a Boolean class value as the antecedent. The term k-DL is used to refer to decision lists that contain a maximum of k literals per conjunction. Compared to decision trees, decision lists have more complex tests in their nodes, but a simpler overall structure. For any given k, a k-DL is more expressive than a decision tree of depth k [Rivest, 1987]. Decision lists are known to be polynomially efficient, both in terms of examples required, and computation time .

It is possible to construct decision lists which return a Boolean value - true if the instance returned the maximum accuracy (bucketed), otherwise false. This results in a much shorter decision list than if we were using a decision list that returned an integer value. It is possible to interpret this shorter list and manually design a few decision list test sets which are optimal in terms of number of operators, and return the maximum accuracy.

Chapter 6

Empirical Evaluation

In this section, we first describe experiments and results on the maze domain, then do the same for the vision domain. A map of the empirical work is given in Table 6.4 to aid the reader in navigating this section.

6.1 Maze Domain Experimental Setup

We use the maze domain in our experiments for several reasons. It is scalable and can be solved by brute force, which is part of the reason it is seen often in reinforcement learning literature (see Chapter 2). Also, since the experiments on the maze domain can be executed quickly (with respect to the vision domain experiments), changing properties of the mazes, policies, rewards and costs during experiments is not costly.

In the maze domain, we employ the four step process to search for the best action set as follows.

Step 1: We generate a small number of random subsets of navigation actions and evaluate them on randomly generated mazes. Each action set (A)'s fitness ($F(A)$) is defined as the average fitness

	Maze Domain	**Vision Domain**
Description	6.1	6.3
Experiments	6.2	6.4
Exhaustive Search	6.2.1	-
Simulated Online Experiments	-	6.3.1, 6.4.4
Operator Set Size Experiments	-	6.3.1, 6.4.5
Comparing Sequences and Sets	-	6.3.1, 6.4.6
Offline Policy vs. Online Policy	-	6.3.1, 6.4.7
Training with Online Data	-	6.3.1, 6.4.8

Table 6.1: Organization of Empirical Evaluation

Maze Domain Definitions	
$C_l(A)$	0 (no learning is done by the agent)
$C_{rt}(A)$	Number of Moves Performed by the Agent
$R(A)$:	Final State of the Agent (goal or non-goal)

Table 6.2: Mapping our definitions to the maze domain

obtained by the agent with this action set over all the training mazes. This will produce a set of tuples $\{A, F(A)\}$. The cost $C(A)$ and reward $R(A)$ of an action set are given in Table 6.2

Step 2: We generalize the tuples gained in step 1 by employing machine learning techniques, thus producing an approximate fitness function (or meta-model).

Step 3: We run a heuristic search over the space of action sets, using the meta-model produced in step 2 as the evaluation function.

Step 4: We validate the action sets produced by the heuristic search on a random set of validation mazes, different from those used in training. The best action sets are output to the user.

While evaluating each action set within the heuristic search is actually possible within this domain, we want HSMM to be portable to *any* markov decision process, and as such need to keep the method evaluation constant.

6.2 Maze Domain

To test HSMM on a non-vision domain, we chose a previously implemented and tested maze domain ([Bulitko *et al.*, 2003]).

In this MDP domain, an agent is placed in a typical maze setting, with a starting state, a goal state and walls (blocked states). A state in this MDP is the current location of the agent within the maze. An action is a transition from one location to another in the maze, with the exception of the *quit* action, with which the agent ceases to move.

The task at hand was to generate the optimal action set for an intelligent agent in the maze domain. With many actions available, the agent can take longer to find the goal, since many actions can be very similar, leading it to explore many similar routes, thus taking longer to solve the maze. We are interested in determining the *optimal* action set (A^*)for a given intelligent agent. The *optimal* action set is one which maximizes an agent's fitness ($F(A)$) over many mazes. The cost $C_{rt}(A)$ is based upon how many moves the agent takes, and the reward R is based upon whether the agent quits, solves the maze, or runs out of moves. There is no learning involved, thus $C_l(A) = 0$ and $C(A) = C_{rt}$. Table 6.2 summarizes these definitions.

The agent policy chosen for our initial maze experiments was an explore-exploit agent that nearly doubled the performance of a depth first search in a tournament of mazes. The agent follows a greedy approach which forces it to follow a complete path to the goal once one is known from the current location. Otherwise, the agent performs exploration with the goal of uncovering the maximum number of unknown locations. Further details are available in [Kovarsky, 2001].

The mazes generated in the experiments were all of size 200×200, with a density of 0.2 (meaning 20% of the maze cells occupied by walls). The goal and start states were randomly chosen, and the agent was given 1000 moves in order to solve the maze. Each move was given a reward of -1 for each unit traversed. Initially, we gave the agent a reward of 1000 when it reached the goal state and penalized it by 500 if it quit. In later experiments, solving the maze generated a reward of 100000, while quitting voluntarily resulted in a -50000 reward. This change was necessary to make solving the maze a much more desirable result than quitting quickly. Nevertheless, we include these results since the methods can still learn to quit if that is the best option.

In order to make the task of action selection non-trivial, we implemented an action set that allows the agent to move in a radius of size X. This creates a library of actions of size $8 * X$, in addition to the *quit* action. We chose a radius of size 5, giving the library 41 operators. This allows for 2^{41} possible subsets of actions for HSMM to search through.

Figure 6.1 shows the results of seven different methods using the initial reward scheme, all of which have been incremented by 5000 in order to make all the scores positive. The sets chosen by genetic algorithms (GAs) clearly outperform all other methods, but this is because they chose sets with very few actions (often just one), which forced the agent to quit early, thus gaining a small negative reward. The other methods chose sets with more actions (because they were forced to), which allowed the agent to explore further, but often not finding the goal, and thus gaining a high negative result. While this can be seen as a victory for the GAs (and thus HSMM), the domain itself was rather uninteresting.

In a more interesting schema, Figure 6.2 shows the results when we change the reward for solving the maze to 100000, and increase the penalty for quitting in a non-goal state to -50000. The hand-chosen set simply contains every second action in the full set. HSMM again outperforms other action selection methods here. The sets chosen by GAs generally contain more than 75% of the operators in the full set, so the full set does not seem to contain many redundant actions

6.2.1 Deliberation Cost Experiments

Penalizing a maze agent only for the moves it takes can be seen as too soft, since the agent's activities are not limited to actual movement actions. The agent must also evaluate *potential* actions, and this can be just as costly as making moves (much like in psychology where evaluating all possible choices can be more costly than making and executing the actual decision (Chapter 1). For this reason we

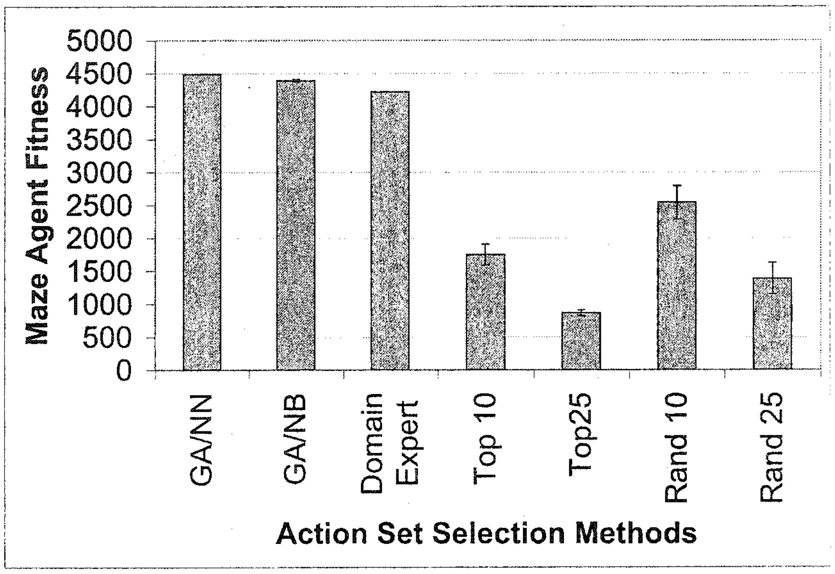


Figure 6.1: Average reward (incremented by 5000) for seven methods with a reward of 1,000 for solving the maze and -500 for quitting.

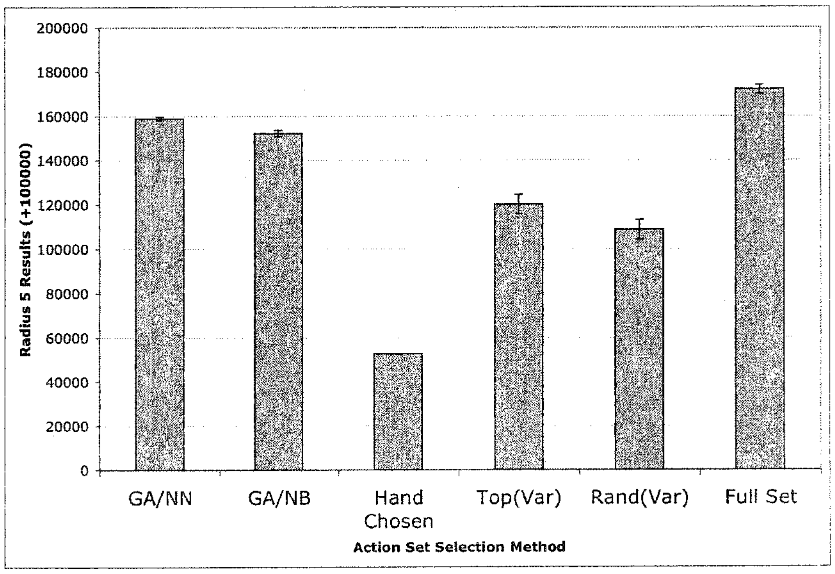


Figure 6.2: Average reward (incremented by 100000) for seven methods with a reward of 100000 for solving the maze and -50000 for quitting.

Revised Maze Domain Definitions	
$C_l(A)$	0 (no learning is done by the agent)
$C_{rt}(A)$	Deliberation Cost of the Agent
$R(A)$:	Final State of the Agent (goal or non-goal) - number of moves taken by the agent

Table 6.3: Revised definition mappings in the maze domain

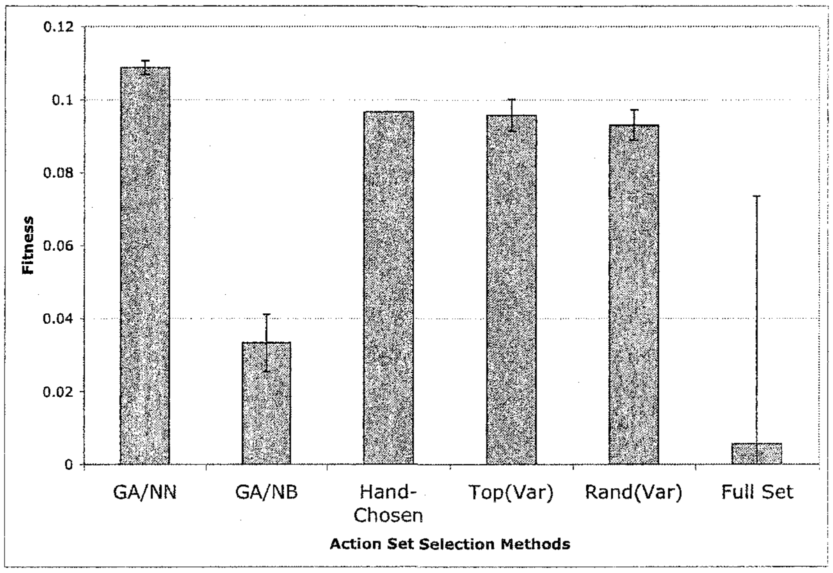


Figure 6.3: Average fitness of sets found by six different selection methods.

add a new cost to the fitness of an action set, being the *deliberation* cost which penalizes the agent for generating the potential state when applying a given action, and for searching to see if this state has been visited. For a depth-first search agent, the agent may not revisit states, since this would lead to infinite looping. The new mapping to costs and rewards is shown in Table 6.3

Our new reward measure $R(A)$ now takes into account the number of moves taken by the agent in reaching its final state, combining this cost with the reward given for the agent's final state. The cost $C(A)$ is the number of state evaluations done by the agent, whether or not it actually travels to the evaluated state. The fitness $F(A)$ is again defined according to Definition 3.5.

Using this new fitness metric, we again employed our HSMM method to choose action sets, and compared against the same methods as in previous experiments. When the agent finished in the goal state it was given a 10000 positive reward, if it quit in any other state it was given a negative 500 reward. Results are shown in Figure 6.3.

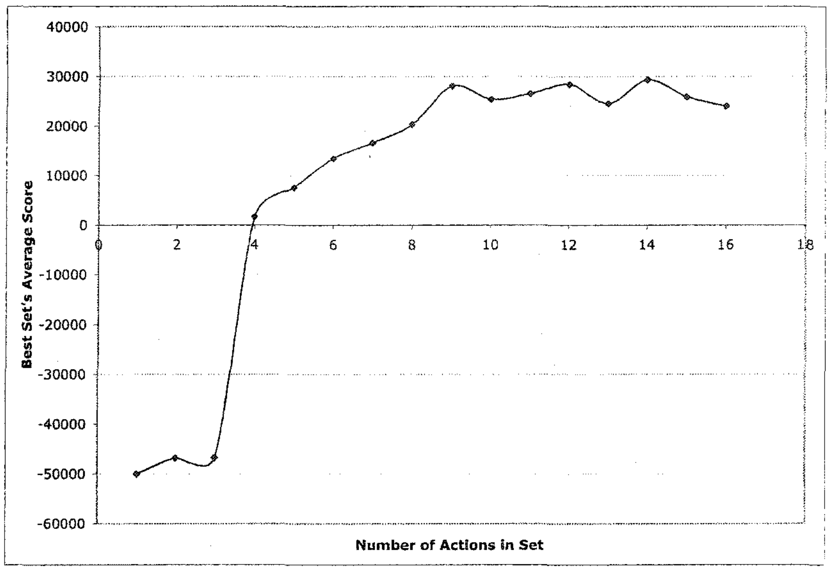


Figure 6.4: Results of an exhaustive search done over all possible action sets with radius set to 2. Note that the full set is *not* the optimal set.

Exhaustive Search

If we reduce the radius in which the agent can move from five to two, its action set is reduced to 16 actions. This gives us 2^{16} possible subsets, which is a number of sets we can exhaustively search through. Figure 6.4 shows the scores for the best action set of each size (1-16) averaged over 100 mazes.

6.3 Action Set Selection for Image Interpretation

Before describing the experiments in the vision domain, we briefly refresh the reader on the principles of MR ADORE.

MR ADORE has two modules, offline and online. In the offline module, full expansions of the vision operator set are performed on training images in order to train the control policy π, which is used in the online module. All legal operator sequences up to a limited length are applied to a given image. The resulting image interpretations are evaluated against the desired label, provided as part of the training data . Action-state rewards are then computed and used to obtain a value (Q) function.

In the online module, the learned control policy exploits the value function to interpret novel images. A "least-commitment" policy is used, first applying a full expansion of the vision operators to the novel image, then using the policy to decide which label to return to the user.

We performed several experiments within the MR ADORE framework. First we performed a pilot study on offline MR ADORE. Genetic algorithms and simulated annealing depend on several different parameters, and the pilot study was performed to determine what were the best range of values for each parameter. Next we tested our four-step HSMM method on MR ADORE and followed this with two sets of cross-validation experiments. We then performed several auxiliary experiments to test different theories with respect to operator set selection in MR ADORE.

In all our experiments, the full operator set contained 295 operators, of which three were necessary to ensure the proper operation of MR ADORE. This left a collection of 292 operators from which to choose our *optimal* operator set, which in turn provided us with a search space of 2^{292} operator subsets – a practically infinite number. The operators inserted in every set were $GrabImage$ (a loading operator), $SubmitImageGreen$, a submission operator that labels target pixels in green, and $RGB\_Segment$, an operator that ensures a path to a labeling is present in every set. The optional operators include *morphological filtering*, *histogram equalization*, and *thresholding*. A more detailed description of these operators is found in Appendix A

Operator sets were represented as 292-bit long bit strings. As usual, bit number n set to 1 indicated presence of operator n in the operator set, for both genetic algorithms and simulated annealing. In the genetic algorithms we used uniform crossover, since the operators are stored in a *set* and not a *sequence*, making their order unimportant.

All experiments were conducted on 72 images of young spruce plots maintained by the Alberta Research Council in Vegreville, AB. The images were captured in 24-bit colour at 256x256 pixels per image. A fragment of a typical image can be found in Figure 2.3

The HSMM method approach was applied to selecting a high-quality compact vision operator library as follows:

Step 1: we evaluate a small random collection of selected operator sets via running each of them with the actual system (MR ADORE) on a set of training images as shown in Figure 6.5. For each operator set o, all limited-length sequences of operators from o are applied to each training image. Each sequence is assigned an image interpretation accuracy for the image label it produces. The maximum image interpretation accuracy for all sequences from operator set o averaged over all training images is stored as $R(o)$. Each operator set also incurs a *cost* $C(o)$, which is a measure of the total time taken for the average full expansion with this set. This cost is the same for *learning* (C_l) as it is for *execution* (C_{rt}), since both of these activities perform full expansions. The operator set's *fitness* $F(o)$ is then $F(o) = \alpha R(o) - \beta C(o) + \sigma$ (as seen in Definition 3.5). These variables enable the user to control the trade-offs in a domain specific fashion. We define *MaxAccuracy* and *MaxCost* to be the accuracy and cost achieved by the full set of operators, averaged over the training

MR ADORE Definitions	
$C_l(A)$:	Full Expansions done in training the online policy
$C_{rt}(A)$:	Full Expansions done in executing the online policy
$R(A)$:	Image Interpretation Accuracy

Table 6.4: Mapping our definitions to the MR ADORE

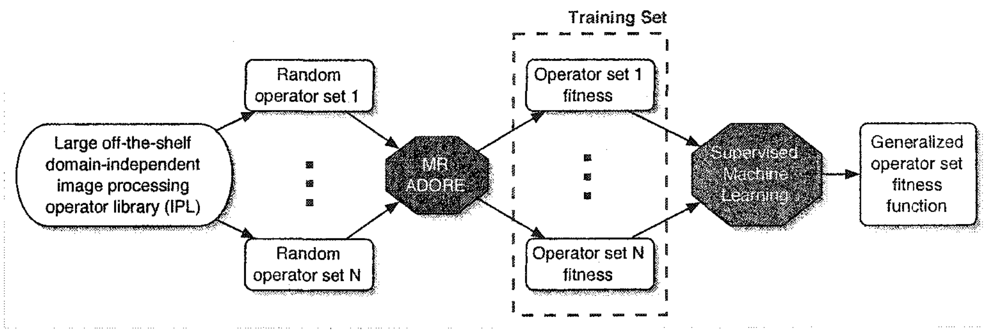

Training Set
Large off-the-shelf domain-independent image processing operator library (IPL)
Random operator set 1
Random operator set N
MR ADORE
Operator set 1 fitness
Operator set N fitness
Supervised Machine Learning
Generalized operator set fitness function

Figure 6.5: Supervised machine learning methods are used to generalize fitness of sampled operator sets into an approximation to the fitness criteria.

images.

Step 2: step one results in a collection of operator sets and their fitness values $\{o, F(o)\}$. In the second step, we generalize this collection using machine learning (ML) techniques (Figure 6.5). As a result, an approximate fitness function F_{ML} is acquired.

Step 3: we then use the approximate fitness function as the optimization criteria in a heuristic search (e.g., genetic algorithms and simulated annealing (SA)) in the space of operator sets (Figure 6.6).

Step 4: the operator sets found by the search are then evaluated against a set of validation images. The best m operator sets are output to the user to be used in the domain of interest.

6.3.1 Refining the Approach

After our first extensive experiments, we performed several further experiments in order to test the effect several phenomena noticed in our experiments with MR ADORE. These experiments are described here.

Simulated Online Experiments

Any policy π other than the optimal policy π will commit errors in its selection of actions in an MDP. We devised a set of experiments to simulate the amount of error in an online policy. We did this in an attempt to understand why, how and when the online module of MR ADORE chooses

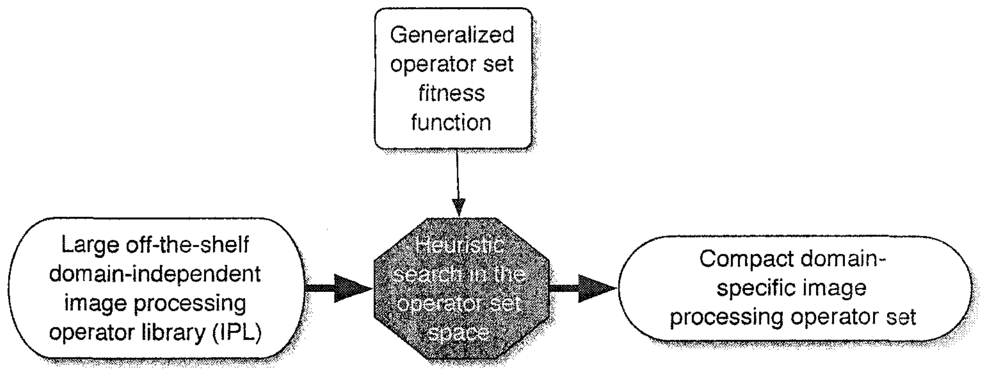

Generalized operator set fitness function
Large off-the-shelf domain-independent image processing operator library (IPL)
Heuristic search in the operator set space
Compact domain-specific image processing operator set

Figure 6.6: Proposed approach for automated operator selection: heuristic search is conducted in the space of operator sets. It is guided by a machine-learned approximation to the performance function of the actual *offline* system.

sub-optimal final labelings, given an operator set. We invoked the offline (oracle) module to test the operator sets chosen by each method, but introduced random choices of image interpretations to mimic how the online policy sometimes chooses the incorrect optimal image interpretation.

First, we ran the offline module (in the same way as before) with an ϵ chance of randomly choosing an image interpretation from the choices generated by an operator set, and thus a $1 - \epsilon$ chance of choosing the optimal image interpretation (since this is known offline). Henceforth we will refer to this model as ϵ*-perfect*.

Operator Set Size Experiments

The online module in MR ADORE is governed by a machine-learned control policy, which is trained by analyzing the data obtained from offline expansions. Having larger operator sets gives the control policy more possible operator sequences from which to learn in the offline module, but also more possible operator sequences to choose from in the online module. We investigated the effect of adding more operators to a set on the online performance of MR ADORE by incrementally adding operators to the domain-expert selected operator set, and running offline and online MR ADORE. The operators added were those ranked *worst* according to our filter selection method used in our experiments, using the same training data as in our cross-validation experiments. We added the worst operator not already present in the set 35 times (until no significant change was seen) and noted the online performance of MR ADORE. Adding the "worst" operator each time was our strategy to attempt to divert the online control policy from the path chosen using just the domain expert hand-picked set.

Comparing Sequences and Sets

The goal in our experiments is to discover operator sets which have *high* image interpretation accuracies ($R(o)$) and *low* cost ($C(o)$). Thus, an ideal way of evaluating such operator sets would be to compare their cost to the lowest cost necessary to produce equal average image interpretation accuracies. Since an exhaustive search of all operator sets is not possible, we determine the average cost of the *static* sequence leading to the best average results by observing how well each static sequence does within the full expansions available. This quantity tells us the minimum cost necessary to reach a given accuracy, as far as the best static sequences are concerned. We know this cost is the minimum since each sequence produces only *one* interpretation, so the online control policy has only one choice for which interpretation to return to the user.

Note that our methods are capable of choosing one of these operator *sequences* as an operator *set*, which would amount to picking one static sequence of operators to use at all times, eliminating all adaptability. As has been shown in [Levner *et al.*, 2003; Levner and Bulitko, 2004], MR ADORE's online module outperforms the best static sequence of operators available in the set in all experiments. Conversely, choosing a static sequence with a reasonably high average interpretation accuracy would likely maximize the fitness of a method, since the cost of a single sequence is negligible in comparison with the cost of a set of approximately 50 operators, since such a set would have many possible sequences of operators, greatly increasing the execution cost. Eliminating adaptability can be detrimental on further test images. A set that returns just *one* interpretation cannot recover if this interpretation is bad on a novel image. A set that returns several interpretations (and that exhibits good average image interpretation) has as an advantage that if one of its interpretations is not a good one, there are still more possible interpretations to return.

Offline (Perfect) Policy versus Online Policy

If operator set A outperforms operator set B *offline*, there is no certainty that A will outperform B online. This is because there is no guarantee that the machine-learned control policy will choose the best interpretation produced with set A. We attempt to determine which methods suffer the most loss in fitness/accuracy when switching from offline (perfect) policy to online (machine-learned) policy. Since the offline and online costs are the same, the change in fitness/accuracy is also the same.

Training with Online Data

Generalizing the fitness function sampled from offline performance may lead to an incorrect action set optimization criteria. We are interested in vision operator sets that will optimize the *online* MR ADORE module, and we thus should be generalizing the fitness functions with training data obtained from *online* MR ADORE activity (see Figure 6.7). Unfortunately, online training data takes much

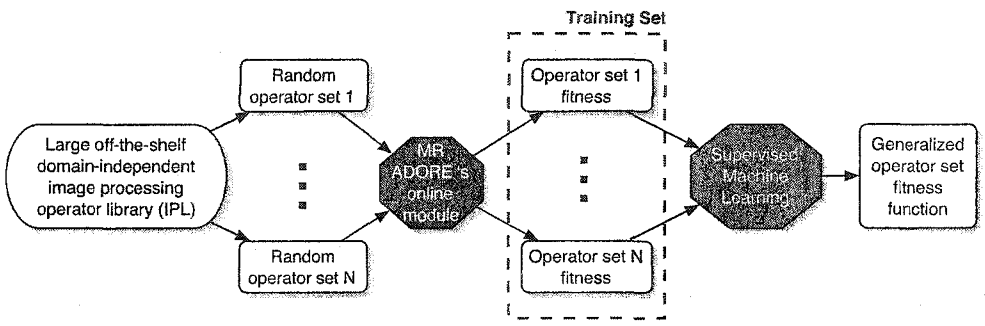

Training Set
Random operator set 1
Random operator set N
Large off-the-shelf domain-independent image processing operator library (IPL)
MR ADORE's online module
Operator set 1 fitness
Operator set N fitness
Supervised Machine Learning
Generalized operator set fitness function

Figure 6.7: Proposed approach for automated operator selection: heuristic search is conducted in the space of operator sets. It is guided by a machine-learned approximation to the performance function of the actual *online* system.

longer to obtain than offline training data. Indeed, a single offline training datum can be obtained in a matter of seconds, whereas obtaining one online datum can take upwards of one hour due to the extensive control policy learning process. Nevertheless, we gathered limited training data by running the MR ADORE online module with randomly generated operator sets. We then followed the same procedure as was used with the training data obtained offline, and tested the same methods.

6.4 MR ADORE Experimental Results

6.4.1 Pilot Study in the Image Interpretation Domain

Genetic algorithms and simulated annealing are known to be sensitive to parameter settings [Harik and Lobo, 1999]. In order to help choose a suitable set of parameter settings, a fitness function and a search method for our cross-validation experiments, we first generated 6986 sample operator sets (which took about two weeks), evaluated their fitnesses on a pool of 37 images (step 1), and used them to train our machine-learned fitness approximators (step 2). The learning was done with decision trees (DT), artificial neural networks (ANN), and naïve Bayes (NB). Note that in order to make decision trees and naïve Bayes applicable we had to discretize the real-valued operator set fitness values into buckets [Lee *et al.*, 2003]. We used decision lists in some experiments, but found the results of the search with decision trees and decision lists similar enough to eliminate the latter, thus speeding up our experiments.

We then ran GAs/SA with every combination of five population sizes (for the GAs) {30, 50, 100, 200, 500}, four iteration numbers {100, 500, 1000, 10000}, three mutation rates {0.05, 0.1, 0.2} and three fitness functions {NB, NN, DT} on the 37 images, and computed true fitness values for each set (steps 3-4). In total, GAs produced 900 operator sets, since we performed five runs at each

Methods and Data Used	
Number of training data:	6986
Number of mages	37
Selection methods used:	Genetic Algorithms and Simulated Annealing
Fitness functions (meta-models) used:	Decision Trees, Artificial Neural Networks, Naïve Bayes
Search Method Parameters	
Populations (GAs only)	30, 50, 100, 200, 500
Iterations	100, 500, 1000, 10000
Mutation (Flip) Rates	0.05, 0.1, 0.2;
Experimental time	30 days

Table 6.5: Pilot study methods and parameters

parameter setting. The SA produced 180 operator sets. The entire process took about one month. These results are summarized in Table 6.5

Genetic algorithms outperformed simulated annealing in this task for all tested parameter combinations. Figure 6.8 shows a typical comparison of performance between GAs and SA.

Decision trees proved to be the most reliable operator set fitness approximator (holding all other parameters in the GAs/SA constant), as Figure 6.9 demonstrates. The decision tree fitness function was not always the best (at any given parameter settings), but there was no case where either the naïve Bayes or the neural net fitness function was statistically significantly better than the decision tree fitness function.

It should be noted that GAs/SA demonstrated a great robustness to changes in their control parameters. When the naïve Bayes fitness function was used, however, with a population size of 500, the fitness of the produced operator sets was decreased four-fold as compared to using a population size of 30. Seemingly, it took the search mechanisms too long to converge with larger populations, meaning the diversity of the population may have caused convergence to slow down. Other than this notable exception, it was generally true that more iterations, larger population sizes and smaller mutation rates made the GAs perform slightly better, so we trimmed our parameter settings accordingly. Despite simulated annealing's inferior performance, it was kept as a control, since genetic algorithms have been known to overfit data. Since there was no significant difference in performance with different fitness functions, we employed all three in the next set of experiments.

Initial Cross-Validation Study

The algorithm for our cross validation experiments is presented in Algorithm 6. Line numbers from this figure are used throughout the remainder of this section. All methods and parameter values are summarized in Table 6.6.

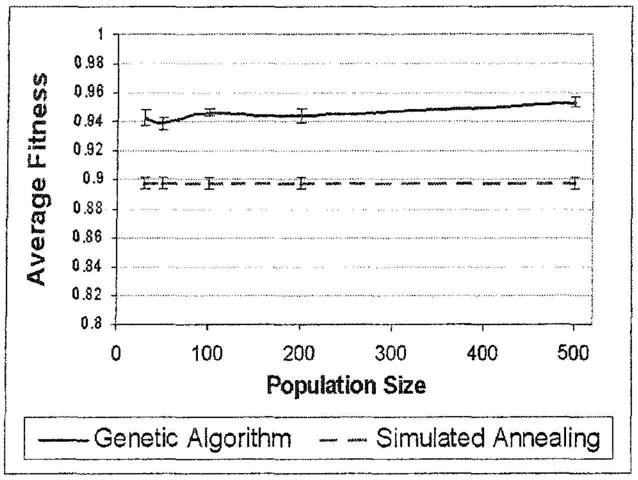


Figure 6.8: Holding the number of iterations and the mutation rate constant, genetic algorithms outperform simulated annealing with every GA population size. Standard deviation is shown as the error bars.

We first split our 72 images into three sets: training, validation, and testing (line 1). We then produced 10000 operator sets by choosing 50% randomly, and then including the complement of each set as well, to ensure all operators are present in half of the sets (lines 2-3). In step 1, we drew 1165 sample operator sets from this pool of 10000 operator sets. The sample operator sets were then evaluated on the 24 training images. All possible sequences up to length four were run on the 24 images for each sample operator set, which were then each assigned a fitness based upon their average fitness ($F(x)$) (line 4). This produced 1165 training pairs : (operator set, its fitness). This process took approximately 2.5 days.

For our fitness functions within the search methods, we again used decision trees, artificial neural networks, and naïve Bayes classifiers to generalize the fitness function sample set (step 2) (line 5). A pruning confidence factor of 65% was used for the decision tree, which was also boosted over 10 trials. The neural net had 100 hidden units and was trained for 10 folds of 750 epochs each, with its learning momentum set to 0.2. These parameter settings maximized the decision tree and neural net's accuracy on the training data. A standard naïve Bayes algorithm was used.

We used two search methods in step 3: simulated annealing and genetic algorithms. In genetic algorithms all 36 combinations of three population sizes {100, 200, 500}, two iteration numbers {1000, 10000} three mutation rates {0.05, 0.1}, and three fitness functions {NB, ANN, DT} were run. The simulated annealing was carried out in the same manner, but since there is no population in this method, there were only eight possible combinations of parameters (line 6). Due to the stochasticity of genetic algorithms and simulated annealing the search methods were run five times

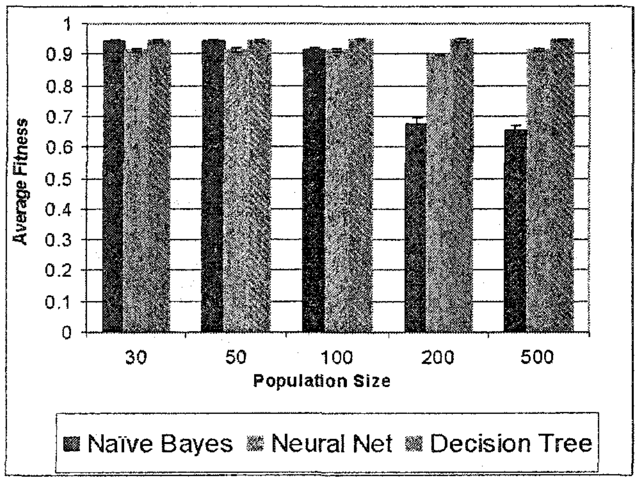


Figure 6.9: Decision tree fitness function proves to be the most robust to parameter variations. Error bars correspond to the standard deviation over the fitness over different sets produced during 5 runs of GAs with each set of parameters. Here we used a mutation rate of 0.05 and 1000 iterations.

with each set of parameters, producing different operator sets on different runs. We evaluated the top five ranked operator sets produced by each combination of parameter settings on 24 validation images (line 7).

We then evaluated the best operator set found in each of the possible $\{$GAs, SA$\} \times \{$NB,NN,DT$\}$ combinations by computing their true fitness values on the 24 test images (step 4, lines 8-9). Two additional operator sets were used: (i) randomly selected and (ii) manually designed by a domain expert. The cross validation process was executed over three folds, rotating the subsets of 24 images between training, validation and testing sets.

In our experiments image interpretation accuracy and execution cost were given equal weight. Thus, in our fitness equation (6.1):

$$f(o) = \alpha r(o) - \beta c(o) + \sigma \tag{6.1}$$

α was set to $\frac{0.5}{MaxAccuracy}$, β was set to $\frac{0.5}{MaxCost}$ and σ was set to 0.5 (to normalize the results to a range of $[0, 1]$). These values for α, β and σ are used in all of the vision experiments.

Evaluating the 3495 (1165*3) selected operator sets took approximately one week and training the neural net three times took approximately two days. Running the genetic algorithms and simulated annealing and evaluating the operator sets produced took approximately one week as well. All experiments were conducted on a dual Athlon 2600 processor with 2 GB of RAM.

Figure 6.10 shows the resulting fitness of the best operator sets found by each of the $\{$GAs, SA$\}$ $\times$ $\{$NB,NN,DT$\}$ combinations on the testing data. Genetic algorithms and simulated annealing show

Algorithm 6 An algorithm for heuristic search method of selecting operator. **Input:** Full domain independent operator set ops, Image/Labeling Pairs il, Heuristic Search methods hs, Machine Learning methods ml, (Empty) Sample Set (ss). **Output:** Domain specific operator set(s)

```
1: Split $il$ into three sets: training($tr$), validation($v$),testing($te$)
2: Randomly select operator subsets of $ops$, add these to $(ss)$
3: Select complement of each randomly selected set, add these to SS
4: Evaluate fitness ($F(x)$) of each operator set ($o$) in $ss$ on $tr$
5: Train ML with obtained $o$, $F(o)$ pairs
6: Search for best performing subset of $ops$ with $hs$, using $ml$ for fitness functions
7: Evaluate fitnesses of $hs$ selected subsets of $ops$ on $v$
8: Choose best subsets of $ops$ based on fitness on $v$ ($bv$) chosen with each combination of $hs$ and
   $ml$
9: Evaluate $bv$ on $te$
10: Output best operator sets in $bv$ based on fitness on $te$
```

Methods and Data Used	
Number of training data:	1165
Number of training images	24
Number of validation images	24
Number of testing images	24
Selection methods used:	Genetic Algorithms and Simulated Annealing
Fitness functions (meta-models) used:	Decision Trees, Artificial Neural Networks, Nave Baïyes
Search Method Parameters	
Populations (GAs only)	30, 50, 100, 200, 500
Iterations	100, 500, 1000, 10000
Mutation (Flip) Rates	0.05, 0.1, 0.2;
Experimental time	15 days

Table 6.6: Initial cross-validation experiment methods and parameters

almost identical behaviour, and both generally outperform a hand-picked set. The top combination of genetic algorithms and neural net produces on average an operator set that retains 93.3% of the image interpretation accuracy (reward) of the full operator set, while only incurring 5.5% of the cost.

6.4.2 Full Cross Validation study

After testing our cross validation method for three folds with the full $\{$GAs, SA$\} \times \{$NB,NN,DT$\}$ set of combinations, we needed to narrow our search in order to run enough folds to gain a further insight into the operator selection problem in MR ADORE. We thus eliminated simulated annealing from our experiments, since both SA and GAs exhibited very similar behaviour in the cross validation study, and GAs outperformed simulated annealing in some of our prior experiments (Section 6.4.1). We also eliminated decision trees, since they exemplified no dominance over any other machine-

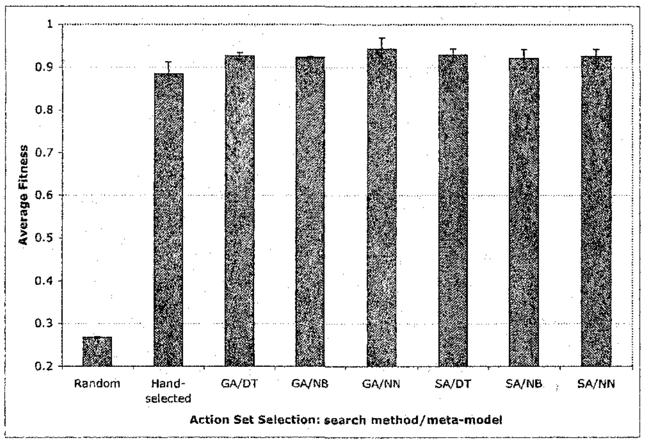


Figure 6.10: Comparison of the fitness of various action set selection techniques, on three folds of cross validation runs. The hand-picked set is constant over all folds.

Algorithm 7 An filter method, called "Top" modified to select operator sets. **Input:** Training data, desired number of operators d in set **Output:** Domain specific operator set(s)

```
1: for Each action $a$ in the full set do
2:    for Each training datum {attributes,fitness} do
3:       if $a$ is present in the training datum then
4:          Add this fitness to $a$'s total
5:          Increment count
6:    Calculate $a$'s fitness by dividing total by count
7: Sort actions by their fitness
8: Output top $d$ operators
```

learner, and were more difficult to train due to an interface with an external software module within our system. We reduced the set of possible population sizes to $\{100{,}500\}$ and the set of mutation rates was decreased to $\{0.5, 0.2\}$. The iteration set was also reduced to be $\{100{,}1000\}$. These values are summarized in Table 6.7.

We used the Top filter method of selecting operators to compare against HSMM (see Algorithm 7. We also compared against a set chosen by a domain-expert, a randomly selected set and the full set of operators. Both our filter method and the random method of selecting operators chose variably sized operator sets (changing with each fold), between 1 and 292 operators.

We ran 81 cross-validation folds with our new $\{\text{GAs}\} \times \{\text{NB,NN}\}$ set of combinations within our HSMM method. Figure 6.11 shows the offline image interpretation accuracy of each set selected in each of our six selection methods. Figure 6.12 shows the cost incurred by the sets chosen by these methods on offline MR ADORE. Using our predefined fitness evaluation function for operator sets, we combined the cost and image interpretation accuracy of each method. Results of this combination

Methods and Data Used	
Number of training data:	1165
Number of training images	24
Number of validation images	24
Number of testing images	24
Selection methods used:	Genetic Algorithms, Random Selection, Filter Selection, Full Operator Set, Domain Expert Selection
Fitness functions (meta-models) used (within GAs):	Artificial Neural Networks, Naïve Bayes
Search Method Parameters	
Populations	100, 500
Iterations	1000, 10000
Mutation (Flip) Rates	0.05, 0.2
Experimental time	90 days

Table 6.7: Full cross-validation experiment methods and parameters.

are shown in Figure 6.13. Note that the GA methods outperform all other methods.

6.4.3 Online Results

In the object recognition domain, we are most interested in the performance of a system on *novel* images, where the labeling is not yet known. In MR ADORE, the *online module* uses a machine-learned control policy to select a labeling for novel images.

We tested the online performance of the best performing GA with NN and GA with NB sets (according to testing on the validation data) on the test data for each fold. We also tested the performance of the set of Top (0-292) ranked operators as a comparison to a filter approach. In addition to these sets, we tested an operator set handpicked by a domain-expert, a set of (0-292) random operators (differing at each fold), and the full set of operators. In Figure 6.14 we see the online image interpretation accuracy of the six different methods of operator set selection. Since we are currently using the least-commitment policy in MR ADORE [Levner *et al.*, 2003; Levner and Bulitko, 2004], the online costs of any operator set is the same as the offline cost (which was shown in Figure 6.12). Using our predefined fitness function for an operator set, we show the fitness of each operator set selection method in Figure 6.15. Note that there is no statistically significant difference between the GA selected sets and the domain-expert selected sets.

The full operator set with 295 instantiated operators contained the following types operators: Load Image, Submit Image, RGB Segmentation, Convert Colour to Gray, FilterMedian, Gaussian Filter, Elliptical Erosion, Elliptical Dilation, Elliptical Closing, Elliptical Opening, Grayscale

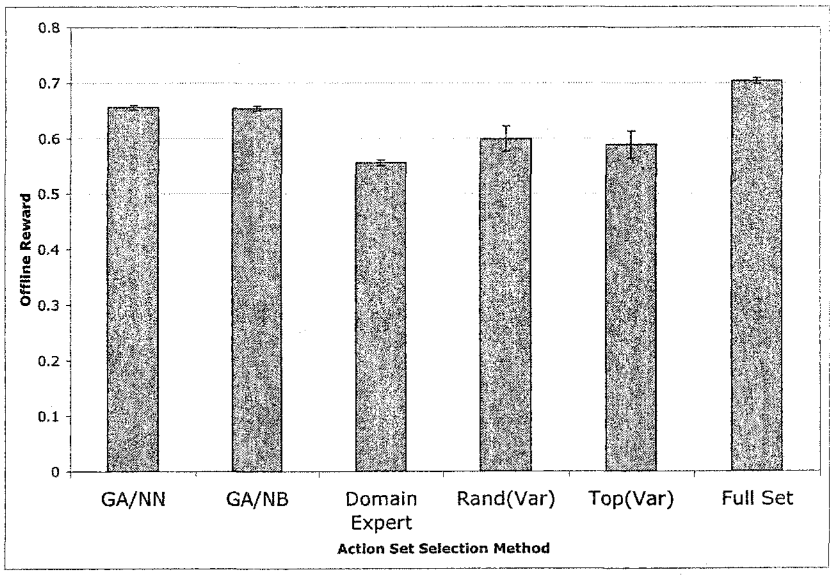


Figure 6.11: MR ADORE's offline image interpretation accuracy with six different operator set selection methods.

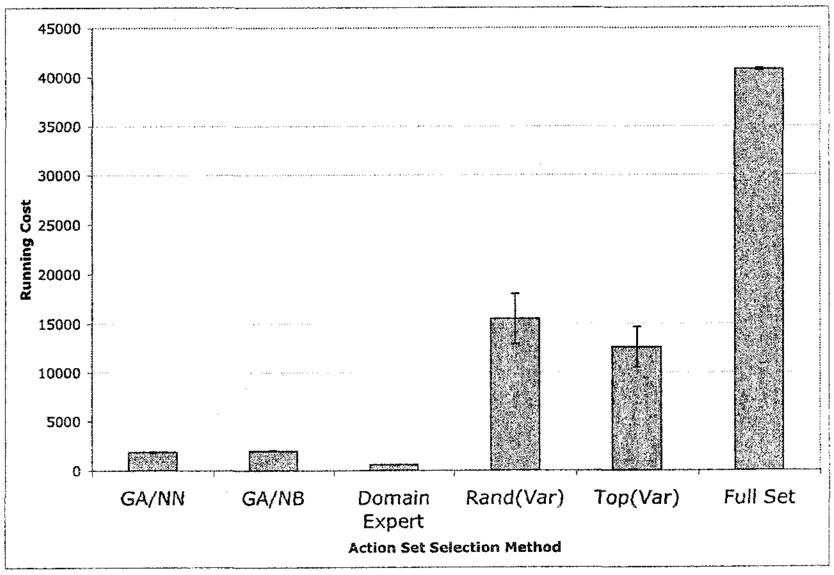


Figure 6.12: MR ADORE's offline (and online) cost with six different operator set selection methods.

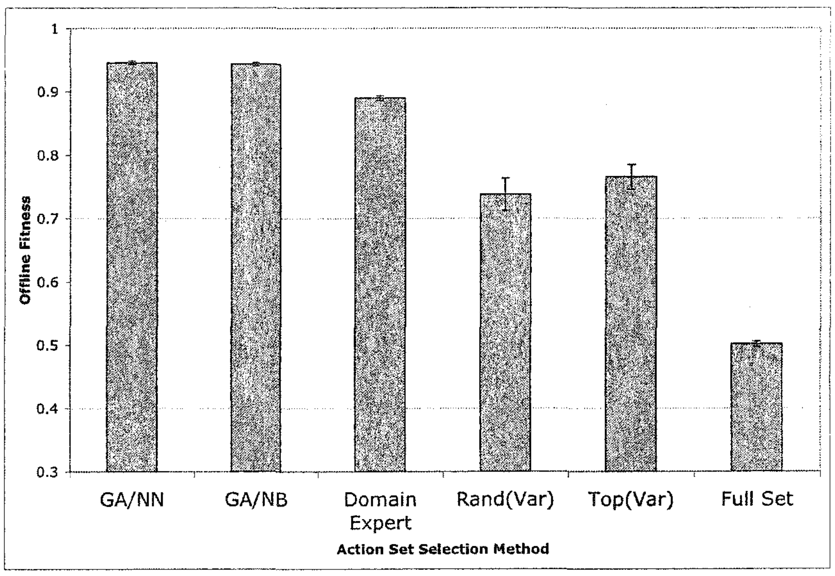


Figure 6.13: MR ADORE's offline fitness with six different operator set selection methods.

Thresholding, Binned Grayscale Thresholding, Probabilistic Thresholding, Colour Histogram Equalization, Probabilistic Histogram Equalization, Colour Stretching by Contrast, Coarse 3D Histogram Intersection, Pyramid Segmentation, Flood Filling, Peak Filling.

The best operator set (with respect to online fitness) found by HSMM contained the following types operators: Load Image, Submit Image, RGB Segmentation, Convert Colour to Gray, Gaussian Filter, Elliptical Erosion, Elliptical Dilation, Elliptical Closing, Grayscale Thresholding, Binned Grayscale Thresholding, Coarse 3D Histogram Intersection, Flood Filling, Peak Filling.

The domain-expert operator set contained: Load Image, Submit Image, RGB Segmentation, Convert Colour to Gray, FilterMedian, Gaussian Filter, Elliptical Erosion, Elliptical Dilation, Elliptical Closing, Elliptical Opening, Grayscale Thresholding, Binned Grayscale Thresholding, Probabilistic Thresholding, Colour Histogram Equalization, Probabilistic Histogram Equalization, Colour Stretching by Contrast, Coarse 3D Histogram Intersection, Flood Filling, Peak Filling.

While all these sets may appear to be similar, it is important to remember that they are instantiated differently. Thus each set can contain different (and multiple) versions of an operator. The full operator set, for instance, contains 93 different thresholding operators. The three sets listed contain 295, 52 and 29 total operators respectively.

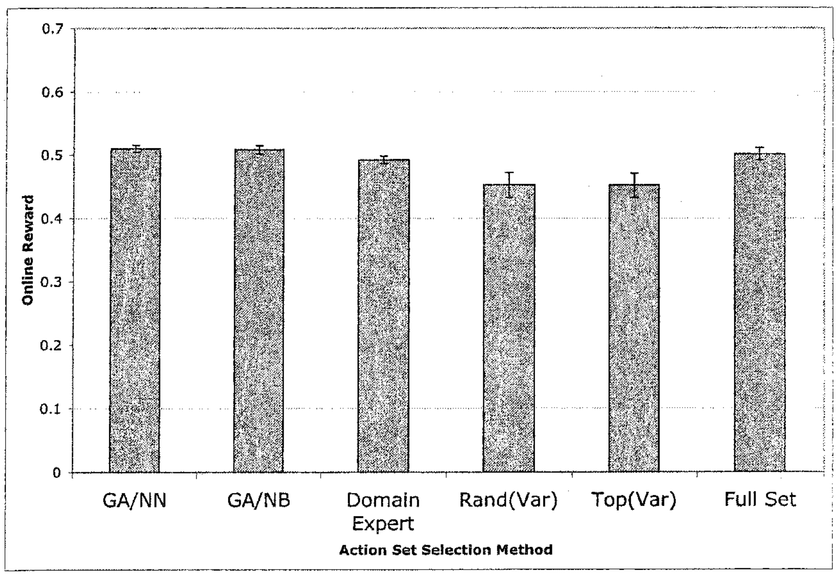


Figure 6.14: MR ADORE's online interpretation accuracy with sets chosen by GA/NN, GA/NB, domain expert, top (0-292) of operators, random(0-292) operators, and the full set of 292 operators.

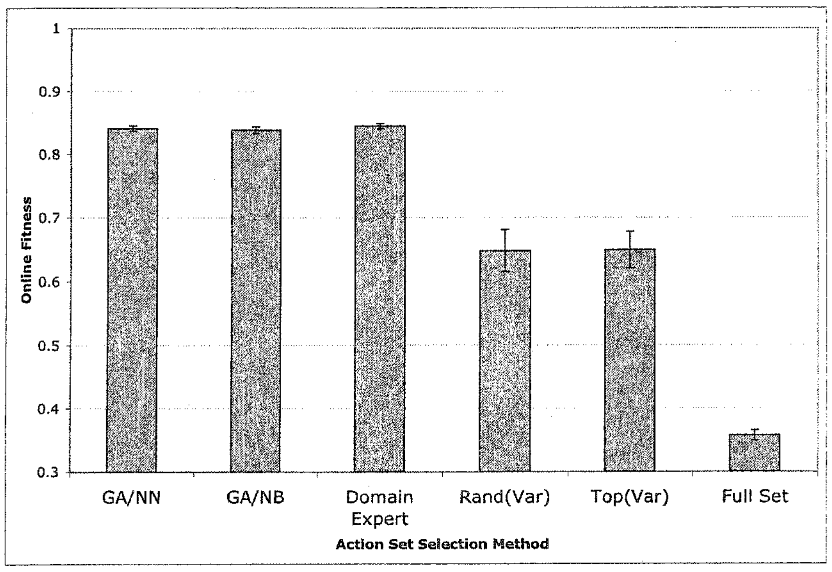


Figure 6.15: MR ADORE's online fitness with sets chosen by GA/NN, GA/NB, domain expert, top (0-292) of operators, random(0-292) operators, and the full set of 292 operators

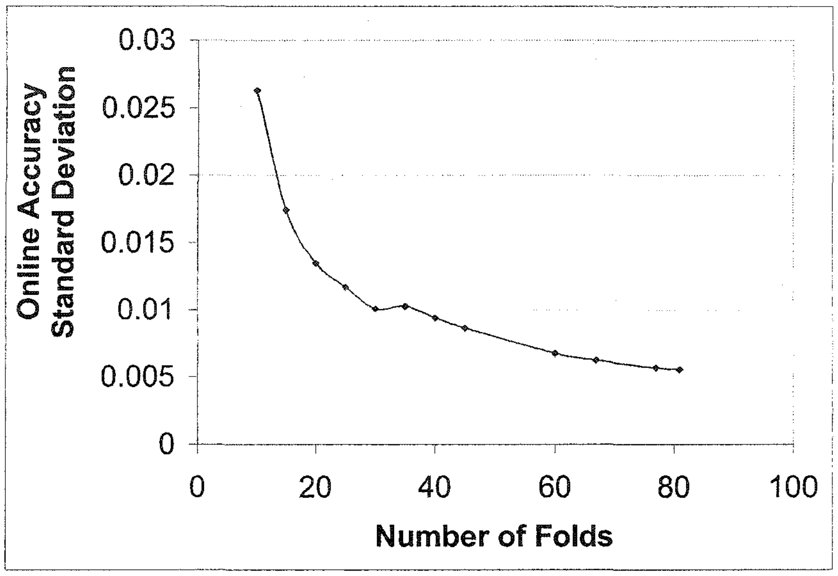


Figure 6.16: Change in standard deviation in online accuracy (for all methods) over 81 folds

Standard Deviation Plot

In order too visualize the decline in standard deviation over folds, we plot the change in standard deviation in Figure 6.4.3. As can be seen, the standard deviation falls steeply at first (as expected), but does not change greatly of the later folds. This trend suggests a higher validity of the reported results.

6.4.4 Simulated Online Experiments

We evaluated seven methods of choosing operator sets with MR ADORE running with the optimal offline policy, online policy and ϵ-optimal policy with $\epsilon = \{0.1, 0.3, 0.5, 0.7, 0.9 \text{ and } 1\}$ (recall that ϵ represents the amount of randomness in the policy, as detailed in Section 6.3.1) . Results are shown in Figure 6.17. Note that with the perfect offline policy, the GA/NN and GA/NB sets are the best for image interpretation accuracy, but that these methods experience a greater drop once randomness is introduced than the Top method with 50 operators. We fixed operator set sizes for the Top and random methods (50 and 100), in order to see if larger operator sets suffered more loss when the chance for a random choice was increased, and because HSMM typically chose operator sets with between 50 and 100 operators. As can be seen on the graph, operator sets with more operators tend to lose more image interpretation accuracy when random choice is added than do smaller operator sets. This finding seems to support the thesis that more choice can be detrimental with imperfect selection.

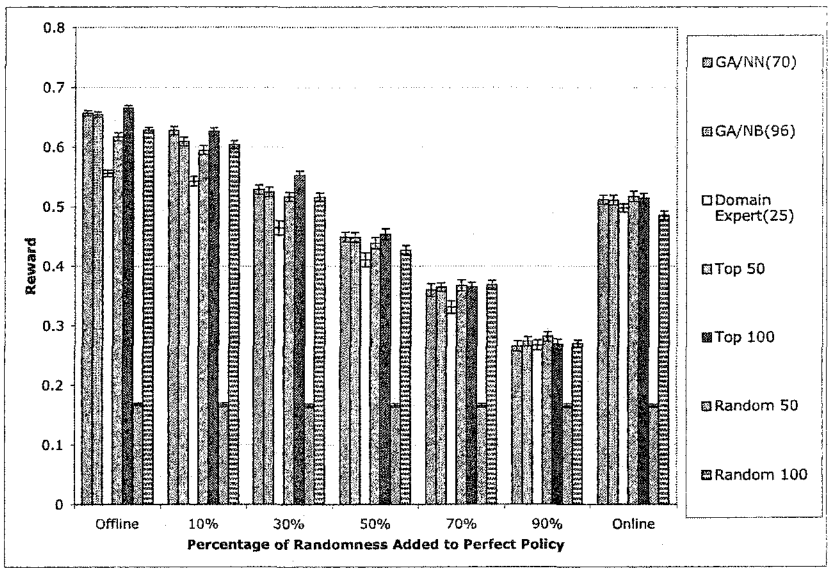


Figure 6.17: Seven different methods of operator set selection evaluated on different control policies. The numbers after the methods are the average (in parentheses) or fixed (not in parentheses) number of operators in the selected sets.

6.4.5 Operator Set Size Experiments

We plotted the online performance of MR ADORE with each of the 35 sets (generated by incrementally adding the "worst" ranked operators to the hand picked operator set) with the performance of MR ADORE with the ϵ-perfect policy, again with the 35 sets. Recall that we added the "worst" operators in an attempt to divert the control policy from choosing the optimal labeling (maximizing image interpretation accuracy), by giving it more sub-optimal labelings to choose from. The ϵ-perfect policy was run for 10 folds, while the online module was run for 9 folds. Results are shown in Figure 6.18. The online policy exhibits essentially unchanging behaviour with added operators. Additionally, we can see that MR ADORE's machine-learned control policy operates at about the same image interpretation accuracy as the 0.3-perfect policy.

6.4.6 Comparing Sequences and Sets

In order to determine the average cost of the sequence leading to the best average results, we obtained each sequence's fitness from the 72 full expansions (2278 sequences), and plotted them in Figure 6.19

Our methods are capable of choosing one of these operator *sequences* as an operator *set*, and the best set of this type would have generated an average online fitness higher than the mean of

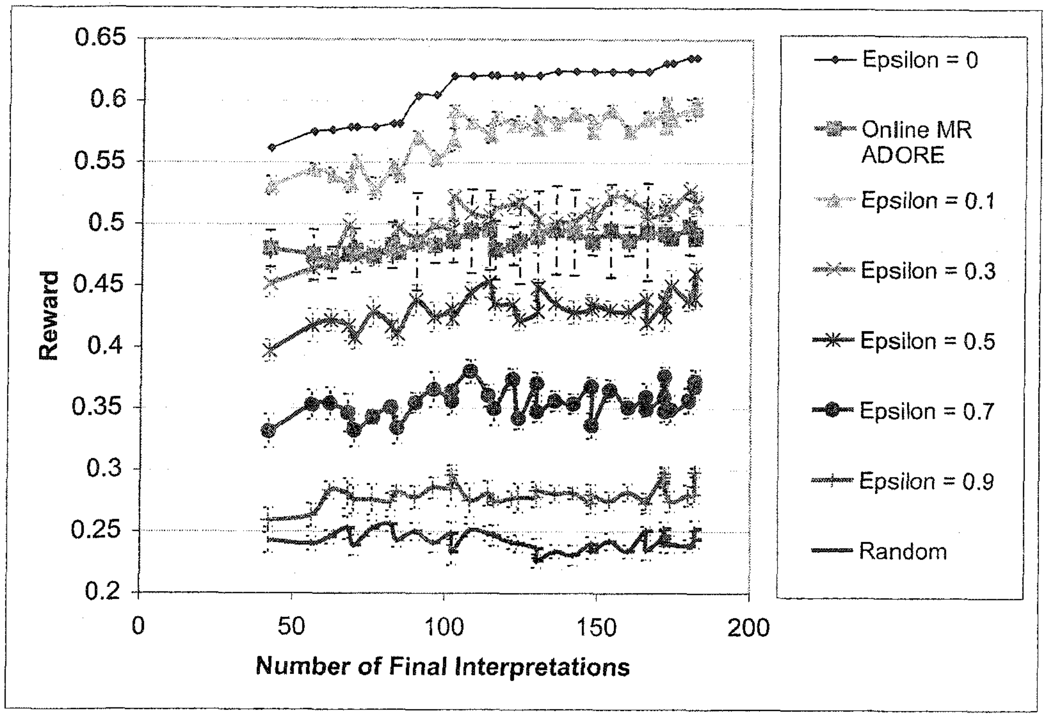


Figure 6.18: Performance of MR ADORE with 35 operator sets with the ϵ-perfect policy (for several ϵs), as well as with the machine-learned control policy.

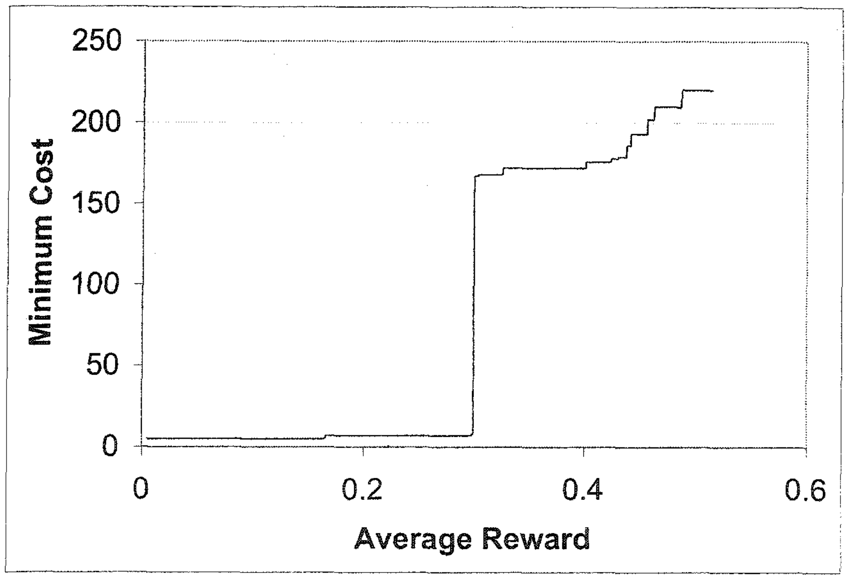


Figure 6.19: Minimum cost required to reach various accuracies with static sequences of four operators

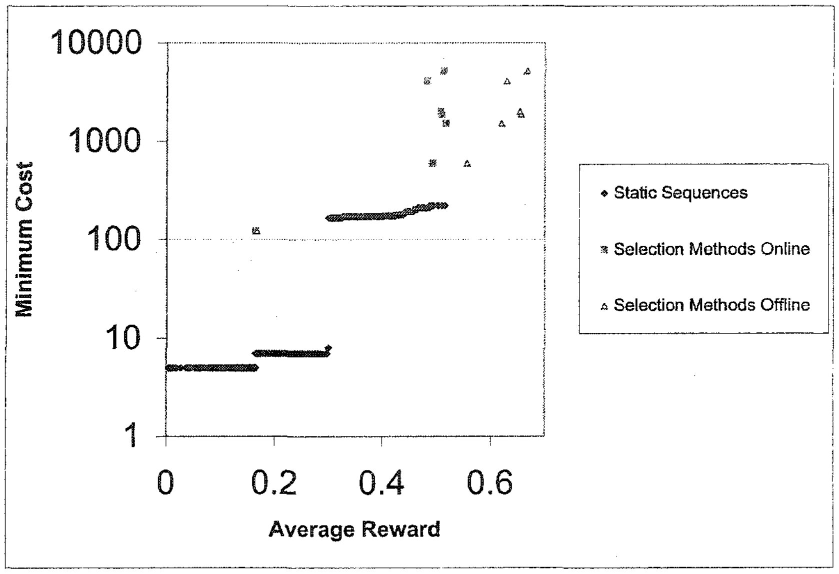


Figure 6.20: Average cost versus average image interpretation accuracy for all sequences of length 4 and our 7 methods, both offline and online. The squares are our methods with the online policy, the triangles are our methods with the offline policy and the filled diamonds represent the minimum cost required to reach a certain image interpretation accuracy.

any of the methods. In Figure 6.20 we compare our methods for selecting operator sets with static sequences of operators.

As can be seen, the operator *sets* average costs are much higher than the costs of the static sequences, while their average image interpretation accuracy is also higher. The square points represent the average online interpretation accuracy of selected operator sets, while the triangular points represent the average offline interpretation accuracy of the selected operator sets. Note that the offline and online costs are the same, due to the currently used least-commitment policy in the online module. It can also be seen in Figure 6.20 that the best static sequence of operators can easily be outperformed in accuracy, since the sets chosen by the methods have a higher average accuracy. This suggests that *adaptability* is necessary to produce higher quality image interpretations.

6.4.7 Offline (Perfect) Policy vs. Online Policy

In an effort to understand why some operator sets produce great results offline, but online experience a severe drop in reward, we graphed the change in accuracy and fitness when moving from the offline (perfect) policy to the online (machine-learned) policy.

Since cost remains the same, Figures 6.21 and 6.22 are essentially the same. One can see that the drop in accuracy and fitness is not uniform over all methods, and, perhaps more importantly,

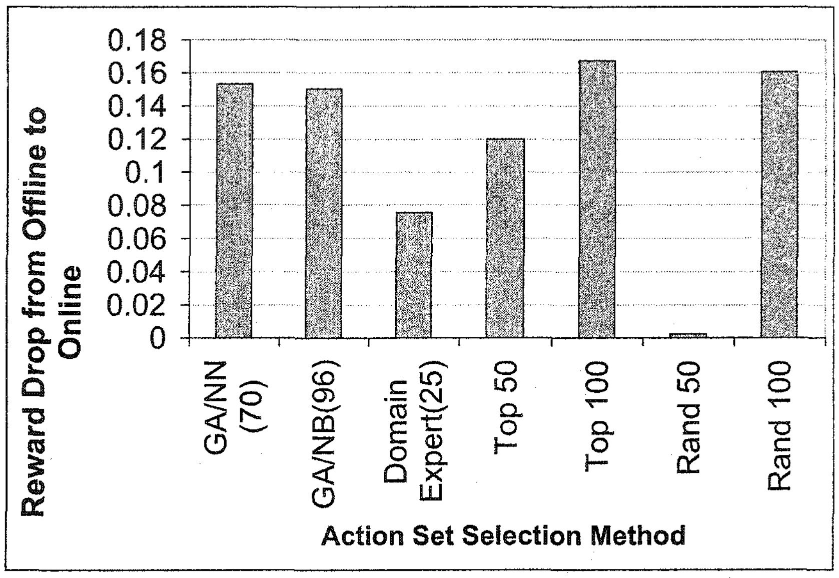


Figure 6.21: Drop in online accuracy (for all methods) over 81 folds

not uniform over different operator set sizes. Figure 6.23 shows how the average size of operator sets of a method corresponds to how much the operator sets generated by this method suffer online, on average. Perhaps more importantly, Figure 6.24 shows how the online policy degenerates in accuracy/fitness with an increase in possible image interpretations. Note, this is the *average* drop in accuracy/fitness versus the *average* number of possible image interpretations. These graphs seem to indicate that the online policy has difficulty when presented with more choices of interpretations, despite the fact that it is generated from more training data.

6.4.8 Training with Online Data

Recall that training the fitness approximators with data obtained from *offline* MR ADORE runs may lead to approximating the incorrect function - a function that optimizes the action set for use on the *offline* module of MR ADORE, where an oracle tells it which image interpretation is the best. Training with data obtained from *online* MR ADORE runs should help to approximate the correct function, but this data is much more costly to obtain than data obtained offline.

In Figure 6.25, we see the results after 16 folds with online training data used in our experiments. The Domain Expert set is independent of the training data, and performs the best out of the seven set selection methods. The GA/NN and GA/NB methods clearly outperform the filter and random methods trained with online data, but do not outperform the GA methods trained on offline data).

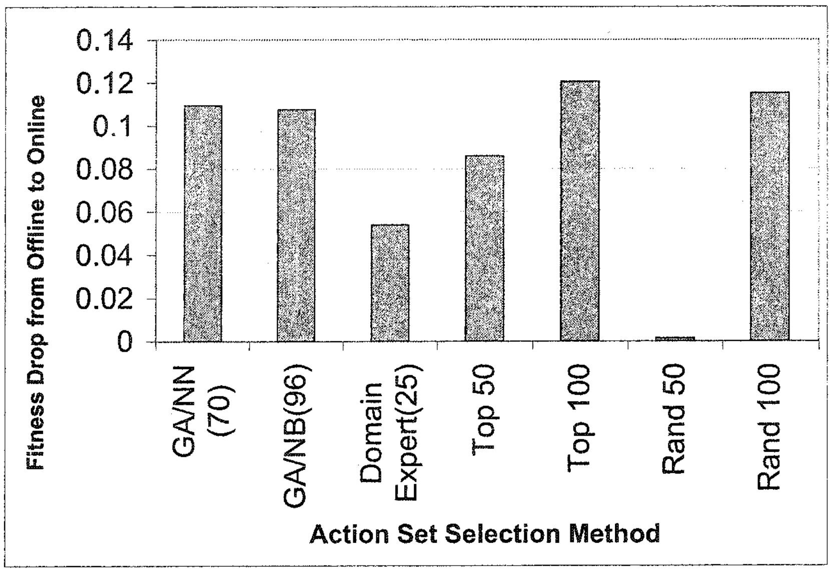


Figure 6.22: Drop in online fitness (for all methods) over 46 folds

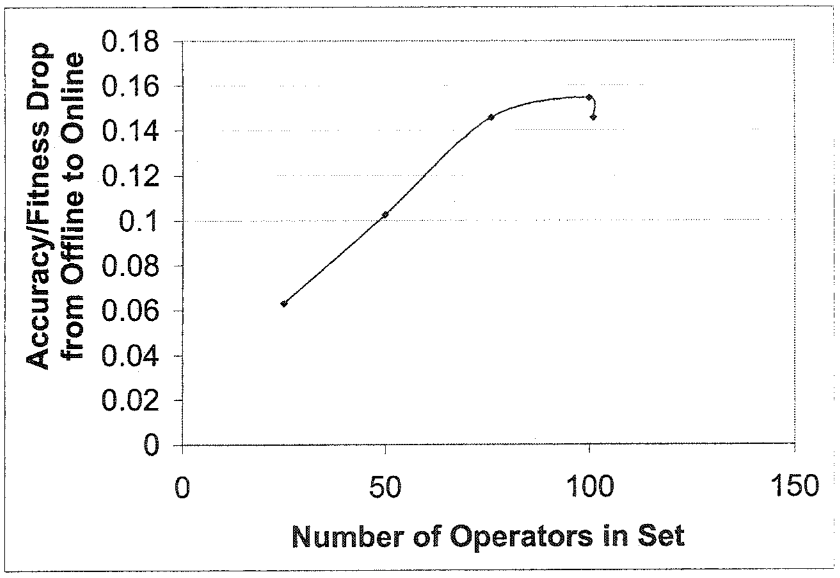


Figure 6.23: Drop in online accuracy/fitness vs average operator set size. The points left to right represent Domain-Expert, Filter50,GA/NN, GA/NB, Filter100.

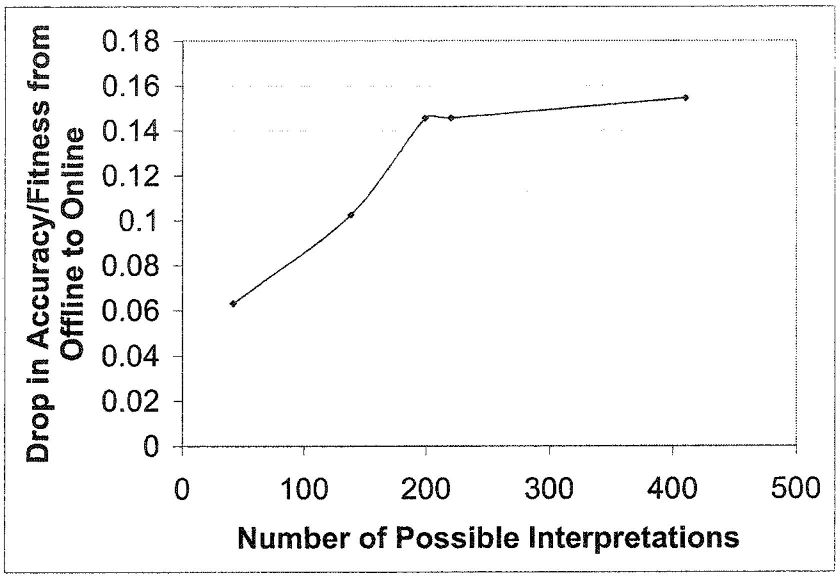


Figure 6.24: Drop in online accuracy/fitness vs average number of possible image interpretations. The points left to right represent Domain-Expert, Filter50,GA/NB, GA/NN, Filter100. Note the change in order, as the GA/NN combination averages more operators per set than GA/NB, but fewer image interpreations (due to the nature of the operators this method chooses).

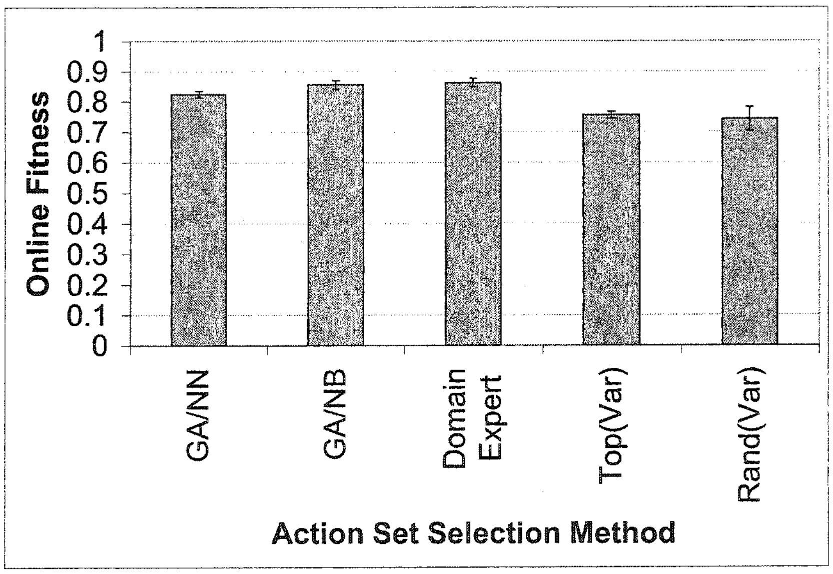


Figure 6.25: Average online fitness of five methods using online training data

6.5 Discussion

From the empirical results several observations can be made:

- HSMM succeeds in discovering action subsets that are smaller in size than the full action set and that when used by the agent allow it to achieve comparable reward $R(A)$ to the reward the agent receives when using the full action set. The cost $C(A)$ incurred by the agent was diminished by as much as 95% in our experiments.

- The Top method is less successful than HSMM on its own because they contain no information as to how many actions to put in a set. Setting the number of actions N to be included in a set as a constant only adds to the human engineering of the system, which is the opposite of one of our goals, which is to eliminate the last vestiges of human engineering. Still, having a ranked list of the operators can be beneficial for slight changes in the final system and action set, and since the method is quite fast, it can be invoked later on in the process without significantly adding to the running time.

- The full operator set used in forestry has numerous operators that are interchangeable without much change in rewards or cost, as can be seen by the fitness of the randomly generated sets. Two different random sets of 150 operators generally have the same approximate fitness. This is because the tabulation of operators is quite close. With the addition of more operators into the full set, this phenomenon should become less likely, since the choice of operators will be increased.

- As expected, MR ADORE's online module does not always return the optimal image labeling available with a given operator set. It errs about 30% of the time and is thus prone to being misled by bad choices. Again, if we keep the number of bad choices small (by supplying the policy with a compact high-quality operator set), we can help keep the online module from being led astray, and increase image interpretation accuracy, as was shown in the experiments.

- In the maze domain, with moves of radius 2, an exhaustive search reveals that the full action set is *not* the optimal set for the explore-exploit maze agent . An action set with just over half the actions of the full set increases fitness by providing fewer choices to the agent as it attempts to discover the goal state in the maze. When we move to radius 5 moves, it seems the full set is the best action set to provide to the explore-exploit agent. This is not proven, since an exhaustive search is not possible. HSMM succeeds in finding action sets with comparable results to the full action set, and better results than other methods action set selection that were tried in our experiments. For radius 2, none of the methods we tried succeeded in finding action sets with good results. This is surprising, since the search space is much smaller than with radius 5, but an investigation into the exhaustive search results reveals that very few of the candidate action sets have performance even close the optimal set, thus it seems our methods simply missed the best sets in their search. With a depth-first search maze agent, and a fitness taking into account deliberation cost, HSMM find an action set that outperforms all other methods, including the full set.

- Using data obtained from the online module of MR ADORE seems like the best approach to training our machine learners to approximate what is a good online operator set. The downfall of this approach is that obtaining one online training datum can take as long as a day with our setup, and generally takes at least a few hours, whereas obtaining one offline training datum can take no longer than a few hours and generally takes only a few minutes.

Chapter 7

Future Work and Conclusions

7.1 Future Work

In the future, we would like to introduce a *Theory of Redundancy*, in order to better understand when our Heuristic Search with Meta-Modelling is applicable. If there is no redundancy in the given action set for a given Markov decision process, than it is less likely that the HSMM method is applicable. One possible similarity measure would be:

$$similarity(a_i, a_j) = 1 - [[F(A/\{a_1\})] - [F(A/\{a_2\})]]$$

where A is the set of n actions $\{a_1, a_2 \ldots, a_n\}$ and $F(A)$ is our fitness equation for an action set given in Definition 3.5. There are many other possible ways to define *similarity*, such as measuring how close the functions of two actions are. In the maze domain for instance, two actions could be considered very similar if they move the agent in almost the same direction, and very different if they move the agent in an opposite direction from each other. In the vision domain, similar vision operators could be operators with the same function, but different parameterizations.

We would also like to use a more intelligent manner to choose the training data for the meta-models within the HSMM method. In [Gilardi and Faraj, 2004] an active learning approach is be taken to select training data for neural networks and is shown to be effective.

Another research avenue worth investigating is using the HSMM method to determine a range for the number of actions to include in an action set. We performed some preliminary tests of this method on MR ADORE.

In our experiments we noted that operator sets with more operators and better *potential* image interpretation accuracy often perform worse when used in online MR ADORE. We hypothesized that the online policy of MR ADORE is led astray by all the *choices* of image interpretations provided by larger operator sets. In contrast to this, the online MR ADORE module was better able to capture the *potential* of smaller operator sets, meaning it more often returned the user an image interpretation

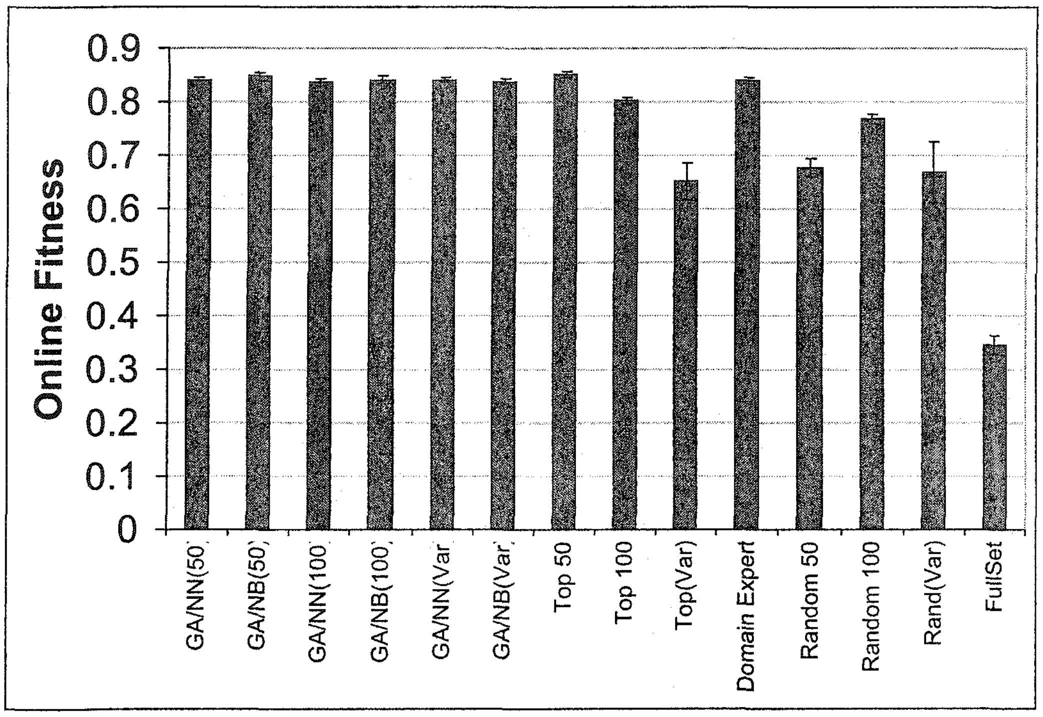


Figure 7.1: Comparing fixed size operator sets to variable size operator sets.

closer to the optimal interpretation provided by a smaller operator set. We thus decided to limit the size of the sets chosen by each method, to decrease the number of choices (and most importantly limit the number of *bad* choices) available to MR ADORE's online module. The values 50 and 100 were chosen because in our earlier experiments (Section 6.4) GAs chose sets with sizes in this range. Results are shown in Figure 7.1.

Limiting the size of operator sets chosen to the range defined by HSMM enables Top method to choose better operator sets for the online module of MR ADORE. GAs performance is not changed significantly, since the bulk of the sets chosen by this method are in the range of 50 to 100 operators anyway. Further investigation is necessary to determine if using the GAs to select an operator set size range then rerunning the experiments is a viable method for choosing operator sets (and action sets in the general MDP case).

We would also like to apply the HSMM method to the feature selection problem to see if it can be useful for more problems besides action set selection in Markov Decision Processes, and to investigate the similarity between feature selection and action set selection.

Currently, in MR ADORE, we work with depth four expansions, meaning at most four operators applications are allowed along any path in the full expansion. Applying more than four operators to an image is possible, and allows for strictly non-decreasing rewards. The problem is that trying all possible combinations of more than four operators becomes intractable with very large operator sets. One idea is to reduce the size of the operator set by performing HSMM on depth four expansions, then executing the depth five expansions with this reduced set, and reducing the set further with another application of HSMM. This process could continue, making it an *iterative* application of HSMM.

7.2 Conclusions

In this thesis we propose a novel method entitled *Heuristic Search with Meta-Modelling* (HSMM) for automatically selecting action sets in Markov decision processes. We combine the strength of wrapper and filter approaches to perform this task that is historically done by human experts, thus reducing the amount of human intervention in otherwise automatic problem solving strategies and reducing the time needed to be spent by humans hand-engineering action and domain specific action sets.

In a broader scope, we investigate the problem of choice, namely, "How much choice is enough?". This problem has been investigated in psychological circles ([Schwartz, 2004]) and many parallels can be drawn between this work and the investigation into how many actions are enough actions in an MDP. This is an action dependent and domain dependent question, thus there is no final answer applicable to all MDPs.

We test our action selection method on two different MDP domains. In the maze domain, we improve the navigation action set for a depth-first search agent by reducing it in size and thereby improving the agent's fitness, by lowering the deliberation cost of the agent. At the same time we keep a comparable reward with regards to solving many different randomly generated mazes.

In the vision domain, we improve the performance of a state of the art adaptive object recognition system (MR ADORE) on novel images by providing it with a compact, domain-specific vision operator set. This set is much less costly in terms of learning and execution cost than is the full operator set, and allows the control policy to match its image interpretation accuracy that was possible with the full operator set.

Bibliography

[Arbib, 1972] M.A. Arbib. *The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory*. Wiley-Interscience, New York, 1972.

[Arbib, 1978] M.A. Arbib. Segmentation, schemas, and cooperative computation, 1978.

[Bins and Draper, 2001] J. Bins and B. Draper. Feature selection from huge feature sets. In *Proceedings of International Conference on Computer Vision*, volume 2, pages 159–165, 2001.

[Blum *et al.*, 1991] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain, 1991.

[Bulitko *et al.*, 2003] V. Bulitko, L. Li, R. Greiner, and I. Levner. Lookahead pathologies for single agent search. In *International Joint Conference on Artificial Intelligence*, 2003.

[Dahm and Ziegler, 2002] Ingo Dahm and Jens Ziegler. Using artificial neural networks to construct a metamodel for the evolution of gait patterns of four-legged walking robots. In *5th International Conference on Climbing and Walking Robots (CLAWAR)*, 2002.

[Dayan and Hinton, 1993] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5, Proceedings of the IEEE Conference in Denver (to appear)*, San Mateo, CA, 1993. Morgan Kaufmann.

[Dougherty and Marryott, 1991] David E. Dougherty and Robert A. Marryott. Optimal groundwater management 1. simulated annealing. In *Water Resources Research*, volume 27, pages 2493–2508, 1991.

[Draper *et al.*, 1996] B. Draper, A. Hanson, and E. Riseman. Knowledge-directed vision: Control, learning and integration. *Proceedings of the IEEE*, 84(11):1625–1637, 1996.

[Draper *et al.*, 2000] B. Draper, J. Bins, and K. Baek. ADORE: adaptive object recognition. *Videre*, 1(4):86–99, 2000.

[Draper, 2003] Bruce A. Draper. From knowledge bases to Markov models to PCA. In *Proceedings of Workshop on Computer Vision System Control Architectures*, Graz, Austria, 2003.

[Emmanouilidis *et al.*, 1999] C. Emmanouilidis, A. Hunter, J. MacIntyre, and C. Cox. Multiple Criteria Genetic Algorithms for Feature Selection in Neurofuzzy Modeling. In *In Proceedings of International Joint Conference on Neural Networks*, Washington, D.C., 1999.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, 1979.

[Gilardi and Faraj, 2004] Nicolas Gilardi and Abdelaziz Faraj. Design of experiments by committee of neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2004.

[Gilmore and Hillston,] S. Gilmore and J. Hillston. Feature interaction in pepa.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Han and Kamber, 2001] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[Harik and Lobo, 1999] Georges R. Harik and Fernando G. Lobo. A parameter-less genetic algorithm. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 258–265, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[Haykin, 1994] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillian College Pub. Co., 1994.

[Holland, 1962] J.H. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3:297–314, 1962.

[Host, Bod Host 2004] Bod Host. http://www.bodhosting.com/html/windowshosting.htm, Bod Host, 2004.

[Jarmulak and Craw, 1999] Jacek Jarmulak and Susan Craw. Genetic algorithms for feature selection and weighting. In *In Proceedings of the IJCAI'99 workshop on Automating the Construction of Case Based Reasoners*, pages 28–33, 1999.

[Jensen, 1996] F.V. Jensen. *An Introduction to Bayesian Networks*. Springer-Verlag, New York, 1996.

[Jin *et al.*, 2001] Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. Managing approximate models in evolutionary aerodynamic design optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 592–599, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[Jin *et al.*, 2003] Y. Jin, M. Huesken, and B. Sendhoff. Quality measures for approximate models in evolutionary computation. In *Proceedings of GECCO Workshops: Workshop on Adaptation, Learning and Approximation in Evolutionary Computation*, pages 170–174, 2003.

[Journel and Huijbregts, 1978] A.G. Journel and Ch. J. Huijbregts. *Mining Geostatics*. Academic Press, London, 1978.

[K.F.Man *et al.*, 1999] K.F.Man, K.S.Tang, and S.Kwong. *Genetic Algorithms: Concepts and Designs*. Springer-Verlag, London-UK, 1999.

[Kira and Rendell, 1992] K. Kira and L. Rendell. The feature selection problem: Traditional methods and a new algorithm. *In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 129–134, 1992.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[Kohavi and John, 1997] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[Kononenko, 1994] Igor Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In *European Conference on Machine Learning*, pages 171–182, 1994.

[Kovarsky, 2001] Alexander Kovarsky. Cmput 650: Final report. Technical report, University of Alberta, 2001.

[Lee *et al.*, 2003] Greg Lee, Vadim Bulitko, Michael Chung, Wesley Mackay, René Malenfant, and Jonathan Newton. Automated operator selection with genetic algorithms and simulated annealing. Technical report, University of Alberta - www.ircl.cs.ualberta.ca, 2003.

[Levner and Bulitko, 2004] I. Levner and V Bulitko. Machine learning for adaptive image interpretation. In *Proceedings of the 16th Innovative Applications of Artificial Intelligence '04 conference*, 2004.

[Levner *et al.*, 2003] I. Levner, V. Bulitko, L. Li, G. Lee, and R Greiner. Towards automated creation of image interpretation systems. In *Proceedings of the Australian Joint Artificial Intelligence Conference*, 2003.

[Madani *et al.*, 2002] O. Madani, V Bulitko, I. Levner, and R. Greiner. Performance of lookahead control policies in the face of abstractions and approximations. In *Proceedings of the Symposium on Abstraction, Reformulation and Approximation*, 2002.

[McCallum, 1993] Andrew McCallum. Overcoming incomplete perception with util distinction memory. In *International Conference on Machine Learning*, pages 190–196, 1993.

[Metropolis *et al.*, 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Simulated annealing. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. WCB/McGraw Hill, Boston, 1997.

[Miyazaki and Kobayashi, 1995] Kazuteru Miyazaki and Shigenobu Kobayashi. Learning deterministic policies in partially observable markov decision processes, 1995.

[N.Srinivas and K.Deb, 1995] N.Srinivas and K.Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. In *Evolutionary Computation*, volume 2(3), pages 221–248, 1995.

[Oliveira *et al.*, 2001] L. S. Oliveira, N. Benahmed, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Feature subset selection using genetic algorithms for handwritten digit recognition. In *In Proceedings of the 14th Brazilian Symposium on Computer Graphics and Image Processing*, pages 362–369, Florianópolis-Brazil, 2001. IEEE Computer Society.

[Oliveira *et al.*, 2002] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Feature selection using multi-objective genetic algorithms for handwritten digit recognition. In *In Proceedings of the 16th IAPR International Conference on Pattern Recognition*, pages 568–571, Quebec City-Canada, 2002. IEEE Computer Society.

[Pudil *et al.*, 1994] P. Pudil, J. Novovicova, and J. Kittler. Floating search methods in feature-selection. *PRL*, 15(11):1119–1125, November 1994.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Kauffman, 1993.

[Rimey and Brown, 1994] R. Rimey and C. Brown. Control of selective perception using bayes nets and decision theory. *International Journal of Computer Vision*, 12:173–207, 1994.

[Rivest, 1987] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[Robert H. Kewley and Embrechts, 2000] Jr. Robert H. Kewley and Mark J. Embrechts. Fuzzy-genetic decision optimization for optimization of complex stochastic systems. In *Proceedings 5th Online World Conference on Soft Computing in Industrial Applications*, 2000.

[Schwartz, 2004] Barry Schwartz. The tyranny of choice. *Scientific American*, pages 70–75, April 2004.

[Soule *et al.*, 1996] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, Stanford University, CA, USA, 28–31 1996. MIT Press.

[Sun *et al.*, 2002] Z. Sun, X. Yuan, G. Bebis, and S. Louis. Neural-network-based gender classification using genetic eigen-feature extraction. In *In Proceedings of IEEE International Joint Conference on Neural Networks*, Honoloulu, Hawaii, 2002.

[Sutton and Barto, 1998] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Talbi and Muntean, 1993] E-G Talbi and T. Muntean. General heuristics for the mapping problem, 1993.

[Teller and Veloso, 1996] Astro Teller and Manuela Veloso. PADO: A new learning architecture for object recognition. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.

[University, 2004] Saint Louis University. http://hem.hj.se/ de96klda/neuralnetworks.htm, 2004.

[Vafaie and Jong, 1992] H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *In Proceeding of the 4th International Conference on Tools with Artificial Intelligence*, pages 200–204, Arlington, VA, 1992.

[Vafaie and Jong, 1993] H. Vafaie and K. De Jong. Robust feature selection algorithms. In *In Proceedings of the Fifth Conference on Tools for Artificial Intelligence*, pages 356–363, Boston, MA, 1993. IEEE Computer Society Press.

[Watkins, 1989] Chris Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, University of Cambridge, UK, 1989.

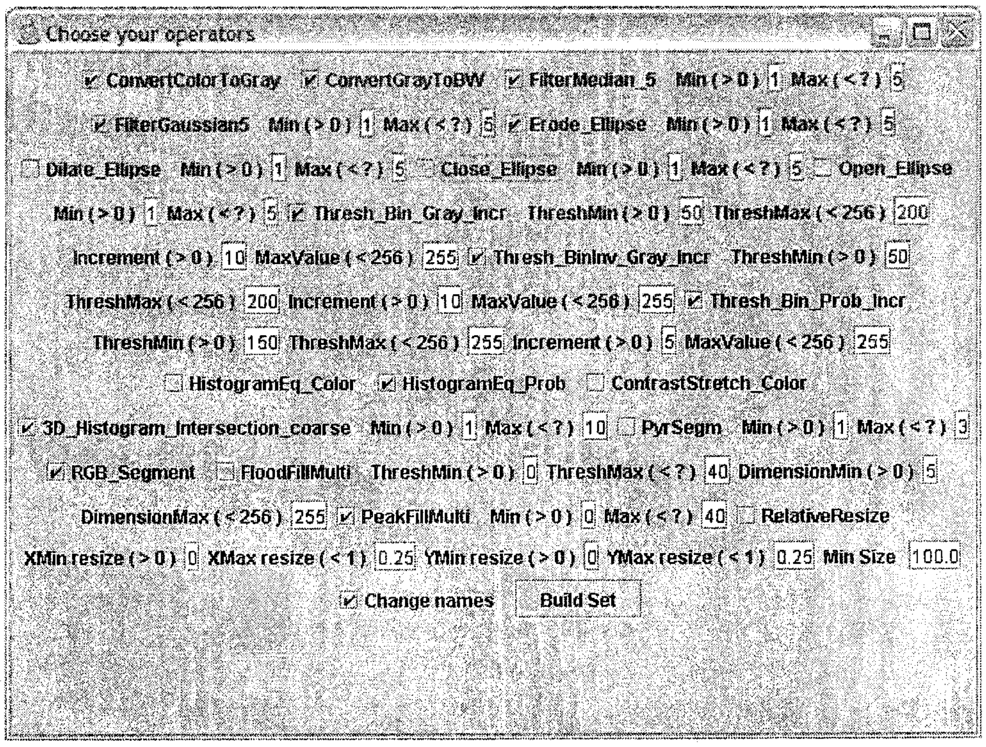
[Willmes *et al.*, 2003] L. Willmes, T. Baeck, Y. Jin, and B. Sendhoff. Comparing neural networks and kriging for fitness approximation in evolutionary computation. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 663–670, 2003.

[Yan and Minsker, 2003] Shengquan Yan and Barbara Minsker. A dynamic meta-model approach to genetic algorithm solution of a risk-based groundwater remediation design model. In *World Water and Environmental Resources Congress 2003 and Related Symposia*, 2003.

Appendix A

Vision Operator Selection Software

In order to facilitate building large vision operators sets, we implemented a java program to easily tabulate different versions of certain operators. Figure A.1 shows a typical screenshot of the GUI. The input to the system is any operator set compatible with MR ADORE [Levner *et al.*, 2003]. The GUI appears with some default (broad) ranges of parameters for those operators that take parameters. The user is asked to specify the minimum and maximum values for parameters, as well as the increment for the parameter in some cases. Operators can be included in the set by clicking the checkbox next to their names (by default all operators are selected to be in the set). The user also has the option to change the names of the operators, which amounts to appending the parameter values to the operator's name. This ensures that we can differentiate between different parameterizations of the same operator. The set is created by clicking the "Build Set" and the output of the system is the set specified by the user, with parameterized operators.



Choose your operators

✓ ConvertColorToGray ✓ ConvertGrayToBW ✓ FilterMedian_5 Min (> 0) 1 Max (< ?) 5

✓ FilterGaussian5 Min (> 0) 1 Max (< ?) 5 ✓ Erode_Ellipse Min (> 0) 1 Max (< ?) 5

☐ Dilate_Ellipse Min (> 0) 1 Max (< ?) 5 ☐ Close_Ellipse Min (> 0) 1 Max (< ?) 5 ☐ Open_Ellipse

Min (> 0) 1 Max (< ?) 5 ✓ Thresh_Bin_Gray_Incr ThreshMin (> 0) 50 ThreshMax (< 256) 200

Increment (> 0) 10 MaxValue (< 256) 255 ✓ Thresh_BinInv_Gray_Incr ThreshMin (> 0) 50

ThreshMax (< 256) 200 Increment (> 0) 10 MaxValue (< 256) 255 ✓ Thresh_Bin_Prob_Incr

ThreshMin (> 0) 150 ThreshMax (< 256) 255 Increment (> 0) 5 MaxValue (< 256) 255

☐ HistogramEq_Color ✓ HistogramEq_Prob ☐ ContrastStretch_Color

✓ 3D_Histogram_Intersection_coarse Min (> 0) 1 Max (< ?) 10 ☐ PyrSegm Min (> 0) 1 Max (< ?) 3

✓ RGB_Segment ☐ FloodFillMulti ThreshMin (> 0) 0 ThreshMax (< ?) 40 DimensionMin (> 0) 5

DimensionMax (< 256) 255 ✓ PeakFillMulti Min (> 0) 0 Max (< ?) 40 ☐ RelativeResize

XMin resize (> 0) 0 XMax resize (< 1) 0.25 YMin resize (> 0) 0 YMax resize (< 1) 0.25 Min Size 100.0

✓ Change names

Figure A.1: A GUI used to select which operators to include in a set, which instantiations of these operators to include, and whether to rename the operators within the set.

Appendix B

Vision Operator Examples

We here give a few visual examples of changes made to images by some of the operators used within MR ADORE.

B.1 Template Matching

A logical manner in which to recognize objects is to try to match a template of the object for which we are looking with the image in which we are looking for them. Figure B.1 shows an image with a spruce tree matched with a spruce tree template. Brighter areas represent areas that are more likely to be a spruce tree, while darker areas represent areas which are most likely not spruce trees.

B.2 Thresholding

Thresholding an image at value *V* changes it to a binary image, with all pixels whose average brightness is equal to or higher than *V* changed to white, and all other pixels changed to black. An example of a thresheld image is shown in Figure B.2.

B.3 Morphological Filtering

Morphological filtering can be done by many methods, a few of which we describe and illustrate in this section.

B.3.1 Gaussian Filtering

Gaussian Filtering works by replacing each pixel by the average pixel with respect to its surrounding neighbours. The operator has as its effect to smooth out an image, as seen in Figure B.3.

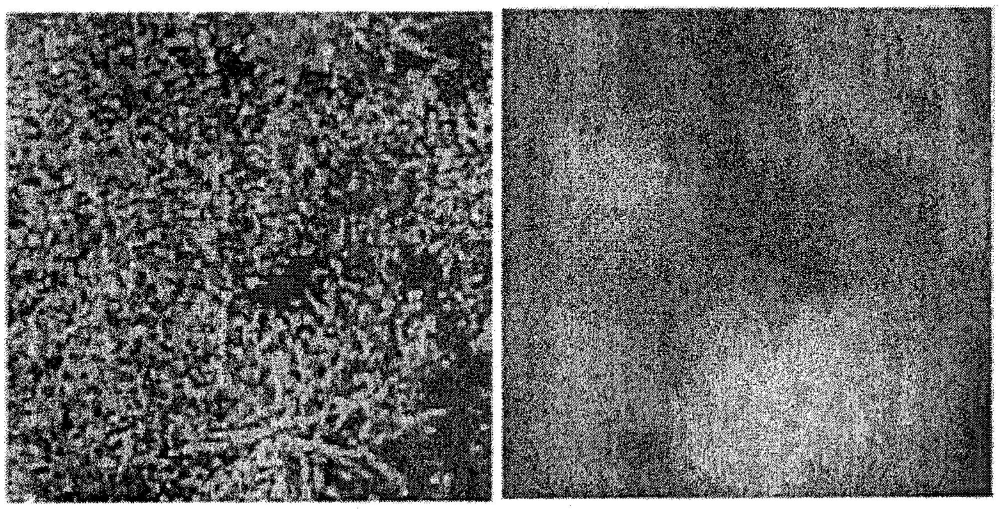


Figure B.1: An image matched with a spruce tree template, producing a likelihood image.

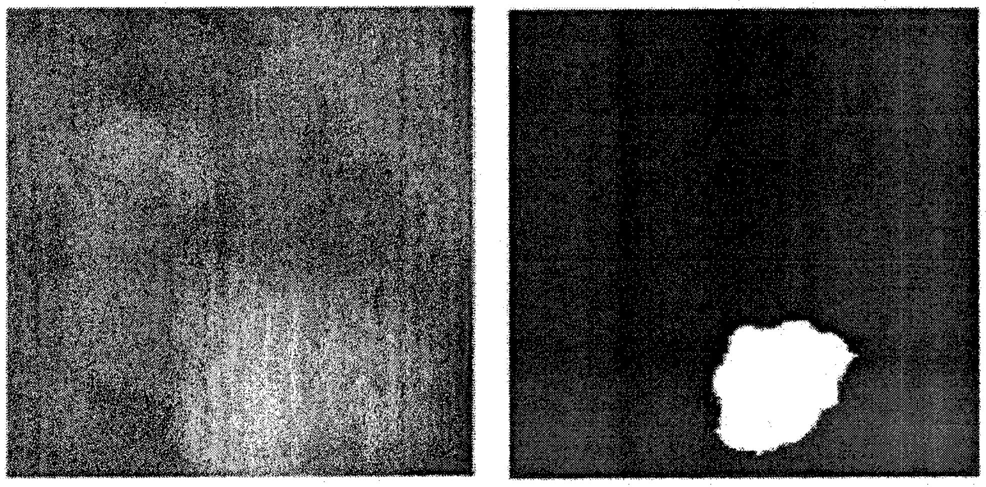


Figure B.2: An image produced by template matching is thresheld to produce a binary image.

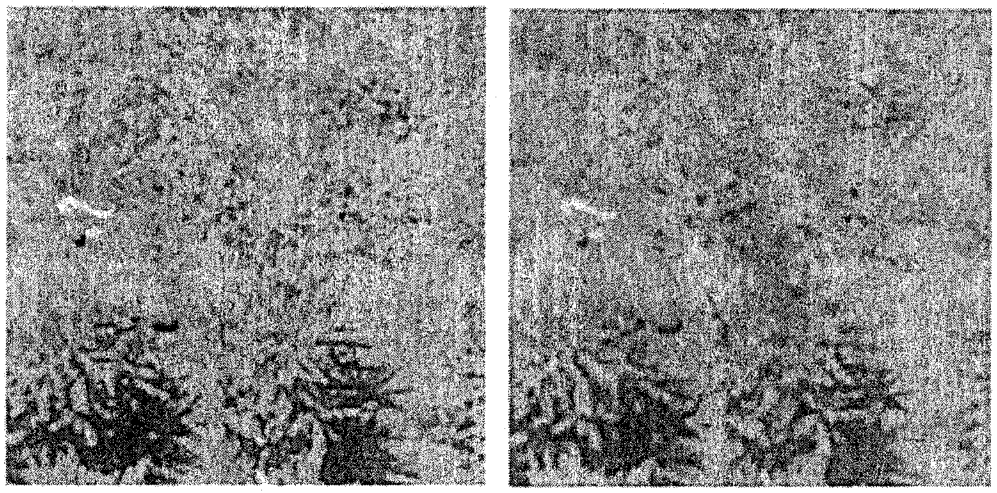


Figure B.3: An image morphed by a gaussian filter.

B.3.2 Minimum Filtering

Another form of filtering, minimum filtering, replaces each pixel with the lowest percentile intensity value within a given neighbourhood. The image is thus darkened as a whole. Figure B.4 shows an example of minimum filtering performed on an image.

B.3.3 Maximum Filtering

Maximum filtering is effectively the opposite of minimum filtering. Each pixel is replaced by the highest percentile intensity value within a given neighbourhood. The image is thus brightened as a whole. Figure B.5 shows an example of maximum filtering performed on an image.

B.4 Histogram Equalization

Histogram equalization is a vision operator that takes three steps. First the histogram of the image is computed. Next the normalized sum of the histogram is calculated, and used to transform the input image by *equalizing* the image intensities, creating broadening peak intensities. Figure B.6 shows an example of colour histogram equalization performed on an image.

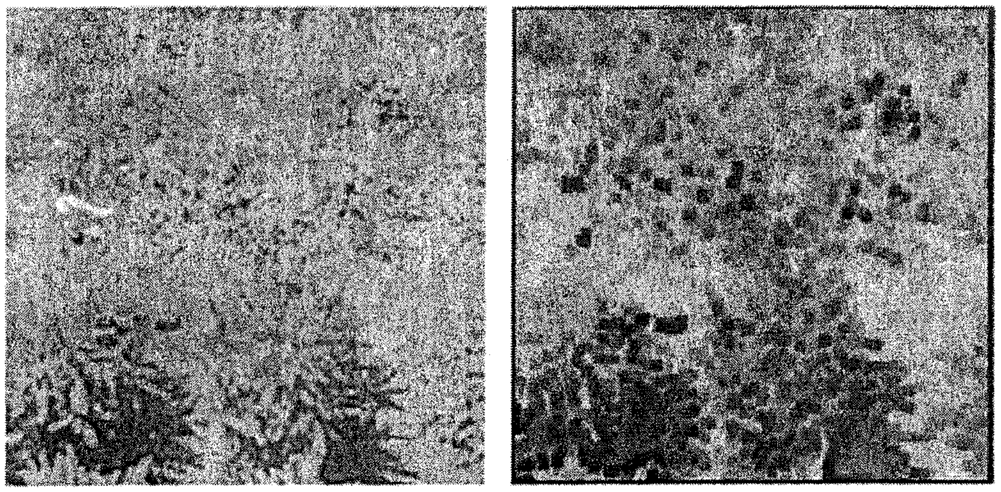


Figure B.4: An image morphed by a minimum filter.

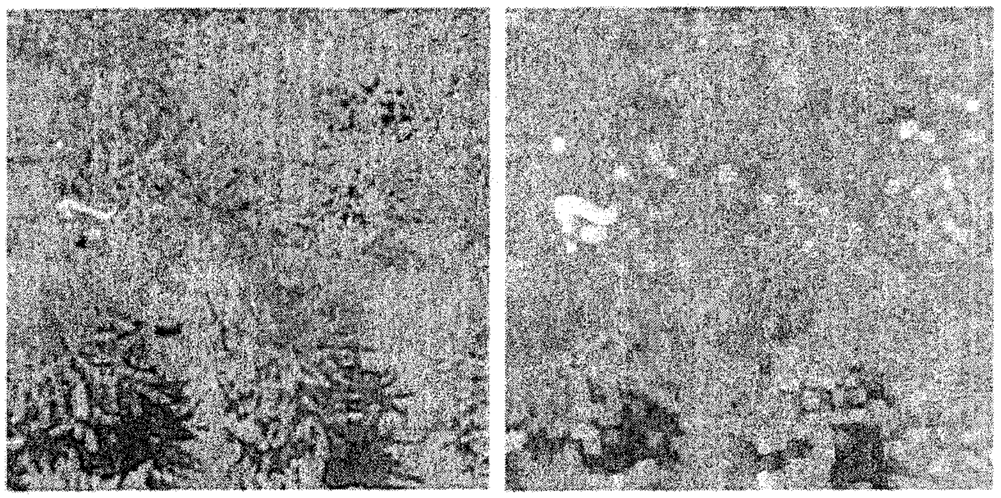


Figure B.5: An image morphed by a maximum filter.

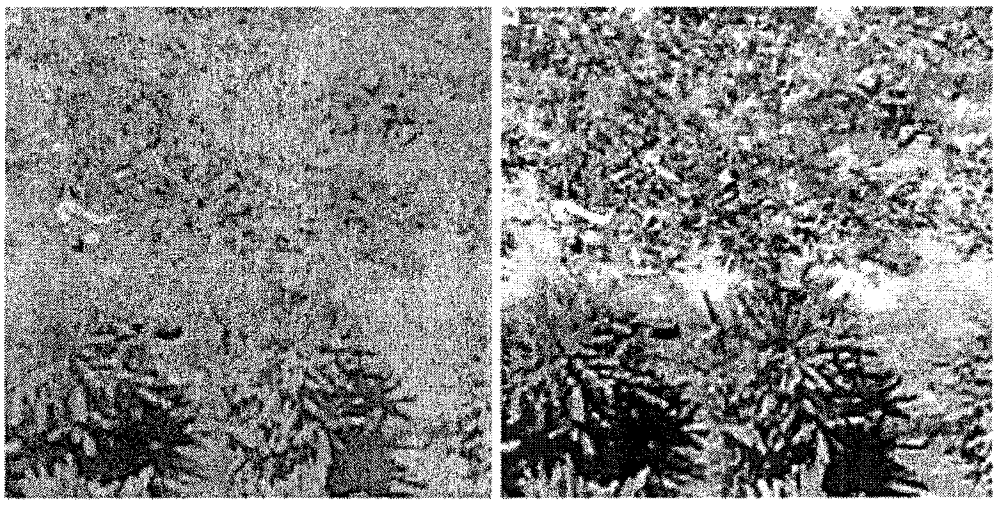


Figure B.6: An image before and after colour histogram equalization has been performed.