National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

VARIATIONS OF A* FOR SEARCHING IN ABSTRACTION HIERARCHIES

By

Maria Beatriz Perez

Thesis Submitted to the School of Graduate Studies in partial fulfilment of the requirements for the degree of

Master

in

Computer Science

under the auspices of the Ottawa-Carleton Institute for Computer Science

Department of Computer Science

Faculty of Science

University of Ottawa

OTTAWA,ONTARIO

ISBN 0-612-07803-5

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Robert C. Holte. Since the undertaking of my thesis, Dr. Holte has been a constant source of support, guidance, and ideas for me. I am most appreciative of his patient assistance.

I would like to thank my family for their constant moral support and encouragement. This would have been a daunting task if they had not been there for me.

Lastly, I would like to thanks all my friends.

Abstract

The aim of this work is to show the usefulness of abstraction in heuristic search. We use the abstract spaces created by applying abstraction techniques to the original problem. These abstract spaces are then used to generate all the heuristic information necessary in order to find an optimal or near optimal solution to the original problem.

One of our objectives is to preserve optimal paths while speeding up search. We have developed new approaches to speed-up A* in an abstraction hierarchy: abstract solutions, minimum edge weight, P-G to generated heuristics from the length of the solution path for those node visited during search, and reuse of heuristic information generated from previous searches. We also consider an approach which postpones the calculation of heuristics obtained by searching in the abstract space. This approach is referred to as Lazy Evaluation.

Another of our objective is to achieve more speed in search sacrificing optimality. For this we introduce a technique called Selective Method.

Our last objective is to identify which abstraction method is the best for the search. We build abstraction hierarchies with different abstraction methods and we evaluate them using different search techniques: Classical Refinement (CR), Optimal Refinement (OR), and Alternating Opportunism (AO), and our variations of A*.

Table of Contents

Table of Tables

Table of Figures

Chapter 1

Introduction

1.1 THE ROLE OF INFORMATION IN SEARCH

Search techniques are the center of many computational processes. We can look at the problem of searching as a fundamental issue of artificial intelligence (AI). The efficiency of the solution of a problem in AI depends on how informed a search technique is. Blind search techniques such as Breadth-First Search and Dijkstra's algorithm [Dijkstra, 1959] do not use information from the problem domain to help its guidance. Although optimal paths can be obtained using this technique, the search can be slow. Since for many big problems this is a disadvantage, an alternative to speed-up blind search was developed. This alternative is called informed search. In an informed search, guidance is provide by heuristics which is a function that estimates the distance from a node to a goal. This information is used to speed-up the search by reducing or ordering the number of alternatives paths during search.

1.2 HEURISTIC GENERATION

A good place to use heuristics is when solving large problems having a huge number of alternatives to determine an optimal path. To show the importance of heuristics in real-life situations, we will present the road map problem (see Fig. 1.1):

Fig.1.1 The Road Map Problem

Suppose we want to find a path from some initial state, say Toronto to some goal state, say Montreal. Assuming that the distance from Toronto to any of its neighbor cities is the same (70 km). At first, if we had to select among all possible roads from Toronto to any neighbor city, we will choose those that lead in the direction of our destination (Montreal). Since Hamilton is in the opposite direction, we will consider this road less promising than Newcastle and Barrie. On the other hand, if we do not know anything about the direction of our destination, and Newcastle and Barrie have the same distance from Toronto any of the two alternatives is equally promising. In order to make a better choice among all the alternatives more information is required, here is when the heuristics plays its role. In our example this information comes from applying the Euclidean Distance in the road map. This will allows us to specify the air distance between cities.

i.e. distance from Newcastle to Montreal (destination). Once we have this additional information, it can be combined with the previous one.

Distance from Toronto to Newcastle (70 Km) + Distance from Newcastle to Montreal (400 Km)

Distance from Toronto to Barrie (70 Km) + Distance from Barrie to Montreal (500 Km)

Some known heuristics in the literature are :

For Towers of Hanoi :

Misplaced Disks [Amarel, 1983]

Alternating Disks [Berlekamp *et al.* , 1982]

Blocking Disks [Amarel,1983]

For The Traveling Salesman Problem (TSP) :

The Minimum Spanning tree (MST) [Held and Karp,1971]

Some of these heuristics do not come exclusively from problem definitions. Some of them use external knowledge in their calculation. Our current system deals with generation of heuristics. The way in which these heuristics are generated is critical when speeding up the search. The generation of accurate and efficient heuristics is not an easy task. It is a time consuming process of discovering. This has motivated research into techniques for creating heuristics automatically. One of these techniques is called abstraction. Using abstraction, a new problem is created from the original problem. The new problem is searched to find a solution which is used to generate heuristic for the original problem. Table 1.1 shows encouraging results of existing techniques based on this idea. All of the Solution lengths using abstraction are between 20% and 34% longer than using no abstraction. However, the amount of work is less than the work done by searching with no abstraction.

The effort involved in the computation of heuristics derived from abstractions can be represented by the amount of search or total work needed in order to generate them. Therefore, our efforts are directed to keep a reasonable amount of search for these computations while preserving the advantage gained by the heuristics. We are also interested in keeping a balance between the total amount of search (including the search needed for the computation of the heuristic) and the quality of the solution path. In order to establish a "good balance" we use as a reference point the results obtained from searching in the original problem without abstraction.

	Search using Abstraction		Search using no Abstraction	
	Solution Length	Total Work	Solution Length	Total Work
Blocks World	11.2	762	9.5	3936
Permutation	8.3	460	6.2	5570
5-puzzle	25	255	21	802
T.O.H	80	767	66	3058

Table 1.1 Search using Abstraction vs. Search using no Abstraction

1.3 SPECIFICATION OF THIS WORK

We consider the problem of finding an optimal or near optimal path in a graph maintaining the amount of work as small as possible. To solve this problem we use abstraction techniques and the heuristic search algorithm A* [Hart, Nilsson and Raphael, 1968].

Applying abstraction recursively, starting from the initial problem until a trivial space or the identity space is created, results in an abstraction hierarchy. Since the idea behind abstraction is to "throw away" details, the new problems or spaces originated are more abstract reducing, in this way, the search in problem solving. Thus, instead of solving the problem directly in the original space we will use a hierarchy of abstractions.

The heuristic search algorithm A* uses an evaluation function F[x] = G[x] + H[x], where the heuristic function H[x] is the estimated distance from a node x to the

goal. An underestimate value of H[x] in A* will find the optimal solution. Dijkstra's algorithm can be considered a special case of A* where H[x]=0 for all **x**, i.e., no heuristic information is given. In order to save time in search while using the A* algorithm we need to generate accurate heuristics. We will compute these heuristics in different ways from the spaces of the hierarchy.

Our overall aims are:

- Solving a problem using a hierarchy of abstractions and A* should be less work than solving a problem without using abstraction.
- The calculation of the heuristic values has to be done so that the amount of work remains small.
- The solution path should be optimal or near optimal. In the case of near optimal, the solution should not be excessively large and the amount of work should be reasonable.

Previous work with these aims has achieved most of these goals but has not completely solved all the problems. Some of our main concerns in this thesis are related to:

- speeding up search without sacrificing optimality.
- additional speed-up gained by sacrificing optimality.
- abstraction techniques that work well.

The techniques we develop will be evaluated experimentally on a set of test state spaces. Two of these test state spaces (Permutation and KL-2000) are from real world applications, the others are standard test state spaces in AI.

The KL-2000 state space is part of a robot motion planning system, such as would be found in manufacturing applications [Kavraki and Latombe, 1993, 1994].

The Permutation state space is a member of the "Carley graph" family, which is being investigated as a promising interconnection topology for multiprocessor networks [Hitz and Mueck, 1994].

1.4 CONTRIBUTIONS

We have developed five approaches to speed-up A* search in an abstraction hierarchy:

1. Reuse of heuristic information from one search to the other.
2. Addition of less expensive sources to compute heuristic information. This sources are combined in the abstract spaces of the hierarchy:

- Minimum edge weight, calculates the minimum weight of the edges emanating from a node.
- Abstract solution, is the solution path obtained from searching in an abstract space of the hierarchy.
- P-G[x] , calculates the difference between the length of the solution path and the distance of a node x from the start node. This source is used for the first time in abstraction.

In the experimental studies we found that the addition of these approaches help to speed-up search without sacrificing optimality.

3. Use of Lazy Evaluation in variations of A* for speed-up. This technique defers the calculation of heuristic information by searching in an abstract space.
4. Introduction of the Selective Method and empirical study of the tradeoff between speed and optimality using this method. To speed-up search, this method reduces the computation of "expensive" heuristics for some nodes by randomly discarding nodes during search.
5. Developing First Time Alternation based on alternating search direction, for A*.

Abstract hierarchies were also created using different abstraction techniques to determine which techniques work well. The maximum hub star abstraction technique proved to be the best.

All the methods have been implemented in the functional language ML.

1.5. Organization of the thesis

Introduction of concepts and algorithms related to state-space search are given in Chapter 2. These concepts will play an important role in our system. Chapter 3 describes the basic ideas of abstraction. It shows how the abstractions are built in our system, and how the search was done by the initial algorithms. It also includes an experimental work for the refinement techniques, limitations of the initial algorithms and possible solutions. Chapter 4 presents the different sources for generating heuristic information, the Selective Method , the Lazy Evaluation and the main algorithms. Chapter 5 describes experiments on several standard AI state spaces. These state spaces are the Towers of Hanoi, 5-puzzle, the Permutation state space, the Blocks World and KL-2000. The experiments are needed to compare the benefits gained from our variations of A*. Finally the conclusion, in Chapter 7, describes the limitations and weaknesses of our approach as well as the future work.

Chapter 2

Definitions

In this chapter we introduce concepts and an algorithm related to state-space search. Particular attention is given to those concepts and algorithm that will play an important role in our system, namely:

Direction of Search

Evaluation Functions (F, H, G)

The A* Algorithm

2.1 PROBLEM SOLVING = STATE-SPACE AND SEARCH

A problem can be represented in many ways. One of them is the state-space approach which consists of representing the problem in terms of states and operators. A state is the representation of a specific situation of the problem. An operator is a partial function that transforms one state into another state.

The state-space can be defined by the following tuple:

$\langle S, O \subset S \times S \rangle$

where S is the set of possible states of the problem, and O is the set of operators, or transitions from state to state.

An example of the state-space of the Blocks World puzzle with two blocks is given in Fig. 2.1. There is a "robot hand" which picks up or puts down a block. There are four basic operations (operators) in this puzzle. Those are: put a block on top of a stack of blocks, put a block onto the table, pick up a block from the table, and pick up a block on top of a stack of blocks. The edges connecting the circles represent the application of the operators.

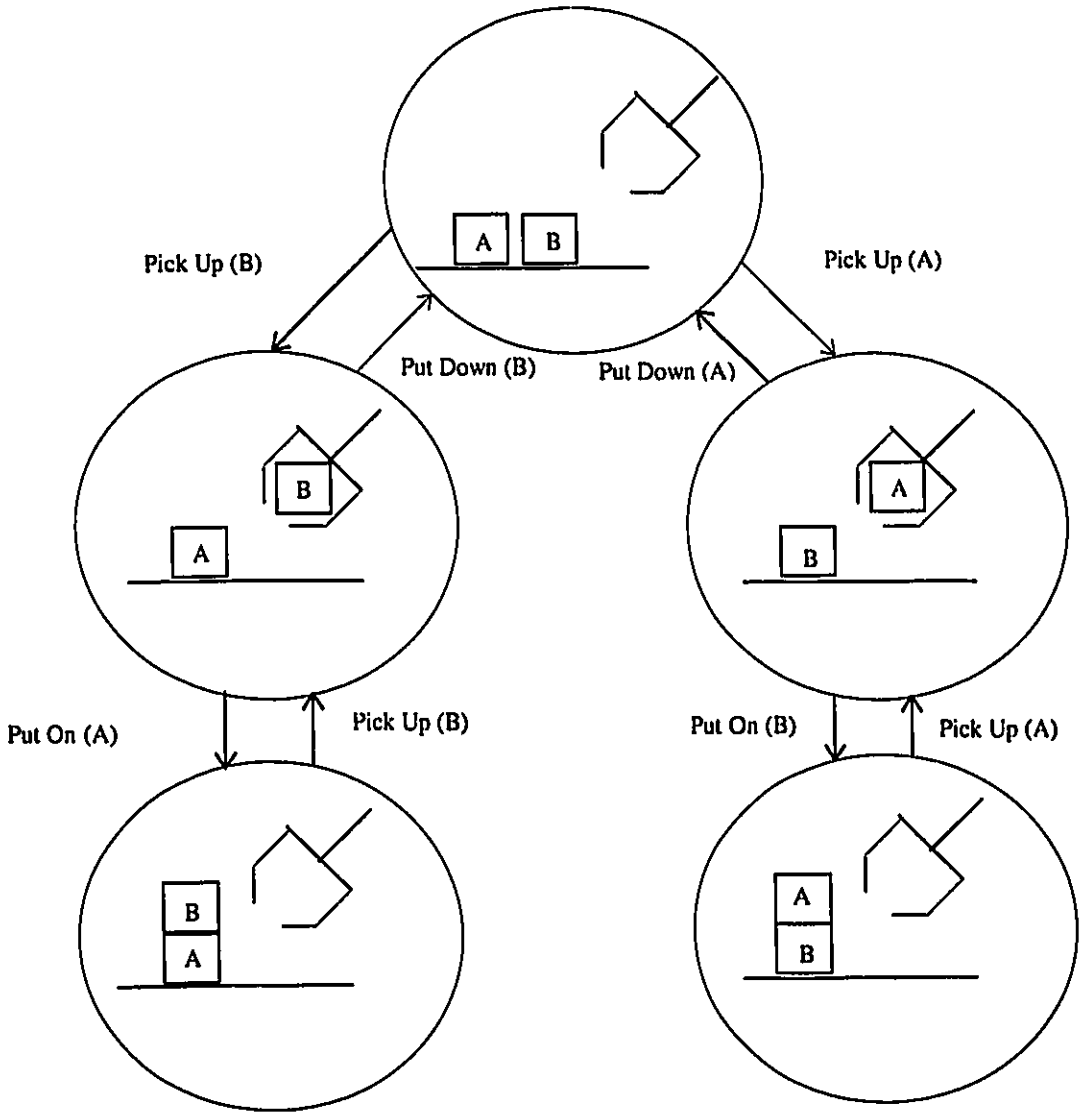


Fig. 2.1 State Space of the Blocks World with two Blocks

Each operator has an associated cost. This cost is given in the form of a weight function w, that generates positive values ($\Re^+$)

w = weight (O,S)

Operators are not necessarily symmetric. The operator applied to one state X generates another state Y that might not have a complementary operator that applied to state Y generates state X (directed graphs). When all the operators have their complementary operator and the weights are the same for all of them the state-space can be represented as an undirected graph.

Given the state space, a problem instance is a pair (s,g) from S. s is the start state corresponding to where we are at the beginning of the search. g is the goal state corresponding to where we want to be.

For each of these instances, the problem solver should find a sequence of actions (operators) that lead from the start state to the goal state.

Solving a problem in a state-space is equivalent to finding a path in the corresponding graph. The nodes in the graph represent the states and the edges represent the operators or transitions from one state to another.

The state-space graph can be represented in an explicit or implicit way. In an explicit representation the nodes and edges are given explicitly in a table, for example. This table might list every node in the graph, its successors and the costs of the associated edges. On the other hand, in an implicit representation nodes are generated gradually by applying a successor operator to a node to give all the successors of that node and the cost of the edges. Then the successor operator is applied to those successors, and so on. This process of searching for a solution makes explicit part of an implicit graph.

In our system some of our techniques (e.g. the star abstraction, see page 24) only work for explicit graphs. Transferring them to implicit graphs is non-trivial.

2.2 SEARCH DIRECTION

The objective in graph search is to find a path from one node to another. Let us call the starting node N_1 and the goal node N_2. The search can proceed in two distinct directions. Searching from N_1 to N_2 is called forward search; searching from N_2 to N_1 is called backward search.

Depending on the topology of the problem space we could choose one of the two directions for searching. A good choice could reduce significantly the number of paths that will be explored.

One of the aspects that we should analyze in the problem space when choosing the search direction is the branching factor or number of children of each node in the graph. The smaller the branching factor, the faster the search.

These issues are irrelevant in undirected graphs. Our motivation in this area is different and it will be clarified in Chapter 3.

2.2.1 Forward Search

The construction of the search tree starts with the start state (the root of the tree). To generate the next level of the tree all the operators applicable to the root node are applied to it creating new nodes (children of the root node). To continue generating the children of each previously generated node apply their corresponding operators. This process continues until one of the nodes generated matches the goal node. This is illustrated by 2.2.

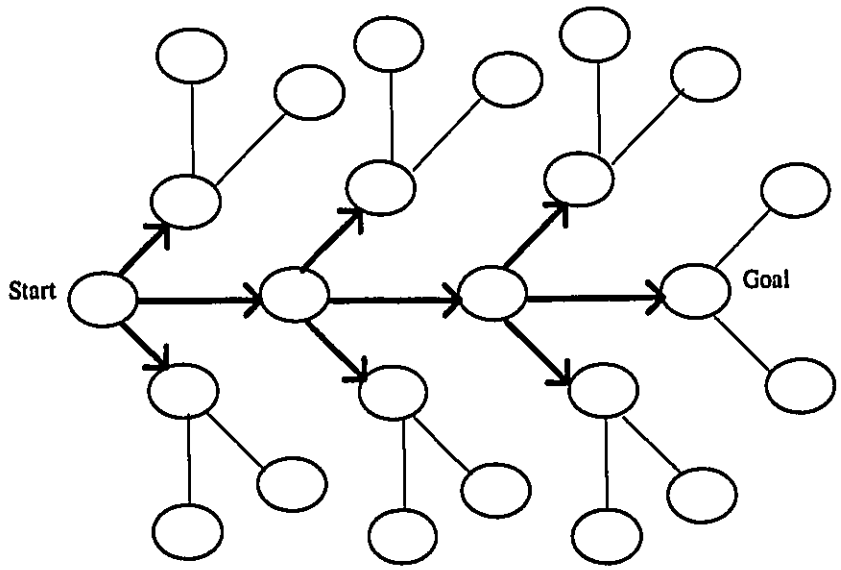


Fig. 2.2 Forward Search

2.2.2 Backward Search

In Backward Search, the root of the search tree is the goal node. The next level of the tree is generated by finding all the predecessor nodes of the goal node (all those states (nodes) that have operators which transform them into the goal state (goal node)). If the graph is undirected, the next level of the tree can be generated by applying the operator to the root node. Then to continue generating the next level of the search tree each node generated in the previous level is examined in the same way as we did with the root node (goal node). This process is done until a node that matches the start node is generated.

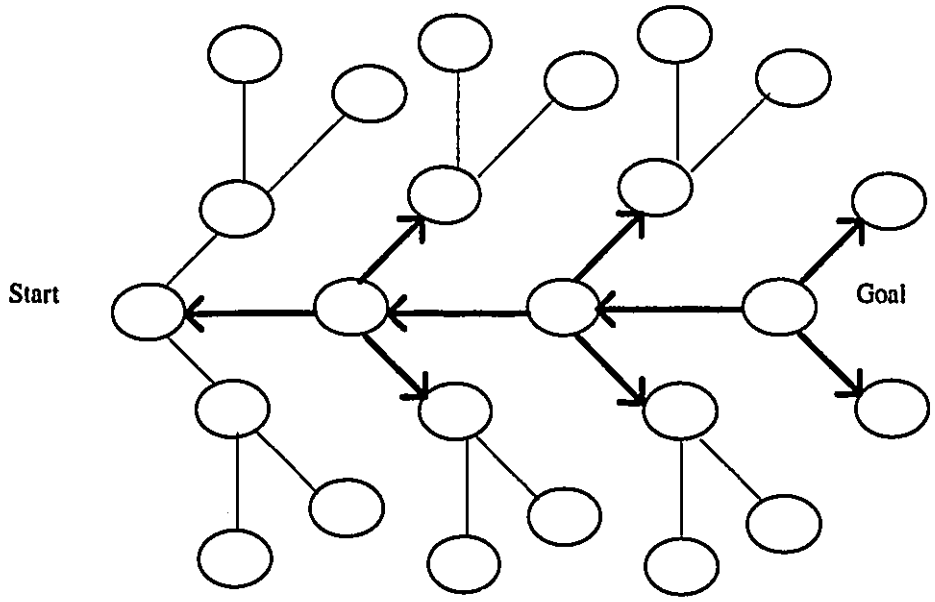


Fig. 2.3 Backward Search

2.2.3 Bidirectional Search

Another way of searching is to use both directions of search simultaneously until the two paths generated from each root meet. This technique is known as Bidirectional Search [Pohl, 1971; Dillenburg and Nelson, 1994]. The problem with this approach is that it can happen that the two paths miss each other. This may occur when several path

solutions exist and the two searches find different paths. As a consequence more nodes could be expanded than in unidirectional search.

2.3 SEARCH CONTROL STRATEGY

To solve a problem in a graph in an efficient manner it is important to make a good choice of the search strategy.

The search strategy guides the search in the construction of the search tree from a graph. The solution to the problem can be obtained by tracing one of the paths of the tree from the root (start state) to one of its leaves (goal state). The other paths in the tree represent additional search. This additional search can be reduce by eliminating some of those paths. The reduction of additional work depends on the search strategy used.

During search the nodes of a graph change from one category to another. The categories are:

expanded nodes = CLOSED nodes (nodes whose children have all been generated)

explored nodes = OPEN nodes (nodes visited but not expanded yet)

unexplored nodes = nodes yet to be visited

visited nodes = OPEN nodes + CLOSED nodes

The idea behind search is to expand open nodes starting from an initial node until a goal node is reached. Based on this idea each search strategy orders the nodes in the OPEN list. The aim is to minimize the number of expansions and the number of alternative paths in the search tree. It is clear that search strategies have a big impact on how quickly a problem is solved and whether it is solved.

2.3.1 Uninformed Graph Search

In an Uninformed Graph Search no knowledge of the problem is used during search. The nodes in the OPEN list are ordered according to their depth in the search tree. Therefore, the search is guided by the depth of a node in the search tree. All alternative paths generated during search are kept in reserve for further consideration while search

proceeds until a goal node is reached. This kind of search is not typically used in AI but we describe it here only for the purpose of comparison.

In this category we can find Depth-first search, Dijkstra's algorithm [Dijkstra, 1959], Breadth-First search, etc.

2.3.2 Informed Graph Search

At any point in time there are many open nodes to be expanded. A good search strategy will make a good decision about which node should be expanded next. Such strategies compute the "goodness" of a node based on knowledge of the problem to be solved. This computation can be represented in terms of evaluation functions. An evaluation function gathers specific information related to the node to which it is applied. The information varies according to the definition of the function. Usually it represents the cost to go from the start node to the node that we are currently evaluating in the function (we will call this evaluation function G[node]), or the cost to go from the node currently being evaluated in the function to the goal node (we will call this evaluation function H[node]), or the sum of G[node] and H[node] (we will call this evaluation function F[node]).

2.4 THE CLASSIC A* ALGORITHM

The most studied version of the informed best-first strategy is the A* Algorithm [Hart, Nilsson and Raphael, 1968]. A* has become the standard heuristic search algorithm in AI.

The general form of the A* algorithm which we will call classic A*, is given in Fig. 2.4.

This algorithm uses **OPEN** and **CLOSED** lists as well as the evaluation functions **G, F,** and **h** defined below.

OPEN is a priority queue in which the elements with highest priority are the open nodes with the smallest value of the evaluation function **F**.

CLOSED is a list containing nodes that have already been expanded.

The evaluation functions are defined as follows: the **G[n]** function represents the distance from the start node to the node **n** along the shortest path found so far. **h(n)** is a heuristic function that estimates the distance from **n** to some goal node, the resulting value of this function is stored in **H[n]**. The distinction between H[n] and h[n] will be important in subsequent chapters. The evaluation function **F[n]** is the sum of **G[n]** and **H[n]**.

PRED[n] is a pointer indicating the current parent of the node **n** in the search tree.

Input : S = Start node G = Goal node

Output : A path from S to G

1 OPEN← {S}; CLOSED← Ø; G[S] ← 0; PRED[S]←NULL

LOOP :

2. if OPEN is empty then found←false, go to 9

3. Select a node X on OPEN for which F[X] is the least (ties are broken arbitrarily, but always in favor of goal nodes)

4. If X is a goal then found←true, go to 9

5. Remove X from OPEN and put it on CLOSED.

6. Generate the set of successors of X

7. for each n in successors do

8. if n is not already on OPEN or on CLOSED

then

(a) add n to OPEN

(b) G[n]←G[X] + distance(X,n)

(c) H[n]←h(n)

(d) F[n] ←G[n] + H[n]

(e) PRED[n]←X

else if n is already on OPEN or CLOSED and G[X] + distance(X,n) < G[n]

then

(f) G[n]←G[X] + distance(X,n)

(g) F[n] ←G[n] + H[n]

(h) PRED[n]←X

(i) if n is on CLOSED remove it from there and insert it on OPEN.

ENDLOOP

9. If found is false then exit with "failure"

else obtain the solution by tracing a path along the pointers from X back to S.

Fig. 2.4 Classic A* Algorithm

2.5 SOME THEOREMS CONCERNING A*

In the previous section we presented the classic A* algorithm. Here we will introduce some theorems about A*. These theorems show the importance of the h function in the A* algorithm. We will refer to these theorems in the next chapters.

Standard Definitions

A heuristic function h(n) is said to be **admissible** if $\forall n[h(n) \leq distance(n,G)]$.

A heuristic function h(n) is said to be **monotone** if for all nodes n_i and n_j, such that n_j is a successor of n_i,

$h(n_i) - h(n_j) \leq distance(n_i,n_j)$

with

$h(G) = 0$

Theorem 2.5.1 [Pearl, 1985. p. 77]:

If there is a path from a start node to a goal node, and h is admissible, A* terminates by finding an optimal path.

Discussion:

When h is admissible, the value of F(n) for every node n on the shortest path (P) from the start to goal is less than or equal to the length of P.

Theorem 2.5.2 [Pearl, 1985. p. 83]:

If the monotone restriction is satisfied, A* finds an optimal path to any node it selects for expansion.

Discussion:

Suppose A* has found a nonoptimal path to a node n, of length $G(n) > G^*(n)$. For every node x on the optimal path to n, $F(x) < G(n) + h(n)$ and therefore will be expanded before n is expanded. This implies that the shortest path to n will be found before n is expanded.

Theorem 2.5.3 [Pearl, 1985. p. 83]:

Monotonicity implies admissibility.

Discussion:

For any two nodes n_i and n_j :

$$h(n_i) \leq \text{distance}(n_i,n_j) + h(n_j)$$

Assuming that the distance between n_i and n_j is defined by the cost of the shortest path between the two nodes, then:

$$h(n_i) \leq \text{Cost of the shortest path from } n_i \text{ to } n_j + h(n_j)$$

As a special case, consider n_j to be the goal node, from this it follows that:

$$h(n_i) \leq \text{Cost of the shortest path from } n_i \text{ to the goal (G)} + h(G)$$

since $h(G) = 0$, then

$$h(n_i) \leq \text{Cost of the shortest path from } n_i \text{ to G}$$

Theorem 2.5.4 [Pearl, 1985. p. 82]:

If the monotone restriction holds, there is no need for A* to test if a newly generated node is already on CLOSED, and there is no need to change the PRED pointer in the search tree of any successors of this node in the search graph.

Discussion:

Using a monotonic heuristic function, A* has already found the shortest path from the start node to **n** for every node expanded. Thus, we never need to check if a newly generated node is in the CLOSED list since such a copy cannot be in CLOSED.

Theorem 2.5.5 [Pearl, 1985. p. 81]:

A heuristic function h1(n) is said to be more informed that another heuristic function h2(n) if $\forall n[h2(n) \leq h1(n)]$ and $\exists n[h2(n) < h1(n)]$ and both are admissible. At termination of the search every node expanded by h1(n) is also expanded by h2(n) i.e. h2(n) expands at least as many nodes as h1(n) does.

Discussion:

To enable A* to construct the smallest possible search graph, we should have h as the highest possible lower bound on h*.

Theorem 2.5.6 [Pohl, 1969. p. 61]:

If h is perfect (always returns the exact distance to the goal), then the search will visit the fewest nodes possible given an optimal path.

Discussion:

The F values of different nodes will be the same as we traverse the solution path: as we move from one of these nodes to the next, the h values will decrease by the same amount that the G value increases. For example, if there is a unique shortest path we shall expand only those nodes that lie on it.

2.6 VARIATIONS ON A*

Perimeter Search and HPA are path finding algorithms that are similar to the A* algorithm. These variations are closely related to some of the variations introduced in Chapter 5 and 6.

2.6.1 Perimeter Search

The perimeter search algorithm [Dillenburg and Nelson, 1994] can be categorized as a bidirectional search algorithm. Unlike normal bidirectional search it does not search in both directions simultaneously. This algorithm splits each search into two separate stages. First it performs a depth-limited breadth first search from the goal node to generate the "perimeter" and then proceeds as a unidirectional search from the start node. Once a node in the second search matches a node in the perimeter the search terminates. Then, the solution path is obtained from the start node to the perimeter node plus the stored path from the perimeter node to the goal. The advantage of a depth-limited perimeter are: first, the perimeter is computed only when it is necessary (once for each possible goal), second, continuos retargeting during search is avoided when the opposite wavefront changes, and third the search efficiency is improved.

The perimeter can also be generated using A*. The nodes on the perimeter generated by using A* have approximately the same F value (constant evaluation perimeter). These perimeters are more expensive to generate, and the number of expansions are more difficult to analyze. The perimeters generated using Breadth First search (constant depth perimeter) overcome these disadvantages.

Since the two searches are independent the second search can be done using any type of heuristic search algorithm. Perimeter Search is limited to using A* and IDA* [Slate and Atkin, 1977, Korf, 1985].

2.6.2 Heuristic Path Algorithm (HPA)

HPA [Pohl, 1969,1970] differs from A* algorithm in the use of a weighted evaluation function :

$$F[X] = (1-w)\, G[x] + w\, H[x], \quad 0 \leq w \leq 1$$

This function allows one to examine the effects of G and H separately. The weight w=0 reflects the effects of G in the search and the weight w = 1 makes HPA base its search on the estimate of the remaining of the path to the goal (function H), as is done by the Graph Traverser system [Doran and Michie, 1966]. The weight w = 0.5 represents the original function for A*. If $w \leq 0.5$, HPA produces an optimal path solution if h is admissible.

For values $w > 0.5$ the heuristic function does not guarantee a minimum length solution, thereby sacrificing the optimal path solution. The solution can be in some cases far from the optimal one.

In order to maintain admissibility or to have a solution within a factor of (1 + e) of the optimal solution, Pohl developed a technique called "dynamic weighting" which makes w a function. We have not investigated this variation in A*.

Chapter 3

Abstraction

The objective of abstraction is to reduce the complexity of large problems by simplifying them. This process has been used in many studies areas in AI, including:

- problem solving [Minton *et al.*, 1989],
- scheduling [Prieditis and Janakiraman, 1993],
- planning [Newell and Simon, 1972],
- theorem proving [Guinchiglia and Walsh, 1992].

The basic idea of abstraction is to ignore details in the original representation and to concentrate on the "important" features of the problem. For a more formal description see [Giunchiglia and Walsh, 1992].

There are many definitions of abstraction including adding edges, dropping operators, and applying homomorphic transformations. Our study concerns homomorphic transformations.

This Chapter reviews previous work on refinement techniques that was our starting point. To evaluate this techniques we ran experiments which are presented in this Chapter with a discussion of results obtained.

3.1 HOMOMORPHIC TRANSFORMATIONS

In a Homomorphic Transformation a state space is abstracted by mapping many states onto a single state. This abstract space is then used to solve the original problem. This is done by using the abstract solution as an outline to guide the search in the original space.

The relation among states in the original problem should be preserved in the abstract space. The mapping of states plays an important role in this task. A bad mapping of states can misrepresent the original problem and therefore mislead the search in the

original space. This is illustrated in Fig. 3.1. To go from N to X in the original graph we must first go to either L or Z or V. In the "bad abstraction" case the abstract space indicates that it is not necessary to do any search at all, because N and X are in the same class, suggesting that there is a path inside this class connecting N to X. In the "good abstraction" case, the abstract space shows that to go from N to X there is a path from N to X made up of two paths: one in the class containing N, the other in the class containing X.

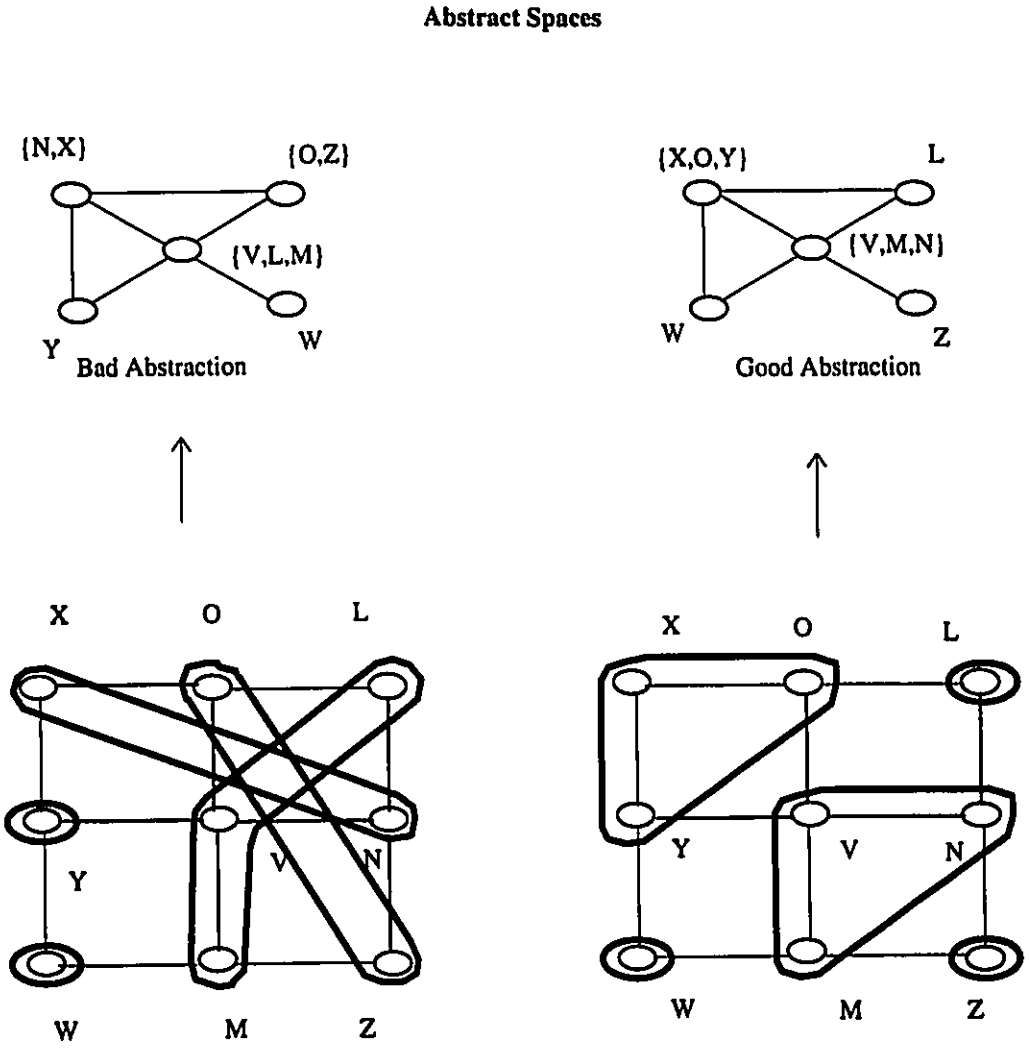


Fig. 3.1 Mapping of States

In this thesis states mapped to a class form strongly connected components of the state space graph (called compact classes in [Mkadmi, 1993]). This guarantees that any two states mapped into a single class are connected by a path within the class.

Compact classes can be created in many different ways. They could have a star shape, or a tree shape, or just be pairs of states. See Fig. 3.2.

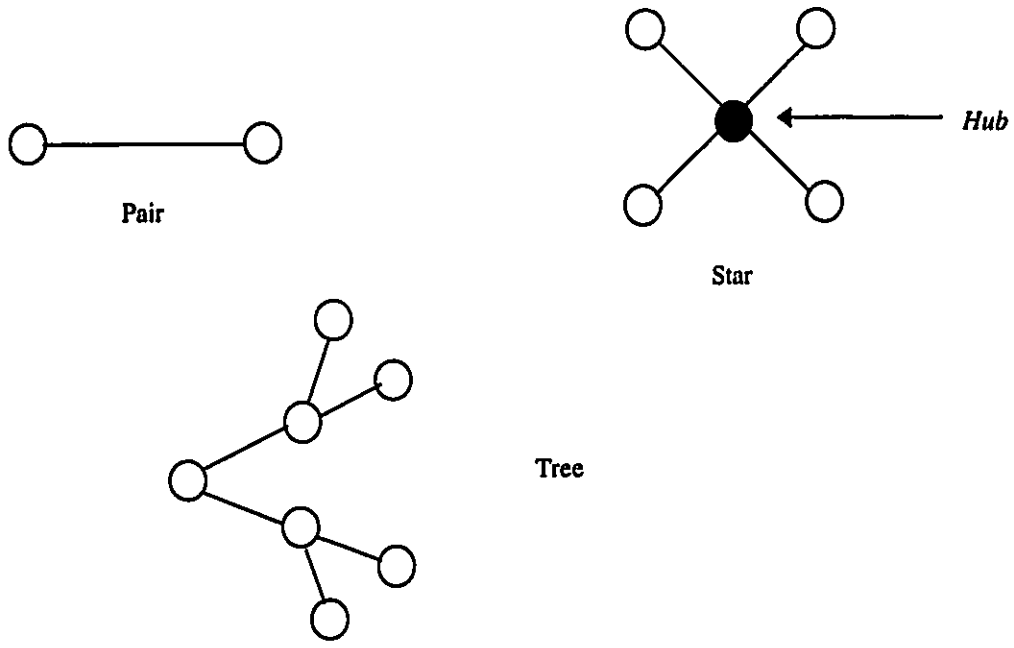


Fig. 3.2 Compact Class Patterns

In this thesis we will only build classes using the star method. In order to build these classes we must select a hub node for each class. There are two main criteria to select a hub. The maximum hub criterion selects as the hub the node with maximum number of neighbors not yet assigned to a class. The random hub criterion randomly selects the hub node. Another factor to consider when building classes with the star method is the maximum distance between any two nodes inside a class. This distance is called diameter. See Fig. 3.3.

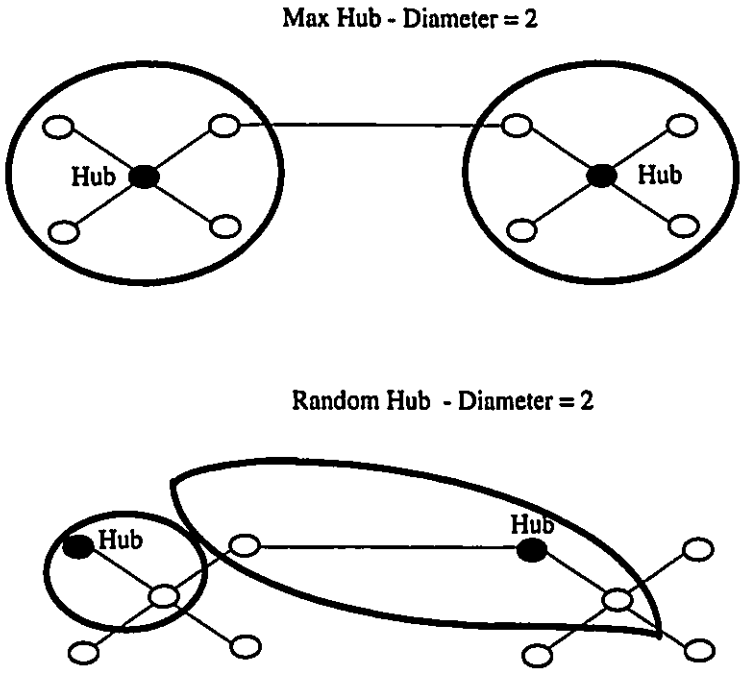


Fig. 3.3 Max Hubs vs. Random Hubs

To create an abstract space all the nodes of a graph are mapped into classes using one or more compact class patterns. A new state space is created using all these classes. The process of mapping nodes of a graph to create a state space can be repeated from the new state space. States spaces are created in this way until a state space with a single node is originated (all the nodes are mapped to the same class). All these states spaces represent an abstraction hierarchy.

Fig. 3.4 shows two abstraction hierarchies. In the abstraction hierarchy on the right side of the figure the compact classes are built using the star method with random hubs. The hub #1 is found first, therefore the first class is formed including all of its neighbors. Hub #2 is found next, but some of its neighbors are already in a class. Those neighbors not included in any class are grouped to form a new class. Each of the three nodes left form a class containing only one node. These nodes are called Orphans. Orphan nodes can be eliminated by adding them to a neighbor class. After all the classes are built, the process is repeated to the new state space (Second level of abstraction). This time all the nodes are grouped into a single class, thus creating, the top level of the abstraction

hierarchy. The left side shows an abstraction hierarchy in which the compact classes are built using the pair method.

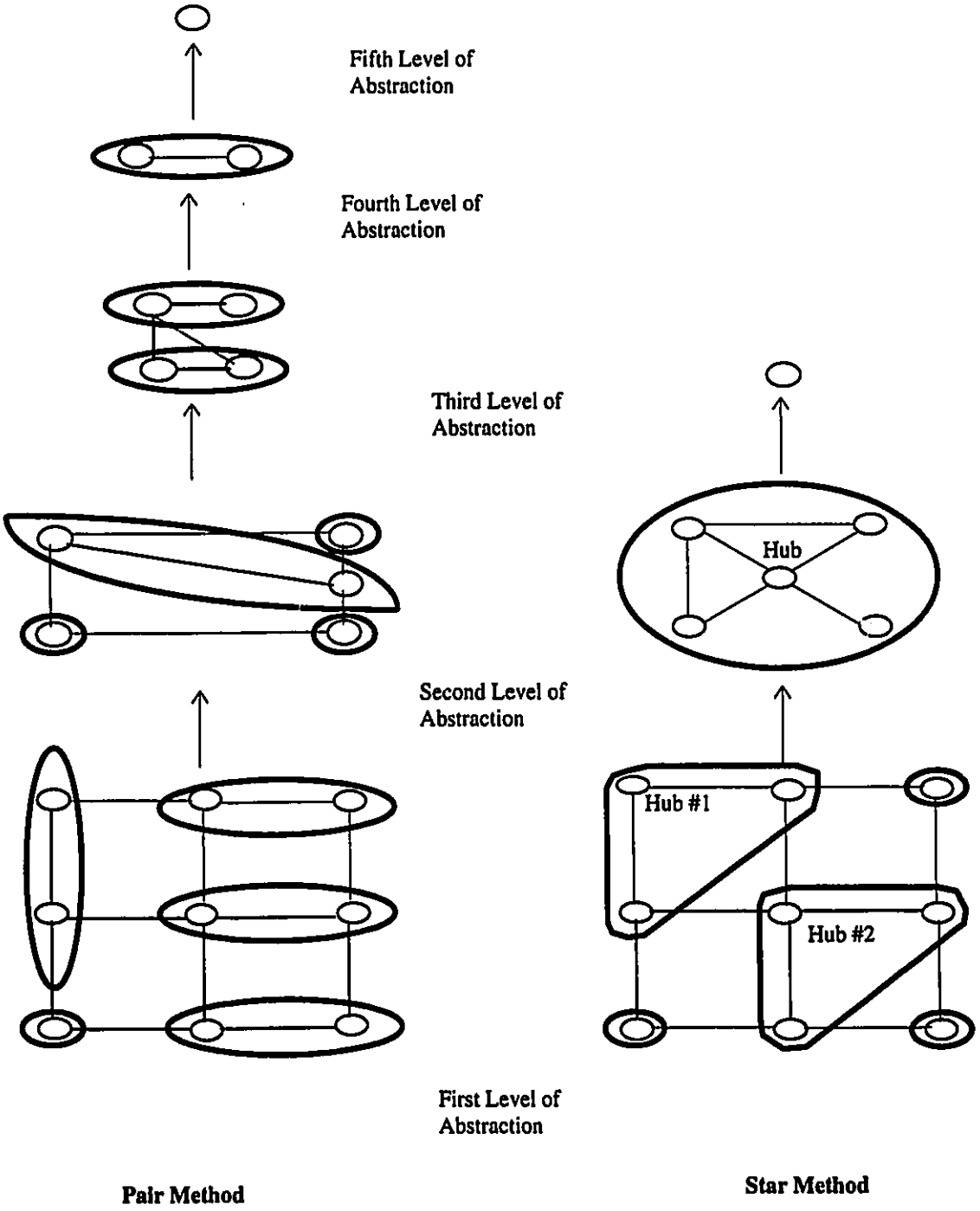


Fig. 3.4 Abstract Space Hierarchies

Studies [Mkadmi, 1993] show that the efficiency of searching in an abstraction hierarchy is heavily influenced by the patterns used to construct the abstract classes. Minimizing the diameter and maximizing the number of states inside a class help to maximize efficiency. This is the motivation of the star method with the maximum hub criterion.

In the next section we will describe some of the first algorithms used to search in abstraction hierarchy created from those previously mentioned patterns.

3.2 ABSTRACT SOLUTION AS A GUIDE TO SOLVE PROBLEMS

The standard way to search using abstraction hierarchy is "Classical Refinement" [Mkadmi, 1993]. It finds an abstract path from a start node to a goal node and uses each node in the path as a subgoal when searching in the original space. In other words, it restricts the search to those states that are on the abstract path just found. Because the solution lengths were to long new techniques for using abstract solutions were developed [Holte *et al.* , 1994]:

Optimal Refinement

Path Opportunism

Alternating Opportunism

3.2.1 Classical Refinement

Let the sequence of states S_1 S_2 S_3 S_4 be the abstract solution found in the abstract space where S_1 is the abstract start state, S_4 is the goal state and all the states in between are in the solution path (see Fig. 3.5). To refine this abstract solution we should look for a path connecting the original start state in class S_1 to any state in class S_2. This process is repeated from the state reached in class S_2 to any state in class S_3 and so on until we reach a state in class S_4. If the state reached is the goal node then refinement is complete, otherwise the search continues in S_4 until the goal node is reached. Classical Refinement always proceeds through the abstract solution in a strict sequence. Each class in the

sequence is considered a subgoal, when a state of the next class in the sequence is reached the search in the current class stops, and a new search begins in the next class. Although Classical Refinement would see the edge $E_{\{S1,S3\}}$ connecting a node from S_1 to S_3 and the edge $E_{\{S2,S4\}}$ connecting a node from S_2 to S_4, it ignores them. In the case of $E_{\{S1,S3\}}$ class S_2 is skipped and in the case of $E_{\{S2,S4\}}$ class S_3 is skipped. In both cases the sequence S_1 S_2 S_3 S_4 has been broken.

No search is done when the start state is equal to the goal state or when the whole space becomes one single state.

Fig. 3.5 shows a path found using Classical Refinement. This path is indicated in bold.

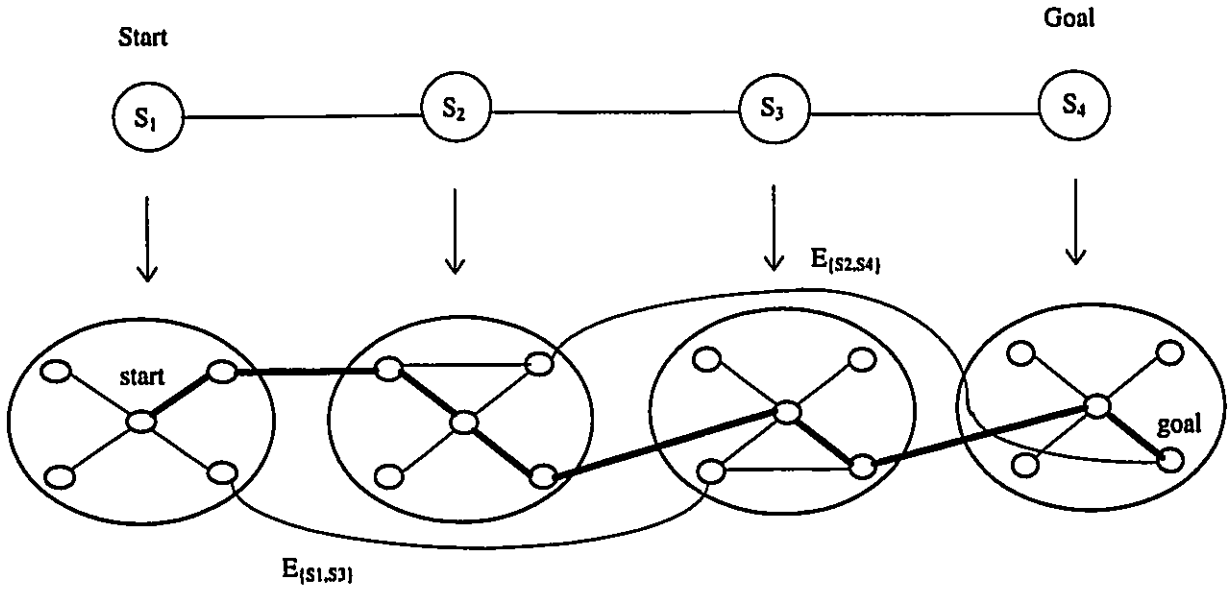


Fig. 3.5 Classical Refinement

3.2.2 Path Opportunism

The idea in this algorithm is to expand first those nodes of classes closer to the class containing the goal node (S_4) during the search. This technique does not demand an strict following of a sequence (S_1 S_2 S_3 S_4) as Classical Refinement does. If there is an opportunity to skip over part of the abstract solution, it will do so. Similar to Classical Refinement, once a particular class is reached, the search in its preceding classes in the sequence stops and the search continues in the new class. As we will see in the next

subsection this stopping condition disallows this technique to find better opportunities ($E_{[S2,S4]}$). This method is identical to Classical Refinement if no opportunities to skip ahead are present. See Fig. 3.6.

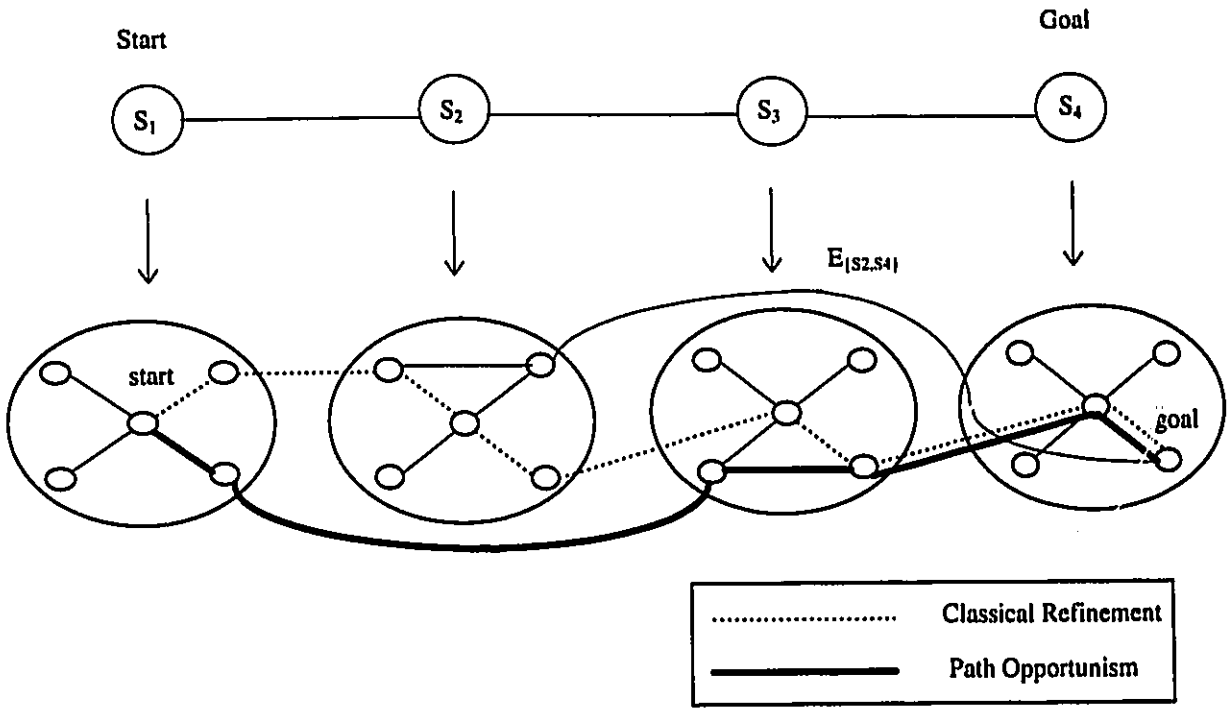


Fig. 3.6 Path Opportunism

3.2.3 Optimal Refinement

Optimal Refinement differs from Classical Refinement and Path Opportunism in that it generates the shortest possible refinement of the abstract path. Since we are looking for the shortest refinement, this technique does not demand an strict following of a sequence (S_1 S_2 S_3 S_4). Unlike the two previous techniques, once a particular class is reached the search in the preceding classes continues. This will guarantee the shortest refinement. The paths found with this method have equal or smaller length than those found with Path Opportunism. This is illustrated in Fig. 3.7.

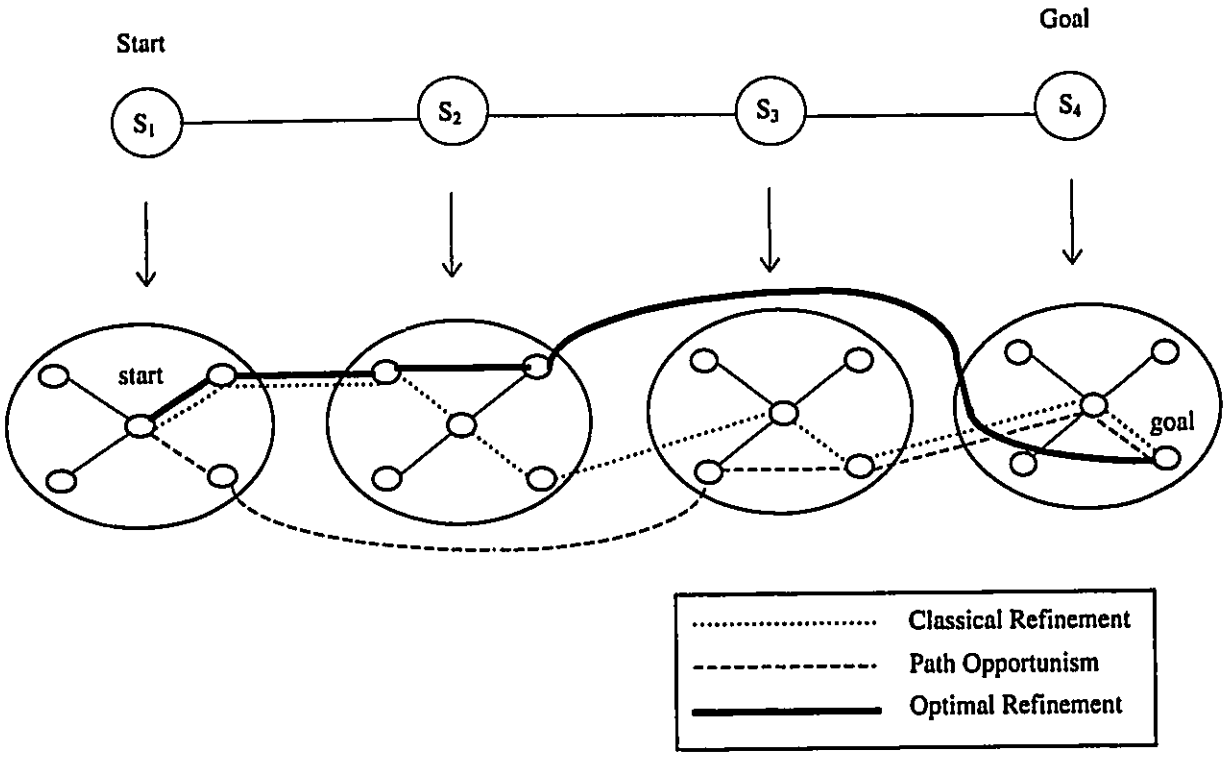


Fig. 3.7 Optimal Refinement

3.2.4 Alternating Search

In this technique the search is carried out by switching search direction in successive levels of the abstraction hierarchy. One level does forward search, the next does backward search, the third does forward search, and so forth. Unlike Bidirectional search, the searches in each level are carried out in one direction only. We will use "source" to refer to the root node of the search tree and "destination" to refer to the node that causes search to terminate. In a search tree, we know the distance from all nodes to the source. These distances represent an estimated heuristic of the distance to the goal for the next level below. In Fig. 3.8 two levels of the abstraction hierarchy are shown. The third abstract level shows the search tree. Each node of the search tree stores the distance from the start node (root of the tree). For those nodes in the second abstract level mapped to the nodes of the search tree, this information represents the estimated distance to the goal node. Then, the search in the second abstract level proceeds in the opposite direction (i.e. backward). In the same way, the search tree from the second abstract level

will generate information for the first abstract level and so on. Notice that not only information of the abstract path is cached.

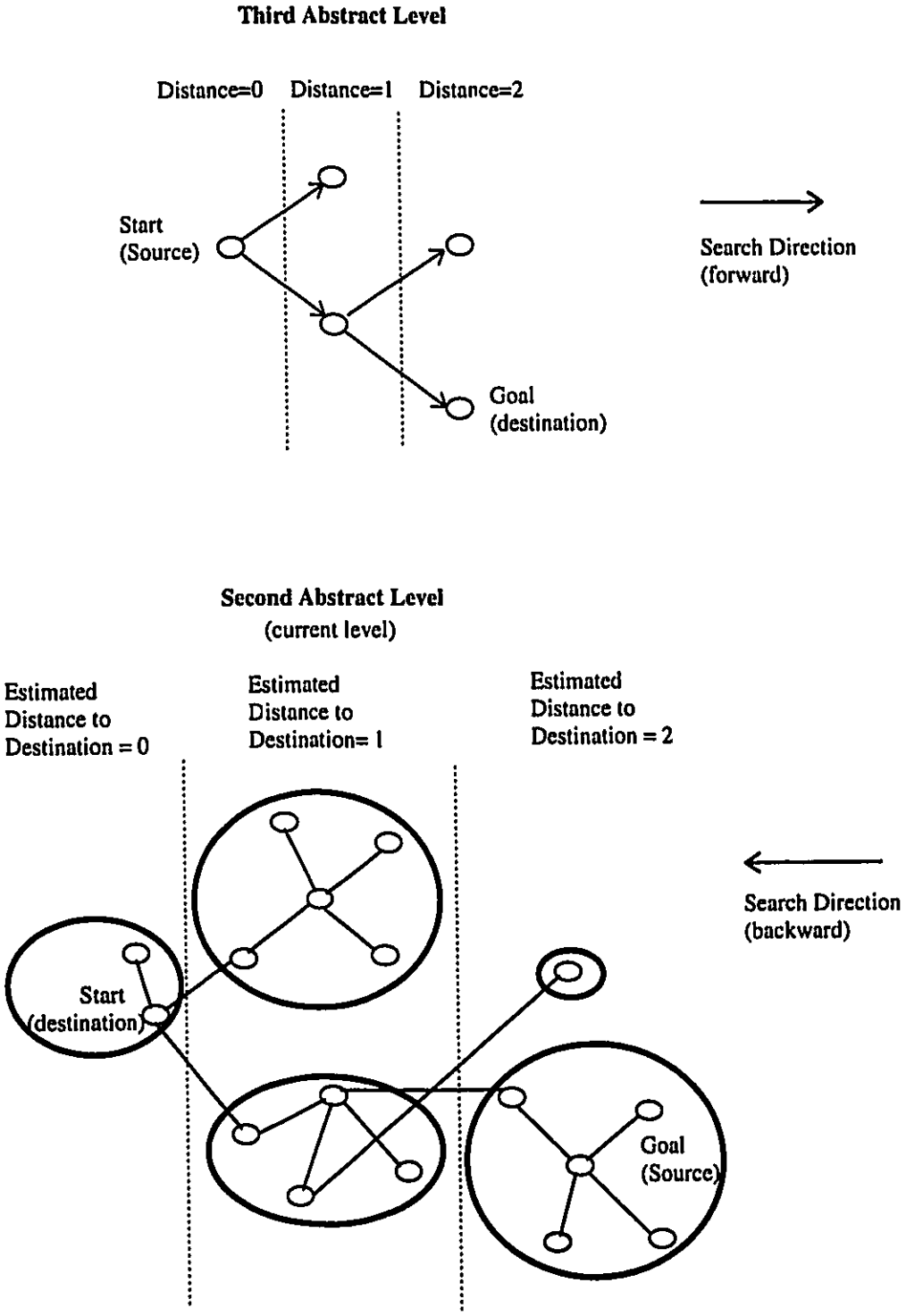


Fig. 3.8 Alternating Search

This idea of alternating search direction can be combined with the idea of opportunism. This combination is called Alternating Opportunism.

3.3 EXPERIMENTAL WORK

The motivation for this experimental work is to evaluate the performance of different search techniques: Classical Refinement (CR), Optimal Refinement (OR) and Alternating Opportunism (AO). For this study we will use two hub selection criteria and different diameters to build the abstract spaces where these techniques will be applied.

The two criteria for hub selection used are random hub and maximum hub. The diameter used were 2, 3, 4, 5, 6, 7. No orphans (classes with a single node) are allowed.

We used 100 pairs of generated randomly states. Each pair served as two different problem instances: <s,g> and <g,s>. The state spaces used were the Towers of Hanoi with 7 disks (2187 states), the Blocks World with 6 blocks (7057 states), the Permutation puzzle with 7 integers (5040 states), and the 5-puzzle, a 2x3 version of the 8-puzzle (720 states). We will only present in this section the results for the Towers of Hanoi state space. These results are typical of all the results for the other puzzles. We will indicate those special cases where the results of other puzzles differ from those of the Towers of Hanoi. See Appendix A for a description of the puzzles.

Our results for the Towers of Hanoi puzzle (T.O.H) are given in Table 3.1 and Table 3.2. The numbers in parentheses represent the results for random hubs. The numbers without parentheses represent the results for Max hubs. Table 3.1 contains all the path lengths for the three techniques. Table 3.2 contains the total amount of work done in order to find the results shown in Table 3.1. The "total work" includes the number of edges traversed as well as other overhead operations such as transferring information from one level of abstraction to another. Each operation increments the counter of total work by "1" unit. The optimal path and the total amount of work for Breadth-First search is shown in brackets beside the tables titles. See Appendix B for the results on the other puzzles.

T.O.H - Path Length (optimal = 66)						
	Diameter					
	2	3	4	5	6	7
CR	98 (91)	101 (95)	80 (82)	82 (86)	82 (95)	79 (88)
OR	88 (80)	77 (81)	72 (72)	72 (74)	72 (86)	69 (81)
AO	80 (75)	76 (83)	75 (77)	76 (78)	77 (79)	75 (79)

Table 3.1 Diameter Experiment for T.O.H - Path Length

T.O.H - Total Work (edges + overhead) (Breadth-First search = 3058)						
	Diameter					
	2	3	4	5	6	7
CR	894 (847)	867 (794)	776 (734)	752 (776)	777 (931)	802 (942)
OR	1048 (1021)	903 (957)	903 (870)	927 (924)	944 (1111)	950 (1106)
AO	767 (765)	711 (763)	751 (738)	740 (763)	765 (830)	828 (916)

Table 3.2 Diameter Experiment for T.O.H - Total Work

Small Diameter (Diameter = 2)

In general, paths are longer than those of higher diameter. CR generates the longest paths. AO generates better paths and does less work than CR and OR.

Medium Diameters (Diameters 3-6)

AO does about the same amount of work as CR and produces solution paths 10% shorter than CR's. OR's solutions are similar in length to those given by AO but AO

does less work than OR on most of the puzzles. In general, the amount of work for these diameters tends to increase as diameter increases.

Large Diameter (Diameter 7)

For all our state spaces except for Towers of Hanoi most of the states collapse into the same class, making abstraction almost useless. The amount of work for this diameter in some puzzles is not cost-effective. It is greater than the work done using Breadth-First search. In the Towers of Hanoi state OR gives shorter or equal paths length than CR's and in most of the puzzles it gives shorter or equal paths length than AO's. However, OR does more work.

From this experiment we can conclude:

1. Medium diameters produce good paths with a reasonable amount of work.
2. Random Hubs generate longer paths than Max hubs.
3. Alternating Opportunism (AO) is a good technique. It gives good paths with the same amount of work as Classical Refinement (CR) and is more consistent across different abstractions.

3.4 LIMITATIONS OF THE INITIAL ALGORITHMS

- Only unweighted graphs can be used by these algorithms, since the basic search in these algorithms is Breadth-first search, which is applicable only to unweighted graphs.
- Suboptimal paths are obtained.

The search in each level of the abstraction hierarchy is restricted to those nodes contained on the classes of the abstract solution path or the abstract search tree. This restriction limits the possibility of finding better paths. If a node is reached that is not contained in any of these classes, it will not be included in the search. This node might lead the search to a better path.

3.5 SOLUTIONS

There are a number of algorithms to search weighted graphs. These algorithms differ from one another in the evaluation function, which determines the search strategy. In the case of Dijkstra's algorithm this function returns the distance from the start node. Graph Traverser algorithm [Doran and Michie, 1966] considers the distance to the final destination or goal. The algorithm that we use is the A* algorithm, which combines these two functions by adding them.

In order to determine the distance to the goal we use the abstract space. This distance is calculated every time it is needed by searching in the abstract space. Therefore, if A* visits a node **n** for the first time, h(n) is calculated by searching in the abstract space from the class containing **n** to the goal class. The distance of each class in the abstract solution to the goal class is an admissible heuristic for the nodes contained in the classes.

Unlike the initial algorithms, search at the abstract level is done whenever a node is visited for the first time during the search. Thus, A* calculates all the abstract solutions necessary to generate the information needed to find an optimal solution.

Table 3.3 shows some of the results of our first implementation. The number of nodes expanded for the Blocks World is twenty one times higher than the nodes expanded with no abstraction. For the Permutation state space, it is four times higher and for the 5-puzzle, it is forty six times higher . The total amount of work is one hundred three times bigger than the total work with no abstraction for the Blocks World, eighteen times bigger for Permutation and ninety six times bigger for the 5-puzzle.

	Search Using Abstraction		Search Using no Abstraction	
	Nodes Expanded	Total Work	Nodes Expanded	Total Work
Blocks World	8362	242037	389	2344
Permutation	1232	42860	285	2434
5-puzzle	16044	181134	348	1884

Table 3.3 Initial Results - Search using Abstraction vs. Search using no Abstraction

The problem with this approach is that A* computes the heuristic information for every node it visits for the first time by searching in the abstract space. Thus, one search in the base level generates many searches at the first level of abstraction, and each of these searches generates more searches in the second level of abstraction and so on.

In order to reduce the amount of total work and number of expanded nodes we have to reduce the number of times we compute the heuristic information by searching at the abstract level. Our solutions are:

1. Reuse of heuristic information from previous search.
2. Using other less expensive sources to generate heuristic information.
3. Deferring the calculation of the heuristic information until it is necessary (Lazy Evaluation)
4. Alternating search direction during the search, in order to generate more heuristic information between the levels.
5. Sacrificing the optimality of the path solution in order to reduce the search efforts (Selective Method).

These solutions will be explained in the next Chapter.

Chapter 4

Efficient computation of heuristic information

The computation of heuristic information by searching in the abstract space (h(n)) is "expensive". In this Chapter we investigate five approaches to reducing the number of times this information is computed.

One approach is to reuse heuristic information values. These values can be used if the goal state of a new search is the same as the goal state of the most recent search done in that level. The search at the base level from S1 to G1 will originate many searches at the abstract levels in order to find the necessary h(n) values. For some of our techniques the reuse of information is fundamental, that is the case of First Time Alternation define in subsection 4.3.1 and P-G define is subsection 4.1.2. All the searches in all the abstract levels will have the same goal state, so the heuristic information will potentially be reused many times. For some abstract levels, these searches might have the same goal state as the most recent search done in that level making possible the reuse of information. There is a special heuristic which is also reused that is worth mention. We call this heuristic, perfect heuristic. When a search at a given level is finished the shortest path to the goal is known for some nodes. If the next search has the same goal, these paths can be reused. Perfect heuristic represents the exact distance to the goal node for each node on the shortest path. The computation of h(n) is not necessary when the heuristic of a node is perfect.

A second approach is to use additional low cost sources to generate heuristic information. These sources are: minimum edge weight and P-G (explained below). Each of these sources generates admissible and monotonic heuristic information when they are individually used. These sources can be used in combination with our first source (abstract distance) preserving the admissibility property. It is not known if the monotonic property holds for all combinations of these sources.

There is no guarantee that one source is better than the other. One might be better in different circumstances. When they are combined we use the best heuristic information generated from all the sources combined by calculating the maximum value. This value represents more information and according to theorem 2.5.5 the number of nodes expanded should decrease.

A third approach is to defer the computation of the abstract distance until it is necessary. The idea behind this approach is to save calculation efforts. The computation of abstract distance requires search in the abstract space. Sometimes this search exceeds the savings offered by the heuristic and in the worst case the nodes for which the heuristic information is computed do not play any important role during search. We can postpone computing abstract distance by using only the less expensive sources when we first put the node on the OPEN list.

A fourth approach is to change the search direction. In subsection 3.2.4 and section 3.3 we saw that alternating search direction is a good technique. Although the conclusion was drawn from using this technique with Breath-First search, we believe that similar benefits will occur with A*.

Our final approach is called the Selective Method. This method discards nodes during search in order to reduce the h(n) computation. The disadvantage of this method is that the optimality of the path is sacrificed.

4.1 MULTIPLE SOURCES OF HEURISTIC INFORMATION

The computation of heuristic information can be relatively inexpensive. This computation cost depends on the source used in order to generated the heuristic value. To reduce this cost and generate more accurate heuristic information we introduce two sources. In addition to our original source (the length of the abstract solution). The two new sources are: minimum edge weight and P-G.

The minimum edge weight emerging from a node (except for the destination or goal node) provides an underestimated of the distance from the node to the destination or goal node. This heuristic information is inexpensive to calculate and is useful for nodes close to the goal.

4.1.1 Minimum Edge Weight Heuristic

Combining the minimum edge weight heuristic with our original heuristic (abstract solution length) produces a more accurate heuristic information for some nodes, especially those nodes close to the goal node. To illustrate this, consider the example in fig. 4.1 through 4.3.

Fig. 4.1 shows the values of minimum edge weight heuristic.

In Fig. 4.2 the heuristic information is generated from the distance of each class to the goal class. All the nodes contained in a class have the same heuristic information. Each class is circled in bold.

In Fig. 4.3 the heuristic information from Fig. 4.1 and Fig. 4.2 is combined by taking the maximum of the two values. For those nodes forming the first triangle on the left, the abstract distance heuristic is better than the minimum edge weight heuristic. But for the other two triangles, the opposite is true: the minimum edge weight heuristic is better.

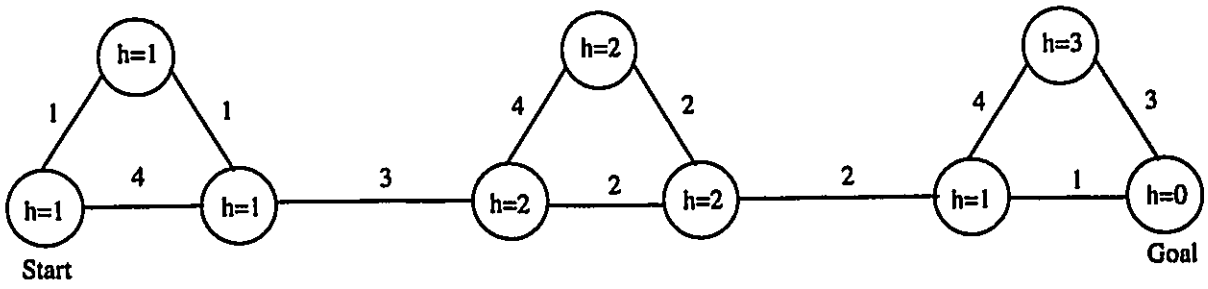


Fig. 4.1 Heuristic Based on Minimum Edge Weight

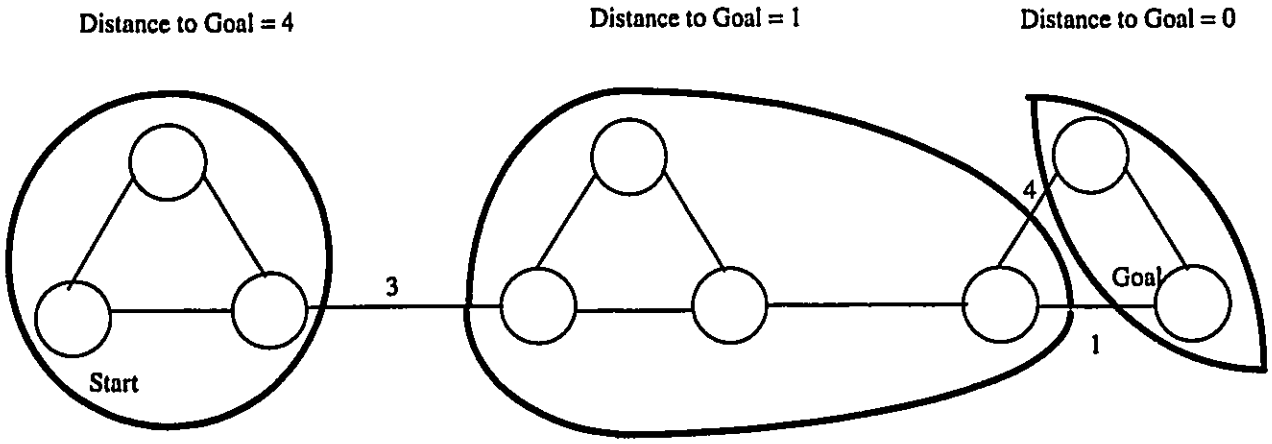


Fig. 4.2 Heuristic Based on Abstract Distance to Goal

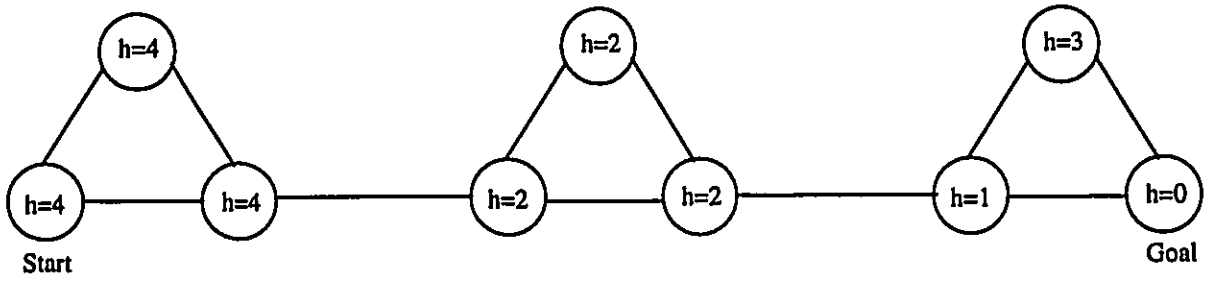


Fig. 4.3 Combined Heuristic

4.1.2 P-G Heuristic

Heuristic information available in the abstract spaces usually will be reused since successive searches in the same abstract level will often have the same goal but different start nodes. Each of these searches will have a solution path. We denote the length of the solution path by P. Every node visited during search will store a G[n] value which represents the distance from the start node to **n**. P is equal to 4 in the example of Fig. 4.4

Another source of heuristic information to compute is given by the formula P-G[n].

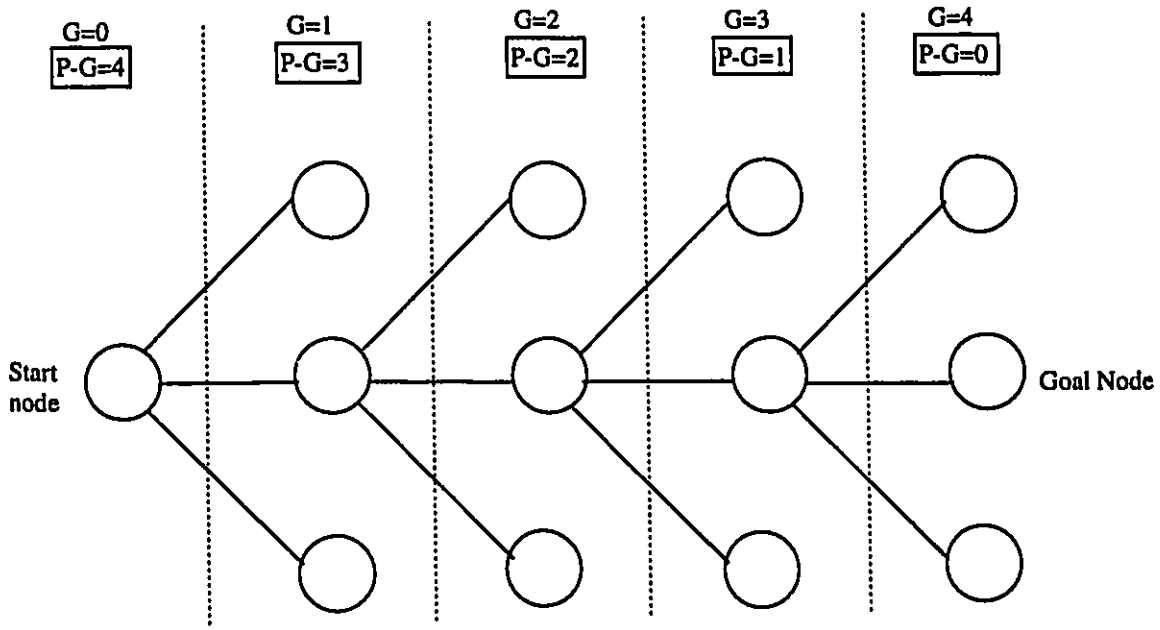


Fig. 4.4 P-G Source of Heuristic Information

This formula is applied at the end of the search in each level of abstraction to all the nodes visited (OPEN and CLOSED nodes) during the search. Since G[n] can be an exact value or an overestimate, P-G[n] is an underestimate of the distance from **n** to the goal and therefore this heuristic is admissible. For those nodes on the solution path P-G computes the exact distance to the goal (h*(n)).

The combination of P-G with our previous sources is not known to preserve the monotone property. In our experimental work the use of P-G never caused any problem because of the lack of monotonicity. The lack of this property will prevent some CLOSED nodes to have an exact G[n] value. Therefore when their exact G values is found the search tree has to be readjusted.

4.2 LAZY EVALUATION

In ordinary A* the heuristic information for a node, h(n), is computed when a node is visited for the first time by searching in the abstract level. This method is known as eager evaluation. An alternative to this method is lazy evaluation. In lazy evaluation

search at the abstract level to compute h(n) is not done immediately. Instead **n** is added to OPEN using its current heuristic value stored in H[n]. Only when **n** reaches the top of OPEN its h(n) value is computed by searching in the abstract level. Once h(n) is computed, H[n] is updated and **n**'s position on OPEN is adjusted accordingly. The top node is removed from OPEN only if h(n) is known. Therefore, nodes are removed from OPEN in the same order by lazy evaluation as they are with eager evaluation. Thus, this approach guarantees optimal paths. Consider the example in Fig. 4.5.

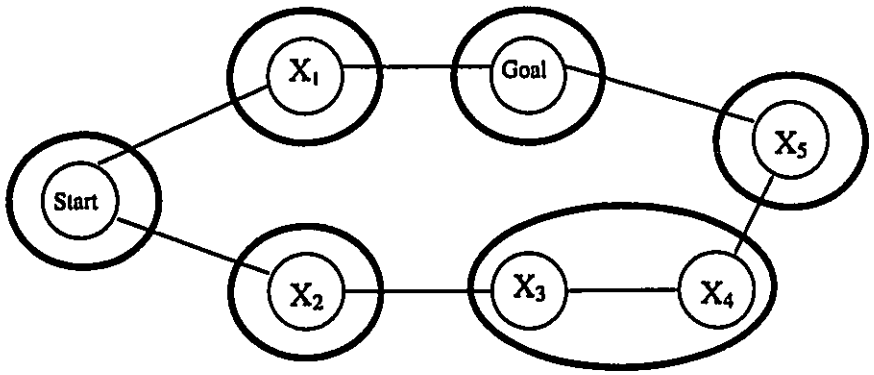


Fig. 4.5 Eager vs. Lazy Evaluation

Outer circles represent the abstract nodes, inner circles represent nodes at the base level.

In the previous approach, to put a node in the OPEN list, we need to have its heuristic information computed from the abstract level. To add X_1 and X_2 to OPEN we have to compute $h(X_1)$ and $h(X_2)$. This might imply two searches at the abstract level at this moment. One of them might take a lot of search effort if the heuristic has a very large value (X_2) and in the end it might never be needed because we may find a path to the Goal before X_2 gets to the top of OPEN.

Lazy evaluation does not need the $h(X_1)$ and $h(X_2)$ to add these nodes to the OPEN list. This information will be computed later when each node reaches the top on the OPEN list. X_1 and X_2 are added to the OPEN list with their current heuristic information. Assuming that X_1 reaches the top of the list before X_2, $h(X_1)$ will computed

by searching in the abstract level. If a path to the Goal is found before X_2 gets to the top of OPEN then $h(X_2)$ will never have been computed and search effort will be saved.

4.3 CHOICE OF SEARCH DIRECTION

In ordinary A* all searching is done in the forward direction, from the start state towards the goal state.

In section 3.2.4 we saw there are advantages to alternating search direction from one level to the next in the abstraction hierarchy. One of these advantages is the generation of more information in a single abstract search. Alternating Search was proven experimentally to be a good technique in section 3.3. This conclusion was obtained with Breadth-First search but the same technique can be used with A*.

4.3.1 First Time Alternation

First Time Alternation is a variation on Alternating Search. So far, the search in each level of the hierarchy for all of our techniques proceeds in one specific direction i.e. either forward or backward. In First Time Alternation, the search direction in each level will sometimes go forward, and sometimes go backward. In particular, if the goal of the current search is the same as the goal of the previous search then forward search is used. If the goal is different, backward search is used. This idea of alternating search direction in a single level will generate more accurate information for the level where the search direction is alternated to help subsequent searches having the same goal node but different start node.

4.4 TABLE OF VERSIONS

In the preceding sections we have presented four approaches to improve search by reducing the number of times h(n) has to be computed: those four approaches are: reuse of heuristic information, additional sources to generate heuristics, lazy and eager evaluation, and change of search direction. These approaches can be combined in many

different ways. We have considered just five versions, which we will refer to as V1 through V5. These are described in Table 1.1.

	Versions (V1...V5)				
	V1	V2	V3	V4	V5
Minimum Edge Weight	X	X	X	X	X
Abstract Path Solution	X	X	X	X	X
P - G	-	-	X (only in current level)	X (current level and next level down)	X (current level and next level down)
Reuse of Heuristic Information	X	X	X	X	X
Search Direction	No Alternation	Alternation	No Alternation	No Alternation	First Time Alternation
Evaluation (Eager/Lazy)	Eager	Eager	Lazy	Lazy	Lazy

Table 4.1 Versions

An "X" in the three first rows entry of this table means the source used in the version. In V3 the heuristic information computed from the source P-G is stored in the level where the search is being done but not passed down to the level below. In V4 and V5 P-G information is stored in the current level and passed to the level below.

Each of the versions can either use or disregard the Selective method explained in the next section.

4.5 SELECTIVE METHOD

Classical Refinement and Classic A* always compute an abstract path from start to goal. The techniques differ when they visit a node whose h(n) value is not known. Classic A* always goes up to the abstract level to compute h(n) while Classical Refinement never does. The computation of h(n) values can be quite expensive since this

computation implies finding other abstract solutions that will also be refined. One way of reducing this computation is to calculate h(n) values for some nodes but not for others. This pruning method can be done in many ways. One alternative is to discard those nodes that cannot be on a path to a goal node. In order to do so, we need to have knowledge about the problem domain. Another alternative is to randomly choose those nodes to be discarded. Although this is not necessarily the best alternative, it is the one we tried in this thesis.

This method uses a probability parameter P. When a node is visited whose h(n) value is not known a decision to keep or discard **n** is made randomly. With P being the probability that **n** is kept. This method behaves like A* when P=1 and like Classical Refinement when P=0.

The drawback of this method is that the optimality of the path is not guarantee. Nodes thrown away may be on the best path to the goal. On the other hand, for many "real" problems a near-optimal path is also a good solution.

The implementation of this method differs slightly depending on whether the computation of h(n) is eager or lazy.

In eager evaluation the probability parameter P is used to decide for each neighbor node visited for the first time, whether the node will be included in OPEN. When a node is included in OPEN, its h(n) is computed.

In lazy evaluation each neighbor node visited for the first time is included in OPEN. When a node reaches the top of the OPEN list and its h(n) value is not known we use the probability to decide if the h(n) value will be calculated. If the probability indicates that h(n) will not be computed then the node is eliminated from OPEN at this point.

4.6 CLASSIC A* vs. OUR VERSIONS

In Fig. 4.6 we present side by side the Classic A* and our modified A* using lazy evaluation for V1, and V2. In Fig. 4.7 we present the Classic A* and our modified A*

using eager evaluation for V3, V4, and V5. The differences between Classic A* and our versions are highlighted in bold. In both versions the selective method has been included. To implement this method we use the following construct:

Choose Randomly:

with probability *<pvalue>* do:

<do-statements>

Otherwise:

<other-statements>

Where *<pvalue>* is a value in the interval [0,1] and represents the probability parameter P. It first generates a random value. If the random value is less than *<pvalue>* then it executes the *<do-statements>* else it executes the *<other-statements>*.

Although we do not have a formal proof, we assume that the combined heuristics are monotone. Therefore, based on theorem 2.5.4 the steps related to the CLOSED list in the Classic A* are not applicable in our versions.

Finally in step 9 information generated by the search is cached as appropriate to each version. When the search at the base level is finished, the solution path is traced back in a similar way as it is done in step 9 of the Classic A* algorithm.

Fig. 4.6 Classic A* vs. Our A* Version with Lazy Evaluation

Input : S = Start node G = Goal node **Output : A path from S to G**	**Input : S = Start node G = Goal node** **P = Probability** **Output : A path from S to G**
Classic A*	Modified A* (Lazy)
1. OPEN ← {S}; CLOSED ← ϕ;G[S] ←0; PRED[S] ←NULL	1. OPEN ←{S}; G[S] ← 0; PRED[S] ← NULL
LOOP : 2. if OPEN is empty then found ←false, go to 9 3. Select a node X on OPEN for which F[X] is the least (ties are broken arbitrarily but always in favor of goal nodes)) 4. If X is a goal then found ← true, go to 9	LOOP: 2. if OPEN is empty then found ←false, go to 9 3. Select a node X on OPEN for which F[X] is the least (ties are broken arbitrarily but always in favor of goal nodes) 4. If X is a goal then go to 9

5. Remove X from OPEN and put it on CLOSED
6. Generate the set of successors of X
7. for each n in successors do
8. if n is not already on OPEN or on CLOSED
   then
   (a) PRED[n] ← X
   (b) G[n] ←G[X] + distance(X,n)
   (c) H[n] ←h(n)
   (d) F[n] ←G[n] + H[n]
   (e) add n to OPEN
   else if n is already on OPEN or CLOSED and
   G[X] + distance(X,n) < G[n]
   then
   (f) G[n] ←G[X] + distance(X,n)
   (g) F[n] ←G[n] + H[n]
   (h) PRED[n] ←X
   (i) If n on CLOSED remove it from there and
   insert on OPEN

ENDLOOP

---

9. If found is false then exit with "failure
   else obtain the solution by tracing a path along the pointers from X back to S.

else
- Check H[X]
- If h[X] in unknown and h*[X] is unknown
then
- Choose Randomly
with probability P do:
- Compute h(X) by searching at the abstract level
- H[X] ← h(X)
- F[X] ← G[X] + H[X]
- Reorder the OPEN list
otherwise: Remove X from OPEN
- Go to 3
else go to 5

5. Remove X from OPEN and **mark it as closed**
6. Generate the set of successors of X
7. for each n in successors do
8. if n is not already on OPEN and **it is not marked as closed**
   then
   (a) PRED[n] ← X
   (b) G[n] ← G[X] + distance(X,n)
   (c) **NOT APPLICABLE**
   (d) F[n] ← G[n] + H[n]
   (e) add n to OPEN
   else if n is already on OPEN and
   G[X] + distance(X,n) < G[n]
   then
   (f) G[n] ← G[X] + distance(X,n)
   (g) F[n] ← G[n] + H[n]
   (h) PRED[n] ← X
   (i) **NOT APPLICABLE**

ENDLOOP

---

9. If found is false then exit with "failure
   else
   9.1 Cache the value for each node on the solution path of its distance to the goal node (in the current level)and pass these values down to the next level to the children of theses nodes.
   9.2 Cache values in the current level obtained from P-G for the rest of the nodes explored during search . In V4 and V5 pass these values down to the children of these nodes.

For V5 step 9 is as follows:
- If the current search direction is forward
then 9.1 and 9.2
else Cache values for all the nodes in the current state-space :
If the node is marked as closed or the node is the goal X
then Cache its G value and pass it down to its children
else clear values stored in node

Fig. 4.7 Classic A* vs. Our A* Version with Eager Evaluation

Input : S = Start node G = Goal node **Output : A path from S to G**	**Input : S = Start node G = Goal node** **P = Probability** **Output : A path from S to G**
Classic A*	Modified A* (Eager)
1. OPEN ← {S};CLOSED ← ϕ;G[S] ←0; PRED[S] ←NULL	1. OPEN ←{S}; G[S] ← 0; PRED[S] ← NULL
LOOP : 2. if OPEN is empty then found ←false, go to 9 3. Select n node X on OPEN for which F[X] is the least (ties are broken arbitrarily but always in favor of goal nodes)) 4. If X is a goal then found ← true, go to 9 5. Remove X from OPEN and put it on CLOSED 6. Generate the set of successors of X 7. for each n in successors do 8. if n is not already on OPEN or on CLOSED then (a) PRED[n] ← X (b) G[n] ←G[X] + distance(X,n) (c) H[n] ←h(n) (d) F[n] ←G[n] + H[n] (e) add n to OPEN else if n is already on OPEN or CLOSED and G[X] + distance(X,n) < G[n] then (f) G[n] ←G[X] + distance(X,n) (g) F[n] ←G[n] + H[n] (h) PRED[n] ←X (i) If n on CLOSED remove it from there and insert on OPEN ENDLOOP	LOOP: 2. if OPEN is empty then found ←false, go to 9 3. Select a node X on OPEN for which F[X] is the least (ties are broken arbitrarily but always in favor of goal nodes) 4. If X is a goal then go to 9 5. Remove X from OPEN and mark it as closed 6. Generate the set of successors of X 7. for each n in successors of X 8. If n is not already on OPEN and it is not marked as closed then - Check H[X] - If h*[X] in known or h[X] is known then (a) PRED[n] ← X (b) G[n] ← G[X] + distance(X,n) (c) NOT APPLICABLE (d) F[n] ← G[n] + H[n] (e) add n to OPEN else -Choose Randomly - with probability P do - PRED[n] ← X - G[n] ← G[X] + distance(X,n) - Compute h(X) by searching at the abstract level - H[X] ← h(X) - F[n] ← G[n] + H[n] - add n to OPEN otherwise ignore n else if n is already on OPEN and G[X] + distance(X,n) < G[n] then (f) G[n] ← G[X] + distance(X,n) (g) F[n] ← G[n] + H[n] (h) PRED[n] ← X (i) NOT APPLICABLE ENDLOOP
9. If found is false then exit with "failure else obtain the solution by tracing a path along the pointers from X back to S.	9. If found is false then exit with "failure else In V1: For the children of those nodes on the solution path

	cache the value of the distance of their parents to the goal node. In V2: Cache the G value into the children of the nodes marked with the close flag and into the children of X.

Chapter 5

Experimental work

5.1 MOTIVATION

Recall from Chapter 1 that our aim in this thesis is to reduce the amount of work needed in order to find a near optimal or optimal solution path. To do so, we first abstract the original problem to simplify it and then we search the abstract spaces using a heuristic search algorithm. In our first version, we looked for an optimal path using the standard A* algorithm. This idea was not sufficient to achieve our aim, as was shown in table 3.3. Numerous approaches described in Chapter 4 were developed to overcome the weaknesses of this version while preserving optimality.

Experimental studies are needed to determine how well our techniques perform. This will allow us to decide which of our five versions (V1,...,V5) is the best and whether any of them outperforms search without abstraction. The best of the versions will be used to study the Selective Method. This technique differs from the others in that it does not preserve the optimality of the path in order to gain speed. The Refinement techniques presented in Chapter 3 are based on this idea. With the refinement techniques the solution paths were between 20% and 40% longer than optimal and the speed-up was very good. Using our Selective method, we hope to generate paths slightly longer than optimal with a big speed-up. Secondly, we determine which abstraction techniques is best for each version of A*. A second goal of the experiment is to study our versions V3, V4, and V5.

5.2 STATE-SPACES

The state-spaces used in these experiments are presented in Table 5.1 and fully described in Appendix A.

State-space	Number of States	Number of Edges	Branching Factor	
			Maximum	Average
Blocks World (5 blocks)	866	2090	5	2.41
Permutation (integers 1-6)	720	3600	5	5
5-puzzle	720	1682	3	2.33
T.O.H (7 disks)	2187	6558	3	2.99
KL-2000	2736	28824	66	10.53

Table 5.1 State-Spaces

Appendix A also gives additional characteristics of the abstract spaces created for the state-spaces by two different abstraction methods belonging to the star family (max hubs and min hubs).

In our experiments, these spaces were treated as undirected graphs, in which the weight from node A to node B is the same as from node B to node A (see Fig. 5.1). In our first four experiments the weights for all the edges in all the state-spaces are equal to "1". In our last experiment we assign random weights between 1 and 10.

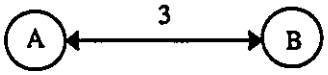


Fig. 5.1 Undirected Graphs with Weights

5.3 EXPERIMENTAL SETUP

This experimental work is divided into different experiments:

Experiment #1

We show the results of the eager A* versions with and without alternating search direction (versions V1 and V2 in Table 4.1). This experiment showed that the amount of work in either case is unacceptably large. Therefore, our studies did not go further in this direction.

Experiment #2

The purpose of this experiment is to compare the versions V3, V4, and V5 (Table 4.1) for A*. This experiment revealed how much improvement was gained by adding the ideas described in the previous chapter to the algorithms.

Experiment #3

From experiment #2, V5 is the best of our versions. We used it in this experiment to evaluate the selective method and the effects of weighing G and H differently (as described in subsection 2.6.2). The results indicate that sacrificing the optimality of the path until an "acceptable" point could reduce considerably the search efforts.

Experiment #4

This experiment compare two criteria for selecting hubs when building the abstraction. Experiment #2, which use the maximum Hub criterion is repeated using minimum Hub. Maximum Hubs is found to give better results.

Experiment #5

We assign random weights between 1 and 10 to the edges and re-run the experiment #2, to see if its conclusions extend to graphs with non uniform weights.

5.4. PERFORMANCE MEASUREMENTS

For each space 100 pairs of states were randomly chosen. Each pair <s,g> represents two problem instances <s,g> and <g,s>. These gives rise to 200 pairs in total. The same 200 pairs for each state-space were used for all the different experiments. The results presented in this chapter are averaged over the 200 problems.

Solution Length:

Each pair <s,g> has a solution length represented by the sum of the weights of the edges connecting s to g.

Nodes Expanded:

This measurement is the number of nodes taken off the OPEN list during search. This counter incremented in step 5 is of Fig. 4.6 and Fig. 4.7.

Arcs Traversed:

When a node is expanded, all its successors are generated by traversing arcs. The number-of-arcs-traversed counter is incremented every time a successor is generated in step 7 of Fig. 4.6 and Fig. 4.7.

Caching (Current Level):

When the search is completed, information generated from the search is saved in the nodes at the same level. The caching measurement counts the number of nodes into which information is cached. This caching is done in step 9 of Fig. 4.6 (for V1, and V2 this is not applicable). We also include in this counter the cost of initializing and re-initializing the graph.

Marking Down:

The information generated by the search in one level represents the heuristic information that will guide the search at the next level below. This information needs to be passed down to that level. One way of passing this information is to mark it down when the search has finished. This marking is done in step 9 of Fig. 4.6 and Fig. 4.7. In Fig. 5.2 the counter for marking down is equal to five.

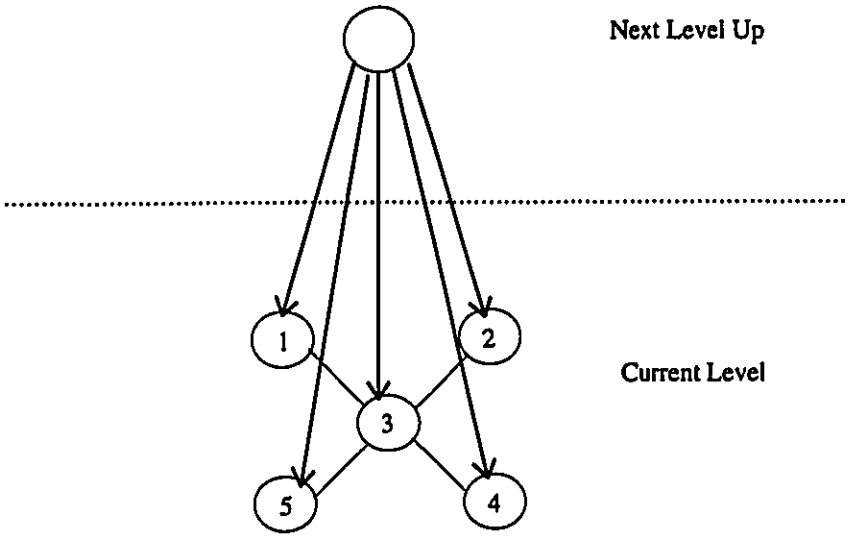


Fig. 5.2 Marking Down

Looking Up:

The information generated by the search in one level can be stored in that same level and it can be accessed by the next level below whenever it is needed. This alternative way of using information generated in one level from its next level below will save the effort of passing information down that will not be used later in search. In Fig. 5.3 the counter for looking up is equal to one. Those other nodes in the current level do not need this information. For the Lazy versions (V3, V4, and V5) this is done in step 4 of Fig. 4.6. For the eager versions (V1, and V2) it is done in step 8 of Fig 4.7.

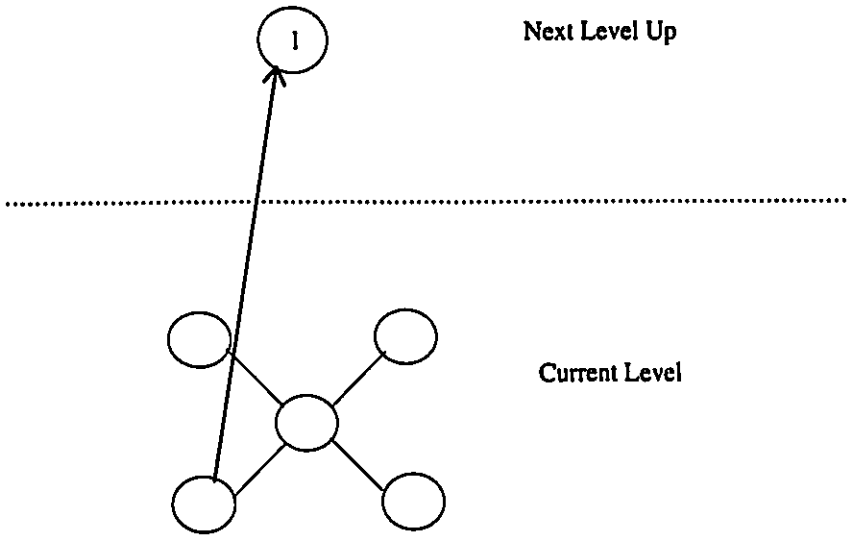


Fig. 5.3 Looking Up

Work (Marking Down):

This parameter is the sum of Arcs Traversed, Nodes Expanded, Caching (Current Level) and Marking Down. It represents the total work.

Work (Looking Up):

This parameter is similar to Work (Marking Down), with the difference that instead of adding Marking Down to the rest we add Looking Up.

For one particular implementation and one machine CPU time represents the correct performance measure but to compare different implementations, or runs on different machines, CPU time is not appropriate because it is sensitive to programming details and machines speeds. Therefore, we do not measure CPU time in our experiments. Instead, we measured the work just defined for comparison. This overcomes all the problems with the CPU time. To decide if work is representative of CPU time we measured the work and CPU time in different runs of the same implementation on the same machine for a set of 5-puzzle problems. Fig. 5.4 shows the plot of work against CPU time. The graph shows that the work parameter is proportional to the CPU time.

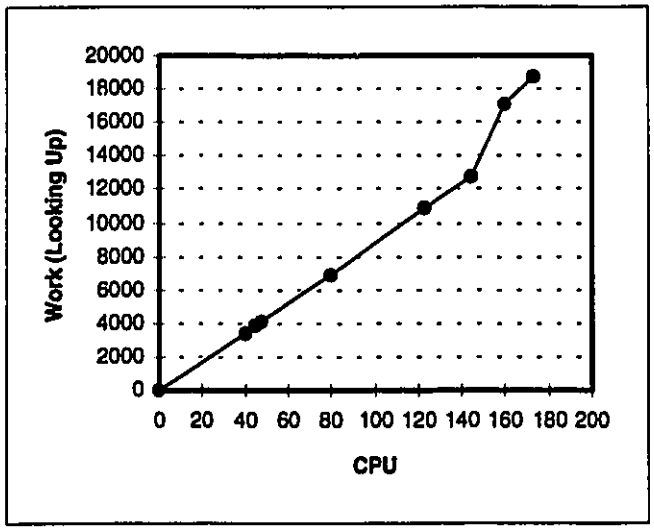


Fig. 5.4 Work vs. CPU

5.5 RESULTS AND DISCUSSION

This section is divided into five subsections. Each subsection describes the results of one experiment.

The results for search without abstraction (NAB) are given in Table 5.2. The H[x] value of all the nodes in the original space were initialized to 1 except for the goal node which is equal to 0. This initialization is done whenever the goal node of the pair currently being searched is not equal to the goal node of the previous searched pair (reuse of heuristic information). When there is no abstraction hierarchy, the algorithmic difference between V1 and V2 is never executed, so their performance is identical. The same is true of V3, V4, and V5. There is one algorithmic differences between the two first versions (V1, and V2) and the three last versions (V3, V4, and V5) that does affect behavior even when there is no abstraction hierarchy. The two first versions do not cache P-G values while the other versions do. The difference in work between versions is precisely the amount of P-G caching. The algorithmic differences does not affect nodes expanded or arcs traversed. The counters for Marking Down and looking up are both "zero" in all the versions. Thus, both ways of measuring Total Work are the same.

No Abstraction

			For V1, V2	For V3, V4, V5	
	Nodes Expanded	Arcs Traversed	Total Work	Total Work	Solution Length
Blocks World	389	1089	2344	2849	9.92
Permutation	285	1428	2434	2916	4.71
5-puzzle	348	816	1884	2261	19.29
T.O.H	1068	3205	6461	7557	67.35
KL-2000	1235	14527	18498	19956	9.87

Table 5.2 Results for Search with No Abstraction

5.5.1 First Experiment

In this experiment we evaluate the two A* versions V1 and V2. V1 uses forward search and V2 uses alternating search. The objective of this experiment is to determine if there is any advantage of using alternating search over forward search. For more complete results of the three state-spaces see Appendix C. The abstract spaces are built using Max Hubs with diameter two and no orphans (a class with a single node) are allowed.

V2 improves search in the base level in the same way as V1 does. The number of nodes expanded in the base level are the same in both versions and it is less than NAB. The number of nodes expanded at the base level for the Blocks World is 62.71% less than NAB, the Permutation state-space is 62% less, and the 5-puzzle is 33.83% less.

V1 always does more work than V2 except for the Permutation state-space (see Table 5.3) but in both versions the work is greater than the work done by NAB. This confirms the conclusions drawn from table 3.3 which showed that the computation of heuristic by searching in the abstract level increases the overall search effort.

Although in all of our other experiments the total work for looking up is always lower than the total work for marking down, for V1 the opposite is true. The reason for this is that information from the most recent search in an abstract level is reused if the abstract goal is the same. Therefore no marking down is done since no search is required to obtain this information. However if this information is stored in the next level up it is

necessary to look it up every time it is needed. Since V1 and V2 evaluate nodes in an eager fashion, this information has to be looked up for every successor.

For our other experiments we will only show the total work for looking up. For results on total work for marking down see Appendix C.

	Total Work					Nodes Expanded		Solution Length
	V1		V2		No Abs.	Base Level	No Abs.	
	Looking up	Marking Down	Looking up	Marking Down				
Blocks World	56799	50924	30043	35748	2344	145	389	9.92
Permutation	31019	26676	36481	41841	2434	109	285	4.71
5-puzzle	33480	30671	15646	19571	1884	230	348	19.29

Table 5.3 Version V1 and Version V2 for A*

5.5.2 Second Experiment

The abstract spaces were created using Max Hubs with diameter two and Max Hubs with diameter four. No orphans were allowed. In general from these results (see Table 5.4, Table 5.5, Table 5.6) we can see that for diameter four less work is done and fewer nodes are expanded than for diameter four. Considering the total nodes expanded in all levels searched for diameter four, the totals are less than using diameter two except for KL-2000 in V4 and V5, and the Permutation state-space in V5. In these cases the totals of nodes expanded are not more than 7% bigger than the totals for diameter two.

However, diameter four expands more nodes in the base level. This may be because the classes in the next abstract level are bigger. Therefore, the heuristic information generated from a class will be the same for those nodes in the base level mapped to it. Then, the search at the base level might not benefit from the heuristic but it will reduce the computation of h(n). In general, the abstraction hierarchy created by using diameter four behaves better than the abstraction hierarchy created by using diameter two.

	Total Work			Nodes Expanded				
	No Abs.	V3		No Abs.	V3 - Base Level		V3 - All Levels	
		D = 2	D = 4		D = 2	D = 4	D =2	D = 4
Blocks World	2849	24893	8820	389	141	233	431	302
Permutation	2916	12374	3372	285	78	170	215	186
5-puzzle	2261	15110	6252	348	229	287	578	418
T.O.H	7557	72409	22226	1068	702	861	3206	1741
KL-2000	19956	55687	21804	1235	729	1011	1119	1078

Table 5.4 Version V3 for A* - Max Hubs

	Total Work			Nodes Expanded				
	No Abs.	V4		No Abs.	V4 - Base Level		V4 - All Levels	
		D = 2	D = 4		D = 2	D = 4	D = 2	D = 4
Blocks World	2849	17858	8478	389	124	228	355	294
Permutation	2916	9985	3380	285	72	170	186	186
5-puzzle	2261	13937	5981	348	225	281	554	407
T.O.H	7557	70765	21885	1068	698	857	3141	1721
KL-2000	19956	47415	21450	1235	683	999	1020	1062

Table 5.5 Version V4 for A* - Max Hubs

	Total Work			Nodes Expanded				
	No Abs.	V5		No Abs.	V5 - Base Level		V5 - All Levels	
		D = 2	D = 4		D = 2	D = 4	D = 2	D = 4
Blocks World	2849	17294	8317	389	128	230	336	290
Permutation	2916	9713	3375	285	71	170	176	186
5-puzzle	2261	12714	5504	348	226	283	512	386
T.O.H	7557	61212	19639	1068	699	858	2903	1571
KL-2000	19956	44433	21261	1235	694	1003	997	1058

Table 5.6 Version V5 for A* - Max Hubs

The results allow us to evaluate the benefit gained by adding new features from one version to the other. For these three versions where consider (V3, V4, and V5). These versions form a sequence of additions of features. V4 is derived from V3 by adding the caching of the P-G into the children of the nodes for which P-G is applied. V5 is derived from V4 by adding the idea of the first time alternation.

V5 expands the least total number of nodes and does the least amount of work among V3, V4, and V5. The transition from V3 to V4 causes a decrease between 2.2% and 28% in the amount of work for diameter two but very little decrease (1.5 - 4.3%) for diameter four. And the transition V4 to V5 causes a decrease between 2.7% and 13.4% in the amount of work for diameter two and a decrease between 0.14% and 10.2% for diameter four. Taking these together, V5 does between 15.4% and 31% less work than V3 for diameter two and between 2.49% and 12 % for diameter four. A similar tendency of decrease occurs with the total nodes expanded for most of the state-spaces.

The amount of work in all the state-spaces is bigger that the total work with no abstraction. For both diameters the total number of nodes expanded by V5, is smaller than the number of nodes expanded with no abstraction in the Blocks World, Permutation and KL-2000. In conclusion, V5 reduces the search effort compared to classic A* but the excessive computation of heuristics by searching in the abstract level is still affecting negatively our results. The reductions obtained in V5 are not enough to outperform search with no abstraction.

5.5.3 Third Experiment

The aim of this experiment is to study the effect of sacrificing optimality for efficiency. Two methods for sacrificing optimality are considered : change of coefficient W of our evaluation function and the selective method. For this experiment we use V5. The abstract spaces were first created using Max Hubs with diameter four and then using diameter two. No orphans were allowed.

Our evaluation function[1] differs from the HPA[Pohl, 1969, 1970] evaluation function. The values of W used for our evaluation function are: 0.0, 0.5, 0.10, 0.20, 0.30, 0.40, 0.50, 0.75, and 1.0. When W= 0.0 the search will depend only on the H value basing its decision on the heuristic information. This strategy will allow us to determine the importance of the heuristic that we are generating. When W=1.00 the search will depend only on the G function and it will behave like Dijkstra's algorithm. W=0.5 is the evaluation function of standard A*.

Another element added in this experiment is the idea of selectivity. This method as described in previous chapter randomly decides to compute h(n) with probability P, where P is a user-specified parameter. The values of P used in this experiment are: 0, 0.25, 0.50, 0.75, 0.90, and 1. When P=0 only those nodes on the first abstract path solution will be considered in the search. Thus P=0 reflects similar behavior to "Classical Refinement" or "Optimal Refinement". When P=1 no node will be discarded during the search.

In this experiment V5 was run with all the different combinations of values for W and P. The complete set of results in given in Appendix C. In order to summarize our results we consider five main combinations of P and W. Those are:

(P= 0, W = 0.0) → Refinement

(P= 0, W = 0.5) → A* Refinement

(P= 0, W = 1.0) → Optimal Refinement

(P= 1, W = 0.0) → Graph Traverser

(P= 1, W = 0.5) → A*

Table 5.7 shows a model of the tables in which the results will be presented (Tables 5.8 through 5.12). Each column corresponds to a different value for W. Each row corresponds to a different value for P. The cell representing the combination (P = 1, W = 1) contains the values for no abstraction. There are three performance parameters in each

[1]Our evaluation function is "F[x] = W G[x] + (1-W) H[x]" which is written in a different way from the HPA evaluation function "F[x] = (1-W) G[x] + W H[x]".

cell: Solution Length, Nodes Expanded, and Total Work. For each of these parameters there are two numbers. The number in parentheses is the value for diameter two. The other number is the value for diameter four.

state-space				
W / P		0.0	0.5	1.0
1.0	Solution Length	D=4 (D=2)	D=4 (D=2)	
	Nodes Expanded	D=4 (D=2)	D=4 (D=2)	
	Total Work	D=4 (D=2)	D=4 (D=2)	
0.0	Solution Length	D=4 (D=2)	D=4 (D=2)	D=4 (D=2)
	Nodes Expanded	D=4 (D=2)	D=4 (D=2)	D=4 (D=2)
	Total Work	D=4 (D=2)	D=4 (D=2)	D=4 (D=2)

Table 5.7 Model of the table for the third experiment

In general, from this experiment we can draw the following conclusions.

The number of alternative paths to the goal increases with the P value. This also gives the possibility to find shorter solution length. In this experiment the solution lengths closer to the optimal solution lengths are obtained when $P \geq .75$. However, in order to evaluate such paths during search, the total amount of work increases.

The total amount of work is also influenced by W. The results show that for bigger values of W the amount of work is bigger and the solution lengths are smaller. This implies that since our heuristics are not completely accurate, a better guidance of the search is achieved by combining the heuristic with G. Using values of W close to 0.5, G and H would have an equivalent effect on the search. However, more informed search results in an increase of the amount of work.

Efficiency could only be improved at the expense of optimality. This is clearly shown for (P = 0). For (P = 0) there is a small and smooth increase of the total amount of work as W increases from 0.0 to 1.0. This increase starts earlier in the Blocks World and the Permutation graphs than in the others. Although for (P = 0) the work seems to be the least, the solution length is not as good as the solution length of the other P's. The same observations can be made for diameter two.

For diameter two the solution lengths for all combinations of P and W, are at most 60% longer than the optimal solution lengths while for diameter four they are at most 44% longer. For combinations of P=0 and any W, the total work is less than for no abstraction and the same is true for most values of (P=1, W=0).

In conclusion, for high values of P and W, the solution length is optimal but the total work is big. There is a trade-off between total work and optimality. Total work can be reduced, while sacrificing optimality, by using lower values of P or/and W.

Blocks World							
W / P		0.0		0.5		1.0	
1.0	Solution Length	10.74	(10.76)	9.92	(9.92)	9.92	
	Nodes Expanded	67	(82)	290	(336)	389	
	Total Work	1944	(3905)	8317	(17294)	2849	
0.0	Solution Length	11.05	(11.96)	10.78	(11.69)	10.6	(10.98)
	Nodes Expanded	63	(27)	82	(45)	119	(84)
	Total Work	1613	(1626)	1692	(1737)	1915	(1991)

Table 5.8 Blocks World - Version V5 for Refinement, A* Refinement, Optimal Refinement, Graph Traverser and No Abstraction

Permutation							
W / P		0.0		0.5		1.0	
1.0	Solution Length	6.7	(6.2)	4.71	(4.71)	4.71	
	Nodes Expanded	64	(28)	186	(176)	285	
	Total Work	1459	(2227)	3375	(9713)	2916	
0.0	Solution Length	6.78	(7.54)	6.74	(7.44)	6.61	(7.01)
	Nodes Expanded	62	(16)	64	(21)	86	(39)
	Total Work	1433	(1214)	1398	(1243)	1566	(1439)

Table 5.9 Permutation - Version V5 for Refinement, A* Refinement, Optimal Refinement, Graph Traverser and No Abstraction

5-puzzle							
W / P		0.0		0.5		1.0	
1.0	Solution Length	21.66	(20.96)	19.29	(19.29)	19.29	
	Nodes Expanded	85	(136)	386	(512)	348	
	Total Work	1712	(3449)	5504	(12714)	2261	
0.0	Solution Length	22.29	(23.62)	21.42	(22.82)	21.09	(21.85)
	Nodes Expanded	48	(37)	91	(64)	119	(93)
	Total Work	1182	(1368)	1403	(1507)	1561	(1666)

Table 5.10 5-puzzle - Version V5 for Refinement, A* Refinement, Optimal Refinement, Graph Traverser and No Abstraction

T.O.H							
W / P		0.0		0.5		1.0	
1.0	Solution Length	71.07	(71.45)	67.35	(67.35)	67.35	
	Nodes Expanded	242	(384)	1571	(2903)	1068	
	Total Work	4927	(11454)	19639	(61212)	7557	
0.0	Solution Length	82.49	(82.02)	78.96	(72.48)	79.15	(71.85)
	Nodes Expanded	176	(136)	364	(274)	398	(328)
	Total Work	3719	(4245)	4814	(5026)	5014	(5342)

Table 5.11 Towers of Hanoi - Version V5 for Refinement, A* Refinement, Optimal Refinement, Graph Traverser and No Abstraction

KL-2000							
W / P		0.0		0.5		1.0	
1.0	Solution Length	11.01	(11.56)	9.87	(9.87)	9.87	
	Nodes Expanded	411	(156)	1058	(997)	1235	
	Total Work	10800	(8344)	21261	(44433)	19956	
0.0	Solution Length	11.6	(12.75)	11.13	(11.95)	10.82	(11.45)
	Nodes Expanded	440	(73)	619	(192)	706	(344)
	Total Work	11190	(4938)	13665	(6844)	14935	(9214)

Table 5.12 KL-2000 - Version V5 for Refinement, A* Refinement, Optimal Refinement, Graph Traverser and No Abstraction

5.5.4 Fourth Experiment

Appendix A shows that the sizes of the abstract spaces for Min Hubs (the hub of each class is the node with minimum degree) are bigger than those of Max Hubs. Therefore, the size of the classes is smaller and fewer nodes in each level will have the same value of h(n). Thus, we expected Min hubs abstraction to produce more informed search than Max hubs.

In this experiment the abstract spaces were created using Min Hubs with diameters two and four. No orphans were allowed.

Our results (see Table 5.13, Table 5.14, Table 5.15) indicate that V5 is still the best search algorithm. It expands fewest nodes and does the least amount of work for all the state-spaces. Compared to V3, the total amount of work done by V5 is between 9% and 38.96% for diameter two and between 0.2% and 25.28% for diameter four. In general, considering all the levels, Max Hubs does up to a maximum of 80.15% less work than Min Hubs. This suggests that the heuristics obtained from Max Hubs are more informed than those from Min Hubs.

	Total Work					Nodes Expanded				
	No Abs.	V3 - Max Hubs		V3- Min Hubs		No Abs.	V3 -Min Base Level		V3 - Min All Levels	
		D = 2	D = 4	D = 2	D = 4		D = 2	D = 4	D = 2	D = 4
Blocks World	2849	24893	8820	52449	14872	389	74	175	526	383
Permutation	2916	12374	3372	27687	5550	285	48	137	332	168
5-puzzle	2261	15110	6252	34389	8389	348	159	262	881	489
T.O.H	7557	72409	22226	114819	25682	1068	599	836	4949	1909
KL-2000	19956	55687	21804	280544	34365	1235	405	834	1630	1083

Table 5.13 Version V3 for A* - Min Hubs

	Total Work					Nodes Expanded				
	No Abs.	V4 - Max Hubs		V4 - Min Hubs		No Abs.	V4 - Min Base Level		V4 - Min All Levels	
		D = 2	D = 4	D = 2	D = 4		D = 2	D = 4	D = 2	D = 4
Blocks World	2849	17858	8478	31976	11811	389	61	158	386	333
Permutation	2916	9985	3380	16958	5543	285	35	137	233	168
5-puzzle	2261	13937	5981	30258	7842	348	155	254	821	470
T.O.H	7557	70765	21885	110929	25098	1068	592	830	4488	1865
KL-2000	19956	47415	21450	184416	31941	1235	341	802	1294	1029

Table 5.14 Version V4 for A* - Min Hubs

	Total Work					Nodes Expanded				
	No Abs.	V5 - Max Hubs		V5 - Min Hubs		No Abs.	V5 - Min Base Level		V5 - Min All Levels	
		D = 2	D = 4	D = 2	D = 4		D = 2	D = 4	D = 2	D = 4
Blocks World	2849	17294	8317	37726	11111	389	64	162	480	316
Permutation	2916	9713	3375	17587	5535	285	37	137	226	167
5-puzzle	2261	12714	5504	30983	7114	348	156	256	850	432
T.O.H	7557	61212	19639	98734	22203	1068	594	832	4337	1683
KL-2000	19956	44433	21261	171240	29953	1235	347	807	1296	1004

Table 5.15 Version V5 for A* - Min Hubs

5.5.5 Fifth Experiment

In order to expand our results to graphs with non-uniform weights, we randomly assign weights between 1 and 10 to the edges of the graphs. The abstract spaces were created using diameter two and four and Max Hubs. No orphans were allowed.

Similar conclusions from those of the second experiment were obtained (see table 5.16, table 5.17, and table 5.18). The results for diameter four are better than those of diameter two. V5 expanded the least total number of nodes and did least total work.

Although the conclusions are similar to those of the second experiment, the results in this experiment for total work and number of nodes expanded are quite a bit larger.

	V3			
	Total Work		Nodes Expanded (All Levels)	
	D = 2	D = 4	D = 2	D = 4
Blocks World	26462	10529	595	404
Permutation	19838	4685	465	292
5-puzzle	16580	6464	717	469
T.O.H	80004	23427	3708	1834
KL-2000	67566	25161	1503	1239

Table 5.16 Version V3 for A* - Graphs with randomly generated weights (1-10)

	V4			
	Total Work		Nodes Expanded (All Levels)	
	D = 2	D = 4	D = 2	D = 4
Blocks World	23558	9989	561	399
Permutation	17802	4678	441	292
5-puzzle	16012	6375	707	466
T.O.H	78517	23267	3655	1826
KL-2000	64575	25083	1476	1238

Table 5.17 Version V4 for A* - Graphs with randomly generated weights (1-10)

	V5			
	Total Work		Nodes Expanded (All Levels)	
	D = 2	D = 4	D = 2	D = 4
Blocks World	21181	9313	485	386
Permutation	16711	4647	413	291
5-puzzle	14357	5726	645	432
T.O.H	67276	20715	3355	1670
KL-2000	56928	24700	1415	1228

Table 5.18 Version V5 for A* - Graphs with randomly generated weights (1-10)

5.6 CONCLUSION

Recall that our aim is to find optimal or near optimal paths doing less work than no abstraction. In the second experiment we evaluated three versions of A*. In all three versions, the solution length was optimal. Search effort decreased compared to our initial implementation of A* using abstraction (Table 3.1). However NAB still performs better.

From all of our five versions, V5 is the best. The improvement of V5 over V3, while noticeable, is not as significant as the difference between V1 and V3. The caching and multiple sources approaches contribute largely to the performance gain using abstraction in A*.

The third experiment showed that P=0 is the most efficient. The results showed that there is a trade-off between optimality and speed-up. To reduce search effort we sacrificed optimality.

Different ways of building abstraction determine the accuracy of the heuristic and directly affects the search effort. For example, Min Hubs abstraction gives bigger results the Max Hub. However, the solution length is not affected. Of all the abstraction methods used Max Hubs showed to be the best.

Chapter 6

Literature review

The A* literature review was reviewed in Chapter 2 and previous work by our research group was reviewed in Chapter 3. This chapter describes previous work on abstraction using A* algorithm and refinement.

6.1 ABSTRACTION AND A*

The idea of creating abstraction and using the length of an abstract solution to guide A* was first introduced by Gaschnig [Gaschnig, 1979]. Afterwards Valtorta [Valtorta, 1984] formally analyzed this idea and Pearl [Pearl, 1984] widely publicized it. The most recent work based on this principle is ABSOLVER [Mostow and Prieditis, 1989].

Our work differs from previous work in many aspects. First, in most of our work we use novel modified versions of A* whereas previous work used classic A*. Our modified versions improve the efficiency of classic A*. Second, our abstraction are obtained by a process called reduced model. This process groups objects and operators which are indistinguishable at the abstract levels. Previous systems use relaxed model to create their abstract problem. In relaxed models preconditions of operators are dropped to generate an abstract problem. This abstract problem has the same number of states as the original problem but with more transitions between states. More transitions might increase the search effort needed to find the heuristics. Finally, our abstraction method creates an abstraction hierarchy of reduce models which is a collection of abstract spaces. To build an abstraction hierarchy the abstraction method is initially applied to the original problem to create a first abstract problem. A second abstract problem is then created by applying the abstract method to the first abstract problem, and so on. Abstract problems are created until we reach one containing a single state. Most of the previous approaches generate just one abstract problem but all used relaxation hierarchies.

Valtorta's main results shows that if the abstract space is created using a relaxed model, then the computation of A* cannot not outperform Dijkstra's algorithm because the computation of a heuristic function by a secondary search exceeds the savings offered by the heuristic. Valtorta measures performance in terms of "number of node expansions". He considers the cost of expanding a node in an auxiliary problem to be the same as the cost of expanding a node in the original problem. [Hansson, Mayer, and Valtorta, 1992] extend Valtortas's initial work. This extension was motivated to determine how best to use a set of available heuristics. This work showed that in certain situations their is no advantage to using a relaxation hierarchy with A* algorithm. It is better to ignore the hierarchy and using the heuristics directly. They concluded from their theorems that a heuristic is effective in reducing search effort only when it is computable by an efficient algorithm.

As we have seen in chapter 5, our A* algorithm does fewer nodes expansions with abstraction than without it for some graphs. If we had been using relaxed models this would not be possible.

ABSOLVER [Mostow and Prieditis, 1989] generates abstractions by dropping information from the STRIPS-like representation of a problem class. In order to do so, it uses a catalog of abstraction transformations. The optimal solution of the abstract problem is used as an estimate of the distance to the goal. This heuristic is used to guide the search in the base level. ABSOLVER is an explanatory model because it rationally reconstructs some existing heuristics and it is a generative model because it suggests new heuristics. A disadvantage is that the heuristic generated depends on the initial representation of the problem and the abstraction transformation. Many abstraction algorithm are sensitive to initial problem formulation. That is the case of ABSOLVER where many representations satisfying a predicate are possible. The abstraction method of our study overcomes this problem by representing the original problem as a graph, where each node has an associated state. Thus, different initial problem representations will have the same graphical representation.

6.2 ABSTRACTION AND REFINEMENT

The problem reduction or means-ends analysis is a methodology that describes the problem in terms of several components reducing the complexity of a global problem. These components represent subproblems or subgoals rather than just a state description. The importance of this methodology is to solve a sequence of subproblems in order from left to right. The mean-end analysis was first used in GPS [Newell and Simon, 1972]. The problem consist in taking differences between the current subgoal and the next subgoal until there are no differences which means that the current subgoal is identical to the desired situation. Another problem solver motivated by GPS is STRIPS [Fikes and Nilsson, 1971] which is based on the First Order Logic. In STRIPS the decisions of which action need to be taken to reduce the differences between the current subgoal and the next subgoal are made mechanically on the basis of operators. Operators are defined using three lists of predicates. One of these list is the precondition list containing the predicates that must be true before applying the operator. The add list contains all the predicates added to the description of the world-state, and the delete list contains all the predicates that are not true and deleted from the state description.

In GPS and in STRIPS the abstraction of the problem had to be given by the user. The mean-end strategy can also be applied creating different levels of subproblems, that is the case of ABSTRIPS [Sarcerdoti, 1974]. These was the first system automated the construction of abstraction. The idea in ABSTRIPS in order to build the abstraction hierarchy was to rank the conditions according to their importance. Having different levels ABSTRIPS is able to verify the correctness of the solution to the most detailed level by adding details to the solution progressively in each level. This is done by first mapping the problem instance to be solved into the most abstract level, solve in that space the problem and then add details the abstract plan until the original problem is solved. When a solution cannot be found, the system backtracks to find a different abstract solution.

ALPINE [Knoblock, 1991] is based on ABSTRIPS but produces more effective abstractions and requires less knowledge: ALPINE differs from ABSTRIPS in that

abstractions are obtained from reduced models. Given the operator definition for a given domain, ALPINE construct a partially ordered graph of the literals. ALPINE abstraction hierarchies guarantees that every possible refinement of an abstract plan leaves the conditions established in the abstract plan unchanged. This property is known as " the ordered monotonicity property". This property guarantees that once a goal is satisfied at one level, it will not be violated while refining a plan at a lower level. However if the plan cannot be refined, it may be necessary to backtrack to the more abstract level. "ALPINE is sensitive to the representation of operators and this could limit the granularity of the abstractions" [Knoblock, 1991].

A problem solver can expand one part of the plan to a given level and then work on a different part of the plan (not necessarily in a sequence manner). Thus, there is no requirement to expand completely a plan at one level of abstraction before refining it to the next level. Instead, there are abstractions for each operator, and each instance of an operator in the abstract plan can be expanded to a different level of detail. This approach has some drawbacks. A problem solver may expand one part of the plan down to a given level and then work on a different part of the plan that then undoes conditions that were needed in the part of the problem already solved. This could cause correct plans at one level can be expanded into incorrect plans at lower levels. This is the case of least-commitment problem solvers, NOAH [Sacerdoti, 1977], SIPE [Wilkins, 1984, Wilkins, 1986], ABTWEAK [Yang, 1990]. In our study, we focus our attention to those systems which expand the subproblems following the sequence establish by the abstract solution, expanding them at the same level of details.

6.3 CONCRETIZATION

Hierarchies generated by concretization transformations [Prieditis and Janakiraman, 1992] are created by adding constraints to a problem. The addition of constraints reduces the branching factor during search. Although the generation of solutions is more efficient, generally the solutions are longer. This strategy when searching is to build a concrete space which is a subspace of the original. To solve a

problem using concretization, first a path has to be found from i (initial state) to the nearest concrete state i' by searching the graph for the original problem. Second, a path has to be found from g (goal state) to the nearest concrete state g' by searching in the original problem. Once we have these two paths, a third path has to be found in the concrete space from i' to g' . Finally the three paths are concatenated to form a sequence:

i - i' - g' - g

This process is repeated recursively in a hierarchy of concrete spaces. Any solution in the concrete space is guarantee to be a solution in the original space. However a solvable problem in the original space may have no solution in the concrete space. One problem of this strategy is that the construction of hierarchies is generally a difficult problem. However, a catalog of problem transformations might prove helpful.

Chapter 7

Conclusion

7.1 SUMMARY OF THE THESIS

The goal of this thesis is to show the usefulness of abstraction in heuristic search. In order to do so, we focused our study on three main points.

First, we tried to speed-up search without sacrificing optimality of the paths. For this we presented several approaches: reuse of heuristic information, the generation of more information from multiple sources, the calculation of "expensive" heuristics when they are strictly necessary, the alternation direction technique "First Time Alternation".

The multiple sources presented were: abstract solution used as heuristics, calculating the minimum edge weight emanating from a node, P-G to generate heuristics from the length of the solution path for those node visited during search. The technique for deferring heuristic calculation was the Lazy Evaluation. The use of multiple sources proved to be fundamental in the speed-up of search. The combination of multiple sources with lazy evaluation and first time alternation technique resulted in the best combinations of our approaches to achieve greater speed-up. Although optimality was preserved, the speed-up was not enough to outperform search without abstraction.

Second, we tried to increase the search speed even more by given up optimality of the paths. To do so, we presented the Selective Method. This method was used in combination with a previous approach based on changing the coefficient W in the evaluation function "F[x] = W G[x] + (1-W) H[x]". Experimental results indicate a big reduction on search for paths far from optimal and a small reduction on search for paths near optimal.

Finally, this thesis presented two experiments to determine which abstraction techniques work well. For all search techniques the max hub star abstraction technique

was the most flexible and consistent. This was later confirmed in an experiment described in Chapter 5

7.2 LIMITATIONS AND WEAKNESSES

Generally the state-space graphs of puzzles are represented implicitly. In this thesis, our graphs are explicitly represented. This representation has a practical value in order to simplify our abstraction creation techniques. Although this representation simplifies our work, it also limits our research with small graphs. Another limitation in our study is the use of few problems which are good testing ground for our approaches. On one hand, the small number and great diversity of the test graphs used prevent us from drawing valid statistical conclusions. On the other hand, only two of our test graphs represent real applications. More real applications need to be tested in order to extend our conclusions to a bigger scope of problems.

7.3 FUTURE WORK

There are many open and unstudied directions in which to continue this work.

- From observations made in Chapter 5, on our fourth experiment, when we use Min Hubs to create abstract spaces, the work at the base level is less than for Max hubs. This indicates that better heuristics could be generated from different abstraction methods. It might be possible to find an abstraction method that has the same effect in all the levels of the abstraction hierarchy.
- However, a problem that we faced is that the speed-up was not enough to outperform search without abstraction. The reason for this could be that the method to create abstractions is not appropriate for the generation of heuristic information.
- Recall also from Chapter 5 that in our fifth experiment, we ran some versions by randomly assigning weights, between 1 and 10, to the edges. However, the abstraction method was not designed to handle weighted graphs. Our current abstraction handles mainly uniform weighted graphs. Appropriate abstractions need to be studied.

- Similar to Perimeter Search, the implementation of Breath-First Search in First Time Alternation algorithm to generate a perimeter, still needs to be investigated. This could generate more heuristic information.
- In refinement techniques, we cannot in general bound how poor the solution length might be compared to the optimal. The abstract solution found at one level might not define the scope of the search in the next level down in which the shortest solution for that level could be found. Perhaps a theoretical analysis could discover a bound, based on a parameter easily measured, for special classes of graphs.

REFERENCES

Amarel, S. (1983), "Representation in Problem Solving". In *Methods of Heuristics.* Lawrence Erlbaum and Associates, Palo Alto, CA.

Berlekamp, E., J. Conway, and R. Guy (1982), *"Winning Ways for Your Mathematical Plays: Volume II"*. Academic Press, London.

Dechter, R. and J. Pearl (1983), "The Optimality of A* Revisited". *AAAI*, pp. 95 - 99.

Dijkstra, E. (1959), "A Note on Two Problems in Connexion With Graphs". *Numerische Mathematik* 1, pp. 269 - 271.

Dillenburg, J. and P. Nelson (1994), "Perimeter Search". *Artificial Intelligence* 65, pp. 165 - 178.

Doran, J. and D. Michie (1966), "Experiments With the Graph Traverser Program". *Proceedings of the Royal Society of London* 294(A):235 - 59.

Fikes, R. and N. Nilsson (1971), "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". *Artificial Intelligence* 2: 189 - 208.

Gaschnig, J. (1979). "A Problem Similarity Approach to Devising Heuristics: First Results". *Proc. IJCAI 6*, Tokyo, pp. 301-7.

Giunchinglia, F. and T. Walsh (1992), "A Theory of Abstraction". *Artificial Intelligence,* 57, pp. 323 - 389.

Hart, P., N. Nilsson, and B. Rapkael (1968), "A formal basis for the heuristic determination of minimum cost paths". *IEEE Trans. Systems Science and Cybernetics,* SSC-4, No. 2, pp. 100 - 107.

Hansson, O., A. Mayer ,and M. Valtorta (1992), "A New Result on the Complexity of Heuristic Estimates for the A* Algorithm. *Artificial Intelligence* 55, pp. 129 - 143.

Hansson, O., A Mayer and M. Yung (1992), "Criticizing Solutions to Relaxed Models Yields Powerful Admissible Heuristics. *Information Science* 63, pp. 207 - 227.

Held, M. and R. Karp (1971), "The Traveling Salesman Problem and Minimum Spanning Trees-Part II". *Mathematical Prog.* 1:6 - 25

Hitz, M., T. Mueck (1994), "Routing Heuristics for Carley Graph Topologies". *Proceedings of the 10th Conference on AI and Applications* (CAIA'94), pp. 474-476.

Holte, R., T. Mkadmi, R. Zimmer, A. MacDonald (1995), "Speeding-Up Problem-Solving by Abstraction: A Graph-Oriented Approach". *Technical report* TR - 95 - 07.

Holte , R., C. Drummond, M.B. Perez, R. Zimmer, and A. MacDonald (1994), "Searching With Abstractions: A Unifying Framework an New High-Performance Algorithm", *Proceedings 10th Canadian Conference on Artificial Intelligence.* pp. 263 - 270.

Kavraki, L. and J. Latombe (1994), "Randomized Preprocessing of Configuration Space for Fast Path Planning", *Proceedings of the IEEE International Conference on Robotics and Automation".*

Kavraki, L. and J. Latombe (1993), "Randomized Preprocessing of Configuration Space for Fast Path Planning", *tech. report* STAN-CS-93-1490, Computer Science Dept., Stanford University.

Knoblock, C. (1993), *"Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning".*

Knoblock, C. (1991), "Automatically Generating Abstractions for Problem Solving". *Tecnical report CMU-CS-91-120,* Computer Science Department, Canergie-Lellon University.

Korf, R. (1985), "Depth-first Iterative-deepening an Optimal Admissible Tree-search". *Artificial Intelligence* volume 27 number 1, pp. 97-109.

Korf, R. (1990), "Real-time Heuristic Search". *Artificial Intelligence* 42, pp. 189 - 211.

Kwa, J. (1989), "BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm". *Artificial Intelligence* 38, pp. 95 - 109.

Minton, S., J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni, and Y. Gil (1989), "Explanation-Based Learning: A Problem Solving Perspective". *Artificial Intelligence,* 40(1 - 3): 63 - 118.

Mkadmi, T. (1993), "Speeding-Up State-Space Search by Automatic Abstraction". *Master's thesis.* Computer Science Dept., University of Ottawa.

Mostow, J. and A. Prieditis (1993), "Discovering Admissible Heuristics by Abstracting and Optimizing: A Transformational Approach". *Machine Learning* vol. 12 number 1 - 3.

Newell, A., H. Simon (1972). *"Human Problem Solving"*. Prentice-Hall. Englewood Cliffs,NJ.

Pearl, J. (1984), *"Heuristics: Intelligent Search Strategies for Computer Problem Solving"*. Addison - Wesley.

Pohl, I. (1969), "Bi-Directional and Heuristic Search in Path Problems". *SLAC* report No. 104.

Prieditis, A. and R. Davis (1994), "Quantitatively Relating Abstractness to the Accuracy of Admissible Heuristics". *Submitted to the Journal of Artificial Intelligence as a Research Note*

Prieditis, A. (1993), "Machine Discovery of Effective Admissible Heuristics". *Machine Learning* 12, pp. 117 - 141.

Prieditis, A. and B. Janakiraman (1993), " Generating Effective Admissible Heuristics by Abstraction and Reconstitution", *Proc. AAAI*, pp. 743 - 748.

Prieditis, A. and B. Janakiraman (1992), "Becoming Reactive by Concretization", *Proceedings of the 1992 Asilomar workshop on Change of Representation and Problem Reformulation.*

Sacerdoti, E. (1977), *"A Structure for Plans and Behavior"*. American Elsevier. New York, NY.

Sacerdoti, E. (1974), "Planning in a Hierarchy of Abstraction Spaces". *Artificial Intelligence* 5(2): 115 - 135.

Slate, D. and L. Atkin (1977), "Chess 4.5 -the Northwestern University chess program". *Chess Skill in Man and Machine*, ed. P. W. Frey. New York: Springer-Verlag.

Valtorta, M. (1984), "A Result on the Computational Complexity of Heuristic Estimates for the A* Algorithm". *Inform. Sci.* 34, pp. 47 - 59.

Wilkins, D. (1986), "High - Level Planning in a Mobile Robot Domain". *Technical report* 388. Artificial Intelligence Center, SRI International.

Wilkins, D. (1984), "Domain - Independent Planning: Representation and Plan Generation". *Artificial Intelligence*, 22(3): 269 - 301.

Yang, Q. and J. Tenenberg (1990), "ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner". *Proceedings AAAI-90.* Boston,MA. 204-209.

Appendix A

State-spaces and Abstract-Spaces

A.1 BLOCKS WORLD

The are N number of blocks on a table. A block can be stacked on top of another block, and there is a "robot hand" capable of : putting the block being held onto the table, putting the block being held on top of a specific stack of blocks, picking up a block from the table, and picking up the block on top of a specific stack. An example of the block world with two blocks is given in Chapter 2.

We use 5blocks. The abstract spaces of the hierarchies were created with the star method using minimum hubs and maximum hubs for this puzzle are given in the next two subsections.

A.1.1 Maximum Hubs

				Branching Factor				
Number of States		Number of Edges		Maximum		Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
866		2090		5		2.41		Base Level
254	119	840	460	24	64	3.30	3.86	1
45	1	204	0	42	0	4.53	0	2
1	-	0	-	0	-	0	-	3

A.1.2 Minimum Hubs

				Branching Factor				
Number of States		Number of Edges		Maximum		Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
866		2090		5		2.41		Base Level
365	165	1088	624	10	22	2.98	3.78	1
174	12	706	22	15	11	4.05	1.83	2
75	1	464	0	12	0	6.18	0	3
29	-	276	-	17	-	9.51	-	4
8	-	56	-	7	-	7	-	5
1	-	0	-	0	-	0	-	6

A.2 PERMUTATION

A state is a permutation of the integers 1,...,N. There are N-1 operators. Each operator reverses the order of the first X integer of the state. X is define by the operator and it takes values between 2 to N.

e.g For operator 3 (X = 3) and state [**1,2,3**,4,5,6]

the resulting state is [**3,2,1**,4,5,6]

We use N = 6. The abstract spaces of the hierarchies were created with the star method using minimum hubs and maximum hubs for this puzzle are given in the next two subsections.

A.2.1 Maximum Hubs

				Branching Factor				
Number of States		Number of Edges		Maximum		Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
720		3600		5		5		Base Level
137	36	2366	414	26	33	17.3	11.5	1
10	1	90	0	9	0	9	0	2
1	-	0	-	0	-	0	-	3

A.2.2 Minimum Hubs

				Branching Factor				
Number of States		Number of Edges		Maximum		Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
720		3600		5		5		Base Level
254	68	2668	1344	20	63	10.5	19.8	1
68	1	1696	0	47	0	24.9	0	2
10	-	88	-	9	-	8.8	-	3
1	-	0	-	0	-	0	-	4

A.3 5-PUZZLE

This puzzle consist of 5 numbered movable tiles set in a 2×3 frame. One cell of the frame is always empty, making it possible to move an adjacent numbered tile to the empty cell, moving, thus, the empty cell.

1	2	3
4	5	empty

For the 5-puzzle the two unconnected regions (each containing 360 states) has been connected by an edge between one randomly chosen state in each of the regions.

The abstract spaces of the hierarchies were created with the star method using minimum hubs and maximum hubs for the puzzle are given in the next two subsections.

A.3.1 Maximum Hubs

Number of States		Number of Edges		Branching Factor: Maximum		Branching Factor: Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
720		1682		3		2.33		Base Level
194	88	630	364	5	10	3.24	4.13	1
39	3	192	4	11	2	4.92	1.33	2
7	1	22	0	6	0	3.14	0	3
1	-	0	-	0	-	0	-	4

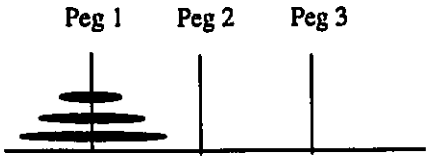
A.3.2 Minimum Hubs

Number of States		Number of Edges		Branching Factor: Maximum		Branching Factor: Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
720		1682		3		2.33		Base Level
337	119	916	478	4	8	2.71	4.01	1
152	12	544	60	6	10	3.57	5	2
59	1	300	0	8	0	5.08	0	3
20	-	136	-	9	-	6.8	-	4
4	-	6	-	2	-	1.5	-	5
2	-	2	-	1	-	1	-	6
1	-	0	-	0	-	0	-	7

A.4 TOWERS OF HANOI

The are three pegs and N different disks of differing sizes. The disks can be stacked on the pegs. Only the top disk on a peg can be moved, but it can never be placed on top of a smaller disk.

e.g. 3-disks Towers of Hanoi



We use N = 7. The abstract spaces of the hierarchies were created with the star method using minimum hubs and maximum hubs for this puzzle are given in the next two subsections.

A.4.1 Maximum Hubs

Number of States		Number of Edges		Branching Factor: Maximum		Branching Factor: Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
2187		6558		3		2.99		Base Level
539	219	1694	680	4	4	3.14	3.10	1
127	22	368	60	4	4	2.89	2.72	2
30	3	90	6	5	2	3	2	3
8	1	20	0	3	0	2.5	0	4
2	-	2	-	1	-	1	-	5
1	-	0	-	0	-	0	-	6

A.4.2 Minimum Hubs

Number of States		Number of Edges		Branching Factor: Maximum		Branching Factor: Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
2187		6558		3		2.99		Base Level
729	243	2184	726	3	3	2.99	2.98	1
243	27	726	78	3	3	2.98	2.88	2
81	3	240	6	3	2	2.96	2	3
27	1	78	0	3	0	2.88	0	4
9	-	24	-	3	-	2.66	-	5
3	-	6	-	2	-	2	-	6
1	-	0	-	0	-	-	-	7

A.5 KL-2000

This is the graph "connect2000_1000.res" provided to us by Lydia Kavraki and J-C. Latombe, of Stanford University. It is produced by their algorithm for discretizing the continuous space of states/motions of robots with many degrees of freedom [Kavraki and Latombe, 1993, 1994].

The abstract spaces of the hierarchies were created with the star method using minimum hubs and maximum hubs for this puzzle are given in the next two subsections.

A.5.1 Maximum Hubs

Number of States		Number of Edges		Branching Factor: Maximum		Branching Factor: Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
2736		28824		66		10.53		Base Level
330	91	2278	328	39	30	6.90	3.60	1
35	2	132	2	16	1	3.77	1	2
5	1	8	0	4	0	1.6	0	3
1	-	0	-	0	-	0	-	4

A.5.2 Minimum Hubs

Number of States		Number of Edges		Branching Factor: Maximum		Branching Factor: Average		Level of Abstraction
Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		Diameter 2 - 4		
2736		28824		66		10.53		Base Level
908	205	11534	1512	57	37	12.7	7.37	1
265	11	3556	32	44	8	13.4	2.90	2
74	1	992	0	32	0	13.4	0	3
17	-	116	-	12	-	6.82	-	4
4	-	8	-	3	-	2	-	5
2	-	2	-	1	-	1	-	6
1	-	0	-	0	-	0	-	7

Appendix B

Experimental Results for refinement

B.1 MAXIMUM HUBS

B.1.1 Path Length

The number in brackets in the following table, represent the solution length for breadth first search without abstraction.

		Diameter					
		2	3	4	5	6	7
T.O.H (66)	CR	98	101	80	82	82	79
	OR	88	77	72	72	72	69
	AO	80	76	75	76	77	75
5-puzzle (21)	CR	29	27	24	29	27	26
	OR	27	24	23	25	26	25
	AO	25	24	23	25	25	24
Blocks World (9.5)	CR	14.7	11.5	12.1	11.9	11.6	10.3
	OR	13.2	11.0	11.6	11.2	10.5	9.7
	AO	11.2	10.8	11.2	11.3	10.8	10.2
Permutation (6.2)	CR	11.6	9.5	9.7	8.1	7.1	7.3
	OR	10.6	9.3	8.9	7.3	6.4	6.4
	AO	8.3	8.0	8.1	7.4	6.8	6.8

B.1.2 Total Work (#edges + overhead)

The number in brackets in the following table, represent the total work for breadth first search without abstraction.

		Diameter					
		2	3	4	5	6	7
T.O.H (3058)	CR	894	867	776	752	777	802
	OR	1048	903	903	927	944	950
	AO	767	711	751	740	765	828
5-puzzle (802)	CR	299	251	294	314	335	333
	OR	362	308	333	371	392	392
	AO	255	238	301	300	321	325
Blocks World (3936)	CR	724	1200	902	1268	2124	3424
	OR	1073	1339	1127	1767	2906	4258
	AO	762	1227	889	1395	2146	4149
Permutation (5570)	CR	512	616	750	1407	3055	5739
	OR	868	762	1191	2443	3926	7053
	AO	460	609	790	2527	5454	6416

B.2 RANDOM HUBS

B.2.1 Path Length

The number in brackets in the following table, represent the solution length for breadth first search without abstraction.

		Diameter					
		2	3	4	5	6	7
T.O.H (66)	CR	91	95	82	86	95	88
	OR	80	81	72	74	86	81
	AO	75	83	77	78	79	79
5-puzzle (21)	CR	32	30	27	28	26	25
	OR	29	29	25	25	26	25
	AO	26	26	25	24	24	25
Blocks World (9.5)	CR	27.2	22.0	19.7	14.3	15.6	17.0
	OR	21.3	19.1	16.5	13.6	14.8	16.2
	AO	12.7	13.1	13.6	12.1	13.0	13.8
Permutation (6.2)	CR	15.6	11.4	10.1	9.5	9.0	6.9
	OR	12.0	10.0	9.6	8.5	7.7	6.4
	AO	8.4	8.1	8.5	8.6	8.0	6.6

B.2.2 Total Work (#edges + overhead)

The number in brackets in the following table, represent the total work for breadth first search without abstraction.

		Diameter					
		2	3	4	5	6	7
T.O.H (66)	CR	847	794	734	776	931	942
	OR	1021	957	870	924	1111	1106
	AO	765	763	738	763	830	916
5-puzzle (21)	CR	316	282	269	280	306	292
	OR	411	338	337	360	363	354
	AO	285	264	264	265	303	302
Blocks World (9.5)	CR	1310	571	699	902	808	964
	OR	1526	783	1027	1093	1048	1297
	AO	1070	546	738	896	780	964
Permutation (6.2)	CR	806	800	811	1258	3142	6295
	OR	1015	978	1250	2491	5064	7008
	AO	817	806	1038	1456	3674	6566

Appendix C

Experimental Results for A*

C.1 FIRST EXPERIMENT

	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
	Blocks World							
V1	9.92	3333.37	20075.27	23113.87	4401.82	10277.08	50924.33	56799.59
V2	9.92	2217.74	11128.24	11877.96	10524.25	4819.61	35748.19	30043.55
	Permutation							
V1	4.71	781.52	13215.59	11029.75	1649.74	5992.68	26676.60	31019.545
V2	4.71	1545.96	25848.94	6238.12	8208.49	2848.23	41841.51	36481.25
	5-puzzle							
V1	19.29	3671.22	12550.09	11900.62	2549.80	5358.91	30671.72	33480.83
V2	19.29	1706.89	5749.21	6080.42	6034.62	2110.03	19571.13	15646.53

C.2 SECOND EXPERIMENT

		Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
		Blocks World							
V3	D = 2	9.92	431.09	1676.61	22246.30	5712.42	539.70	30066.41	24893.70
	D = 4	9.92	302.41	1060.21	7107.51	6610.25	349.94	15080.37	8820.05
V4	D = 2	9.92	355.36	1438.11	15627.66	7001.70	437.25	24422.82	17858.38
	D = 4	9.92	294.88	1037.32	6806.56	8046.37	340.01	16185.12	8478.76
V5	D = 2	9.92	336.14	1308.17	15230.85	6632.41	419.04	23507.57	17294.19
	D = 4	9.92	290.63	984.72	6707.02	7282.19	335.05	15264.55	8317.41
		Permutation							
V3	D = 2	4.71	215.69	2797.68	9089.72	1533.23	271.23	13636.32	12374.31
	D = 4	4.71	186.22	1084.63	1898.82	3080.32	203.11	6249.98	3372.77
V4	D = 2	4.71	186.49	2372.58	7195.35	6891.12	230.62	16645.54	9985.04
	D = 4	4.71	186.53	1086.37	1903.70	6138.41	203.52	9315.01	3380.12
V5	D = 2	4.71	176.42	2178.90	7136.11	6125.99	222.19	15617.41	9713.61
	D = 4	4.71	186.02	1072.78	1913.11	5595.00	203.42	8766.91	3375.33
		5-puzzle							
V3	D = 2	19.29	578.49	1743.81	12133.30	2752.30	654.64	17207.89	15110.23
	D = 4	19.29	418.23	1216.91	4165.99	2130.46	451.79	7931.58	6252.91
V4	D = 2	19.29	554.43	1667.26	11091.30	4487.92	624.63	17800.90	13937.61
	D = 4	19.29	407.13	1181.24	3954.16	3842.19	438.52	9384.71	5981.04
V5	D = 2	19.29	512.06	1535.63	10084.77	4051.31	582.25	16183.77	12714.71
	D = 4	19.29	386.43	1090.63	3613.16	3217.15	414.47	8307.36	5504.68
		T.O.H							
V3	D = 2	67.35	3206.90	9867.33	55987.28	12541.42	3347.82	81602.92	72409.32
	D = 4	67.35	1741.77	5300.57	13389.27	8154.72	1794.42	28586.32	22226.02
V4	D = 2	67.35	3141.22	9663.18	54681.77	22584.56	3279.23	90070.71	70765.39
	D = 4	67.35	1721.46	5238.72	13151.84	16818.63	1773.04	36930.64	21885.05
V5	D = 2	67.35	2903.22	8900.06	46371.47	20568.16	3037.41	78742.89	61212.14
	D = 4	67.35	1571.39	4770.69	11677.81	14397.64	1619.32	32417.52	19639.20
		KL-2000							
V3	D = 2	9.87	1119.14	12244.75	41077.77	17335.98	1246.20	71777.63	55687.85
	D = 4	9.87	1078.33	12605.92	7009.23	34574.84	1110.69	55268.32	21804.17
V4	D = 2	9.87	1020.53	11353.24	33916.68	23604.17	1125.18	69894.61	47415.62
	D = 4	9.87	1062.77	12447.71	6846.96	35876.89	1093.54	56234.31	21450.97
V5	D = 2	9.87	997.54	11160.70	31178.35	20707.04	1097.28	64043.62	44433.86
	D = 4	9.87	1058.51	12441.73	6673.98	32242.62	1087.35	52416.83	21261.56

C.3 THIRD EXPERIMENT

	Blocks World - Diameter 2							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	11.96	27.16	131.59	1438.90	109.59	29.04	1707.23	1626.68
W=.05 ,P=0.0	11.96	27.19	131.64	1433.99	108.91	29.07	1701.73	1621.89
W=.10 ,P=0.0	11.96	27.19	131.64	1433.99	108.91	29.07	1701.73	1621.89
W=.20 ,P=0.0	11.83	28.84	137.87	1435.07	108.95	30.73	1710.72	1632.50
W=.30 ,P=0.0	11.73	33.59	153.33	1439.21	109.51	35.48	1735.63	1661.60
W=.40 ,P=0.0	11.69	39.40	174.16	1444.63	111.55	41.29	1769.74	1699.47
W=.50 ,P=0.0	11.69	45.17	193.44	1451.58	113.05	47.05	1803.23	1737.23
W=.75 ,P=0.0	10.99	78.38	318.30	1477.82	152.38	80.26	2026.88	1954.76
W=1 ,P=0.0	10.98	84.29	334.98	1485.84	152.56	86.17	2057.66	1991.28
W=.0 ,P=0.25	11.29	54.41	278.94	2234.91	857.36	63.21	3425.62	2631.47
W=.05 ,P=0.25	11.24	70.11	357.65	2668.46	1457.63	84.45	4553.84	3180.66
W=.10 ,P=0.25	11.09	64.53	332.94	2609.91	1344.12	78.07	4351.50	3085.45
W=.20 ,P=0.25	11.17	77.41	394.32	2957.71	1631.61	93.49	5061.03	3522.91
W=.30 ,P=0.25	10.91	121.45	570.71	3790.86	2285.42	143.27	6768.43	4626.28
W=.40 ,P=0.25	10.71	146.06	672.16	4814.27	2728.40	173.13	8360.87	5805.61
W=.50 ,P=0.25	10.64	139.50	631.64	5198.21	2509.29	166.94	8478.64	6136.29
W=.75 ,P=0.25	10.62	202.07	821.32	6929.80	2950.53	237.15	10903.71	8190.33
W=1 ,P=0.25	10.63	195.59	761.52	7366.77	2641.32	230.57	10965.19	8554.45
W=.0 ,P=0.5	11.10	67.82	336.52	2624.79	1346.98	80.81	4376.09	3109.92
W=.05 ,P=0.5	10.89	94.61	463.46	3545.42	2472.50	118.49	6575.98	4221.97
W=.10 ,P=0.5	10.95	95.16	458.32	3487.57	2413.72	118.33	6454.77	4159.38
W=.20 ,P=0.5	10.87	101.64	495.67	3903.28	2654.79	127.60	7155.37	4628.17
W=.30 ,P=0.5	10.30	163.25	733.72	5268.30	3626.41	198.08	9791.67	6363.34
W=.40 ,P=0.5	10.25	241.04	1025.74	8225.00	4830.81	291.01	14322.57	9782.78
W=.50 ,P=0.5	10.34	227.32	952.15	9186.67	4518.10	278.98	14884.23	10645.11
W=.75 ,P=0.5	10.25	333.17	1260.22	13735.75	5458.25	403.98	20787.39	15733.12
W=1 ,P=0.5	10.37	328.56	1193.72	15430.87	5170.01	402.30	22123.15	17355.44
W=.0 ,P=0.75	10.91	79.14	382.00	3117.20	1853.64	96.99	5431.96	3675.31
W=.05 ,P=0.75	10.71	112.73	529.07	4119.75	3176.12	143.12	7937.66	4904.66
W=.10 ,P=0.75	10.74	109.20	518.66	4135.16	3154.48	139.59	7917.49	4902.60
W=.20 ,P=0.75	10.69	132.26	608.81	4799.33	3586.02	167.16	9126.41	5707.54
W=.30 ,P=0.75	10.18	204.23	872.16	6738.88	4568.09	249.35	12383.35	8064.61
W=.40 ,P=0.75	10.06	299.19	1229.71	11025.56	5997.04	364.36	18551.49	12918.81
W=.50 ,P=0.75	10.09	292.28	1169.73	12698.85	5764.15	362.20	19925.00	14523.05
W=.75 ,P=0.75	10.08	453.65	1620.50	21029.09	7435.20	558.47	30538.44	23661.71
W=1 ,P=0.75	10.06	458.43	1565.80	23760.60	7456.54	570.49	33241.36	26355.31
W=.0 ,P=0.9	10.81	79.47	382.26	3204.70	1915.96	97.92	5582.38	3764.34
W=.05 ,P=0.9	10.75	123.69	567.85	4523.77	3517.24	157.82	8732.55	5373.13
W=.10 ,P=0.9	10.75	124.28	568.78	4498.37	3519.92	158.32	8711.34	5349.73
W=.20 ,P=0.9	10.57	130.78	604.59	4909.05	3770.69	167.40	9415.11	5811.82
W=.30 ,P=0.9	10.14	218.24	924.80	7297.71	5008.28	267.90	13449.03	8708.65
W=.40 ,P=0.9	10.03	333.11	1338.45	12585.57	6572.18	406.13	20829.30	14663.25

W=.50	,P=0.9	10.01	322.68	1266.21	14294.81	6371.66	401.09	22255.35	16284.78
W=.75	,P=0.9	9.96	510.62	1779.35	24292.36	8209.28	630.06	34791.60	27212.38
W=1	,P=0.9	9.96	526.23	1746.61	28200.21	8611.33	658.05	39084.37	31131.09
W=.0	,P=1.0	10.76	82.27	393.67	3327.91	2056.04	102.09	5859.88	3905.93
W=.05	,P=1.0	10.70	135.43	602.22	4650.10	3726.37	171.18	9114.12	5558.93
W=.10	,P=1.0	10.70	134.60	600.31	4645.99	3721.27	170.30	9102.16	5551.19
W=.20	,P=1.0	10.55	140.95	640.83	5111.38	3995.96	179.58	9889.11	6072.73
W=.30	,P=1.0	10.08	228.45	958.83	7707.21	5235.10	280.73	14129.59	9175.22
W=.40	,P=1.0	9.92	340.76	1365.26	13097.37	6779.04	416.65	21582.42	15220.03
W=.50	,P=1.0	9.92	336.14	1308.17	15230.85	6632.41	419.04	23507.57	17294.19
W=.75	,P=1.0	9.92	543.50	1861.64	26405.26	8644.06	672.09	37454.45	29482.48
W=1	,P=1.0	9.92	563.45	1836.80	30576.38	9162.55	705.57	42139.18	33682.20

	Blocks World - Diameter 4							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	11.05	63.82	253.65	1230.99	185.37	65.32	1733.83	1613.78
W=.05 ,P=0.0	11.05	63.82	253.65	1197.85	185.37	65.32	1700.69	1580.64
W=.10 ,P=0.0	11.01	64.63	256.15	1198.56	185.37	66.13	1704.71	1585.47
W=.20 ,P=0.0	10.84	71.52	275.78	1204.61	185.37	73.02	1737.28	1624.92
W=.30 ,P=0.0	10.78	74.62	285.04	1207.13	185.37	76.12	1752.16	1642.90
W=.40 ,P=0.0	10.78	78.62	297.42	1211.69	185.37	80.11	1773.10	1667.84
W=.50 ,P=0.0	10.78	82.77	308.71	1217.17	185.37	84.27	1794.01	1692.91
W=.75 ,P=0.0	10.60	118.69	427.55	1245.47	240.49	120.18	2032.19	1911.88
W=1 ,P=0.0	10.60	119.36	428.96	1245.98	240.49	120.85	2034.78	1915.14
W=.0 ,P=0.25	10.94	67.08	272.28	1362.16	429.25	69.89	2130.76	1771.39
W=.05 ,P=0.25	10.91	82.74	354.65	2383.67	2178.16	94.23	4999.21	2915.27
W=.10 ,P=0.25	10.74	83.66	357.37	2406.67	2217.73	95.34	5065.41	2943.03
W=.20 ,P=0.25	10.55	106.51	434.02	2659.28	2511.96	119.97	5711.76	3319.78
W=.30 ,P=0.25	10.46	121.67	482.33	2805.44	2733.44	136.17	6142.87	3545.59
W=.40 ,P=0.25	10.50	152.00	582.73	3116.86	3147.82	168.64	6999.41	4020.23
W=.50 ,P=0.25	10.49	158.82	590.85	3114.74	3065.80	175.41	6930.20	4039.81
W=.75 ,P=0.25	10.41	240.11	838.12	3865.14	4016.61	262.15	8959.97	5205.51
W=1 ,P=0.25	10.27	235.92	808.72	3806.24	3643.39	257.62	8494.27	5108.50
W=.0 ,P=0.5	10.82	66.67	272.45	1421.18	546.72	70.07	2307.02	1830.36
W=.05 ,P=0.5	10.72	92.56	401.95	3205.53	3343.14	110.52	7043.17	3810.55
W=.10 ,P=0.5	10.68	92.11	399.46	3185.51	3290.69	109.92	6967.77	3787.00
W=.20 ,P=0.5	10.36	124.16	503.01	3675.80	3845.05	145.58	8148.02	4448.55
W=.30 ,P=0.5	10.20	152.51	591.81	4006.92	4278.94	176.32	9030.17	4927.56
W=.40 ,P=0.5	10.06	195.63	727.81	4527.39	4849.43	223.18	10300.25	5674.00
W=.50 ,P=0.5	10.28	218.10	779.94	4666.19	4967.27	246.60	10631.49	5910.82
W=.75 ,P=0.5	10.20	332.97	1112.48	6083.83	6586.90	372.07	14116.17	7901.33
W=1 ,P=0.5	10.21	339.52	1106.16	6093.17	6245.61	378.79	13784.46	7917.64
W=.0 ,P=0.75	10.78	67.08	278.59	1471.84	656.77	71.00	2474.27	1888.50
W=.05 ,P=0.75	10.60	97.59	425.25	3808.49	4099.97	120.33	8431.30	4451.65
W=.10 ,P=0.75	10.66	100.43	433.23	3804.28	4078.55	123.11	8416.48	4461.04
W=.20 ,P=0.75	10.19	140.48	560.12	4489.20	4849.08	168.29	10038.88	5358.08
W=.30 ,P=0.75	10.09	183.33	686.58	5086.69	5518.81	215.59	11475.40	6172.18
W=.40 ,P=0.75	10.03	238.21	855.37	5845.05	6384.93	276.04	13323.55	7214.66
W=.50 ,P=0.75	10.06	256.50	891.98	5798.40	6253.23	293.83	13200.11	7240.70
W=.75 ,P=0.75	10.02	398.03	1288.43	7662.37	8277.97	449.42	17626.79	9798.25
W=1 ,P=0.75	10.00	409.26	1291.33	7724.14	8094.64	461.24	17519.36	9885.96
W=.0 ,P=0.9	10.78	68.04	282.61	1516.90	728.08	72.34	2595.62	1939.88
W=.05 ,P=0.9	10.63	102.51	443.16	4133.72	4512.80	127.82	9192.18	4807.21
W=.10 ,P=0.9	10.62	103.61	445.68	4119.88	4501.73	128.80	9170.90	4797.97
W=.20 ,P=0.9	10.10	146.33	577.45	4809.88	5248.51	176.67	10782.17	5710.33
W=.30 ,P=0.9	9.96	192.28	713.21	5547.66	6053.34	228.19	12506.48	6681.33
W=.40 ,P=0.9	10.01	255.99	903.57	6422.94	7039.28	298.37	14621.77	7880.86
W=.50 ,P=0.9	9.97	276.89	948.62	6383.79	6957.15	318.79	14566.44	7928.08
W=.75 ,P=0.9	9.97	427.40	1364.16	8422.46	9136.89	484.75	19350.91	10698.77
W=1 ,P=0.9	9.94	440.27	1367.17	8460.35	8961.57	498.04	19229.36	10765.83
W=.0 ,P=1.0	10.74	67.75	282.07	1522.33	744.09	72.10	2616.23	1944.24
W=.05 ,P=1.0	10.58	104.40	449.97	4260.25	4683.21	130.71	9497.83	4945.33

W=.10	,P=1.0	10.56	106.17	455.12	4290.64	4711.41	132.71	9563.34	4984.64
W=.20	,P=1.0	10.06	151.85	594.88	5052.73	5531.47	184.10	11330.91	5983.55
W=.30	,P=1.0	9.92	198.13	730.97	5806.05	6357.54	236.09	13092.69	6971.23
W=.40	,P=1.0	9.92	266.28	932.15	6706.18	7324.26	310.87	15228.86	8215.47
W=.50	,P=1.0	9.92	290.63	984.72	6707.02	7282.19	335.05	15264.55	8317.41
W=.75	,P=1.0	9.92	445.38	1409.05	8855.15	9589.90	506.13	20299.47	11215.69
W=1	,P=1.0	9.92	457.39	1406.93	8860.67	9404.82	518.32	20129.80	11243.30

	Permutation - Diameter 2							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	7.54	16.64	137.15	1042.22	62.50	18.54	1258.50	1214.54
W=.05 ,P=0.0	7.50	16.57	136.89	1030.37	62.00	18.47	1245.83	1202.30
W=.10 ,P=0.0	7.50	16.57	136.89	1030.37	62.00	18.47	1245.83	1202.30
W=.20 ,P=0.0	7.49	17.68	142.44	1031.19	62.00	19.58	1253.30	1210.88
W=.30 ,P=0.0	7.45	19.30	150.51	1032.42	62.00	21.20	1264.22	1223.42
W=.40 ,P=0.0	7.44	20.53	156.69	1033.59	62.00	22.43	1272.80	1233.23
W=.50 ,P=0.0	7.44	21.58	161.91	1036.92	62.00	23.48	1282.41	1243.88
W=.75 ,P=0.0	7.03	37.87	289.67	1058.32	81.74	39.77	1467.60	1425.63
W=1 ,P=0.0	7.01	39.14	300.54	1059.09	84.45	41.04	1483.21	1439.80
W=.0 ,P=0.25	6.95	22.13	238.78	1366.65	555.19	27.75	2182.75	1655.31
W=.05 ,P=0.25	6.44	33.46	414.41	1903.05	1311.59	45.36	3662.50	2396.27
W=.10 ,P=0.25	6.44	33.46	414.41	1903.05	1311.59	45.36	3662.50	2396.27
W=.20 ,P=0.25	6.06	40.19	501.99	2100.64	1565.62	53.57	4208.43	2696.38
W=.30 ,P=0.25	5.69	55.09	695.51	2472.54	2093.78	71.01	5316.91	3294.15
W=.40 ,P=0.25	5.73	85.06	1071.47	3374.50	3078.84	106.80	7609.86	4637.82
W=.50 ,P=0.25	5.78	91.79	1122.20	3395.49	3169.13	113.53	7778.60	4723.00
W=.75 ,P=0.25	5.67	191.52	2173.41	5315.92	5046.03	226.39	12726.88	7907.24
W=1 ,P=0.25	5.59	141.99	1206.18	4878.94	3376.69	175.40	9603.80	6402.51
W=.0 ,P=0.5	6.54	25.08	294.94	1555.02	847.90	32.60	2722.93	1907.64
W=.05 ,P=0.5	6.00	47.20	641.27	2589.21	2138.59	64.82	5416.27	3342.50
W=.10 ,P=0.5	6.00	47.20	641.27	2589.21	2138.59	64.82	5416.27	3342.50
W=.20 ,P=0.5	5.56	58.06	785.88	2902.42	2496.83	77.53	6243.18	3823.88
W=.30 ,P=0.5	5.36	79.89	1076.41	3674.64	3296.02	104.14	8126.96	4935.08
W=.40 ,P=0.5	5.19	118.35	1565.70	5032.34	4560.37	151.09	11276.76	6867.47
W=.50 ,P=0.5	5.23	132.37	1662.18	5013.89	4712.53	164.65	11520.97	6973.09
W=.75 ,P=0.5	5.19	283.86	3134.61	9129.43	7498.42	343.07	20046.32	12890.97
W=1 ,P=0.5	5.15	250.83	2092.15	9080.94	5939.72	312.04	17363.64	11735.96
W=.0 ,P=0.75	6.37	27.25	332.10	1716.96	1039.79	36.03	3116.10	2112.34
W=.05 ,P=0.75	5.75	55.63	786.82	3099.49	2659.57	77.00	6601.50	4018.93
W=.10 ,P=0.75	5.75	55.63	786.82	3099.49	2659.57	77.00	6601.50	4018.93
W=.20 ,P=0.75	5.30	71.20	993.01	3654.99	3204.57	96.13	7923.76	4815.32
W=.30 ,P=0.75	5.09	96.69	1328.84	4689.87	4125.07	127.94	10240.47	6243.34
W=.40 ,P=0.75	4.90	140.06	1878.77	6275.77	5477.27	180.77	13771.86	8475.36
W=.50 ,P=0.75	4.94	155.27	1955.23	6281.37	5584.88	195.70	13976.74	8587.56
W=.75 ,P=0.75	4.90	355.68	3735.94	11488.27	8541.13	429.85	24121.02	16009.74
W=1 ,P=0.75	4.84	321.31	2639.66	11547.10	7118.51	398.37	21626.57	14906.43
W=.0 ,P=0.9	6.20	27.61	339.58	1751.78	1083.03	36.73	3201.98	2155.69
W=.05 ,P=0.9	5.64	61.30	880.52	3458.92	2989.94	85.28	7390.68	4486.02
W=.10 ,P=0.9	5.64	61.30	880.52	3458.92	2989.94	85.28	7390.68	4486.02
W=.20 ,P=0.9	5.29	76.91	1085.89	4021.52	3528.45	104.30	8712.77	5288.61
W=.30 ,P=0.9	4.98	103.12	1425.60	5133.39	4441.98	137.22	11104.09	6799.33
W=.40 ,P=0.9	4.80	157.60	2083.98	7194.72	6032.05	204.19	15468.34	9640.48
W=.50 ,P=0.9	4.81	168.47	2101.61	6785.00	5938.86	212.08	14993.93	9267.15
W=.75 ,P=0.9	4.79	387.32	4039.01	12658.16	9221.48	468.77	26305.96	17553.25
W=1 ,P=0.9	4.79	368.89	2975.09	13041.18	7893.37	455.70	24278.52	16840.85
W=.0 ,P=1.0	6.20	28.62	355.46	1804.88	1150.46	38.14	3339.41	2227.09
W=.05 ,P=1.0	5.58	64.03	924.87	3630.62	3140.89	89.28	7760.40	4708.79

W=.10	,P=1.0	5.58	64.03	924.87	3630.62	3140.89	89.28	7760.40	4708.79
W=.20	,P=1.0	5.19	78.26	1114.46	4135.95	3617.43	106.41	8946.10	5435.07
W=.30	,P=1.0	4.92	106.62	1485.61	5324.94	4614.04	141.99	11531.21	7059.15
W=.40	,P=1.0	4.71	161.79	2138.30	7460.37	6171.35	209.94	15931.80	9970.38
W=.50	,P=1.0	4.71	176.42	2178.90	7136.11	6125.99	222.19	15617.41	9713.61
W=.75	,P=1.0	4.71	404.09	4165.57	13096.73	9426.60	488.35	27092.98	18154.73
W=1	,P=1.0	4.71	383.79	3079.69	13561.14	8104.15	473.93	25128.77	17498.55

	Permutation - Diameter 4							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	6.78	62.58	322.08	984.33	169.67	64.04	1538.64	1433.01
W=.05 ,P=0.0	6.78	62.58	322.08	936.74	169.67	64.04	1491.05	1385.42
W=.10 ,P=0.0	6.78	62.59	322.13	936.74	169.67	64.05	1491.11	1385.49
W=.20 ,P=0.0	6.74	63.23	325.35	937.28	169.67	64.69	1495.53	1390.55
W=.30 ,P=0.0	6.74	63.65	327.43	937.57	169.67	65.11	1498.31	1393.75
W=.40 ,P=0.0	6.74	63.90	328.68	937.91	169.67	65.36	1500.14	1395.83
W=.50 ,P=0.0	6.74	64.14	329.90	938.72	169.67	65.60	1502.42	1398.36
W=.75 ,P=0.0	6.66	84.39	435.51	945.97	183.04	85.85	1648.90	1551.71
W=1 ,P=0.0	6.61	86.04	443.79	949.26	188.19	87.50	1667.27	1566.58
W=.0 ,P=0.25	6.75	64.62	333.57	992.99	207.89	66.17	1599.06	1457.34
W=.05 ,P=0.25	5.60	121.59	707.38	1469.05	3499.42	131.30	5797.42	2429.31
W=.10 ,P=0.25	5.60	121.59	707.38	1469.05	3499.42	131.30	5797.42	2429.31
W=.20 ,P=0.25	5.56	124.59	726.01	1499.50	3633.82	134.70	5983.91	2484.78
W=.30 ,P=0.25	5.45	128.15	740.96	1492.92	3514.79	138.18	5876.81	2500.20
W=.40 ,P=0.25	5.48	129.62	751.19	1499.08	3525.86	139.66	5905.75	2519.54
W=.50 ,P=0.25	5.52	125.47	730.28	1496.91	3569.04	135.48	5921.69	2488.13
W=.75 ,P=0.25	5.41	223.24	1252.91	1838.67	4848.18	238.12	8162.98	3552.93
W=1 ,P=0.25	5.41	229.91	1286.47	1845.68	6354.63	244.80	9716.67	3606.84
W=.0 ,P=0.5	6.73	63.98	330.44	991.96	215.58	65.55	1601.94	1451.92
W=.05 ,P=0.5	5.17	136.64	806.20	1650.32	4488.23	149.59	7081.39	2742.75
W=.10 ,P=0.5	5.17	136.64	806.20	1650.32	4488.23	149.59	7081.39	2742.75
W=.20 ,P=0.5	5.16	146.43	857.32	1696.01	4588.03	159.95	7287.78	2859.70
W=.30 ,P=0.5	5.13	154.72	900.96	1721.44	4671.66	168.62	7448.77	2945.73
W=.40 ,P=0.5	5.09	155.23	902.70	1714.39	4649.27	169.01	7421.58	2941.32
W=.50 ,P=0.5	5.05	156.00	905.84	1700.55	4625.99	169.52	7388.36	2931.90
W=.75 ,P=0.5	5.13	283.65	1579.49	2156.01	6288.34	304.23	10307.48	4323.38
W=1 ,P=0.5	5.03	272.63	1519.95	2122.81	8270.76	292.66	12186.15	4208.05
W=.0 ,P=0.75	6.72	63.99	330.44	992.30	213.56	65.56	1600.27	1452.28
W=.05 ,P=0.75	4.95	156.14	914.59	1783.67	5052.44	171.35	7906.83	3025.74
W=.10 ,P=0.75	4.95	156.14	914.59	1783.67	5052.44	171.35	7906.83	3025.74
W=.20 ,P=0.75	4.92	162.11	944.55	1810.03	5195.93	177.84	8112.61	3094.53
W=.30 ,P=0.75	4.88	163.78	954.89	1825.66	5244.90	179.75	8189.23	3124.07
W=.40 ,P=0.75	4.84	169.78	986.11	1829.78	5200.44	185.73	8186.09	3171.39
W=.50 ,P=0.75	4.83	173.30	1002.02	1823.28	5168.19	189.01	8166.79	3187.60
W=.75 ,P=0.75	4.86	298.14	1659.42	2286.44	6868.10	321.30	11112.10	4565.29
W=1 ,P=0.75	4.83	287.83	1603.62	2254.97	9133.53	310.45	13279.94	4456.86
W=.0 ,P=0.9	6.72	63.92	330.27	992.77	220.82	65.51	1607.77	1452.46
W=.05 ,P=0.9	4.84	162.22	948.61	1836.33	5404.38	178.51	8351.53	3125.66
W=.10 ,P=0.9	4.84	162.22	948.61	1836.33	5404.38	178.51	8351.53	3125.66
W=.20 ,P=0.9	4.84	167.68	977.36	1866.37	5541.10	184.48	8552.51	3195.88
W=.30 ,P=0.9	4.79	172.10	999.91	1872.31	5499.21	188.90	8543.52	3233.21
W=.40 ,P=0.9	4.77	178.71	1035.17	1889.80	5465.48	195.72	8569.16	3299.40
W=.50 ,P=0.9	4.72	178.64	1031.80	1865.53	5357.71	195.20	8433.68	3271.16
W=.75 ,P=0.9	4.75	303.85	1689.63	2335.44	7212.64	328.04	11541.55	4656.95
W=1 ,P=0.9	4.74	299.11	1662.25	2315.49	9584.69	322.97	13861.54	4599.82
W=.0 ,P=1.0	6.72	64.63	334.01	994.33	223.26	66.23	1616.22	1459.19
W=.05 ,P=1.0	4.80	165.98	970.07	1869.53	5562.93	182.91	8568.51	3188.48

W=.10 ,P=1.0	4.80	165.98	970.07	1869.53	5562.93	182.91	8568.51	3188.48
W=.20 ,P=1.0	4.78	170.88	995.39	1888.42	5612.92	188.08	8667.60	3242.77
W=.30 ,P=1.0	4.75	175.38	1019.52	1902.99	5638.54	192.75	8736.42	3290.63
W=.40 ,P=1.0	4.71	182.90	1057.93	1921.39	5640.38	200.54	8802.60	3362.76
W=.50 ,P=1.0	4.71	186.02	1072.78	1913.11	5595.00	203.42	8766.91	3375.33
W=.75 ,P=1.0	4.71	308.29	1713.92	2370.82	7330.99	333.16	11724.02	4726.19
W=1 ,P=1.0	4.71	308.10	1708.82	2358.61	9866.47	332.74	14241.99	4708.26

	5-puzzle - Diameter 2							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	23.62	37.69	107.44	1183.56	73.93	39.97	1402.62	1368.65
W=.05 ,P=0.0	23.62	37.69	107.47	1182.76	74.02	39.97	1401.93	1367.89
W=.10 ,P=0.0	23.62	37.69	107.47	1182.76	74.02	39.97	1401.93	1367.89
W=.20 ,P=0.0	23.07	39.41	112.22	1183.31	75.70	41.69	1410.63	1376.62
W=.30 ,P=0.0	22.88	48.33	133.67	1189.07	80.10	50.61	1451.17	1421.68
W=.40 ,P=0.0	22.82	56.73	154.96	1195.14	86.43	59.01	1493.25	1465.83
W=.50 ,P=0.0	22.82	64.63	175.04	1201.27	92.53	66.91	1533.46	1507.84
W=.75 ,P=0.0	21.84	89.55	242.08	1219.25	120.20	91.82	1671.07	1642.69
W=1 ,P=0.0	21.85	93.96	253.01	1223.16	122.50	96.23	1692.62	1666.35
W=.0 ,P=0.25	22.92	68.51	223.05	1704.74	432.57	78.77	2428.86	2075.07
W=.05 ,P=0.25	22.64	78.59	263.44	1798.23	531.41	90.94	2671.66	2231.20
W=.10 ,P=0.25	22.71	76.27	254.13	1783.48	515.69	88.63	2629.57	2202.50
W=.20 ,P=0.25	21.91	73.43	244.88	1823.42	506.87	86.09	2648.59	2227.81
W=.30 ,P=0.25	21.17	134.37	436.88	2309.34	876.51	151.85	3757.09	3032.43
W=.40 ,P=0.25	21.03	181.63	575.69	2872.75	1170.60	204.29	4800.66	3834.35
W=.50 ,P=0.25	20.84	167.36	518.49	2873.06	1067.67	189.74	4626.57	3748.64
W=.75 ,P=0.25	20.89	174.40	522.40	2985.86	1034.22	196.40	4716.87	3879.05
W=1 ,P=0.25	21.05	163.52	468.24	2989.39	905.00	183.57	4526.14	3804.71
W=.0 ,P=0.5	22.39	108.02	367.17	2176.52	855.87	125.75	3507.57	2777.45
W=.05 ,P=0.5	21.79	117.51	406.57	2285.81	967.40	138.14	3777.28	2948.02
W=.10 ,P=0.5	21.64	119.60	412.00	2319.57	986.30	140.47	3837.46	2991.63
W=.20 ,P=0.5	21.06	116.67	404.23	2420.34	1001.69	138.65	3942.92	3079.88
W=.30 ,P=0.5	20.23	224.03	741.93	3476.07	1708.62	255.28	6150.65	4697.31
W=.40 ,P=0.5	19.95	309.48	990.05	4778.00	2289.63	348.67	8367.15	6426.18
W=.50 ,P=0.5	20.03	301.23	936.78	5171.19	2277.70	342.16	8686.89	6751.35
W=.75 ,P=0.5	20.16	286.54	860.35	5997.56	2298.25	330.27	9442.69	7474.71
W=1 ,P=0.5	20.14	255.58	723.89	6082.42	2099.95	297.56	9161.84	7359.45
W=.0 ,P=0.75	21.75	126.24	434.08	2479.15	1084.21	148.06	4123.68	3187.53
W=.05 ,P=0.75	21.29	142.44	497.27	2700.98	1277.43	169.29	4618.11	3509.98
W=.10 ,P=0.75	21.42	143.33	500.38	2737.41	1290.99	170.36	4672.10	3551.47
W=.20 ,P=0.75	20.33	144.66	507.12	2870.78	1337.04	173.15	4859.59	3695.70
W=.30 ,P=0.75	19.75	275.86	908.36	4500.91	2201.56	315.22	7886.69	6000.35
W=.40 ,P=0.75	19.58	412.82	1299.31	6743.67	3134.58	464.62	11590.37	8920.41
W=.50 ,P=0.75	19.48	427.05	1299.24	7985.10	3368.39	485.40	13079.77	10196.78
W=.75 ,P=0.75	19.63	438.42	1288.61	10886.76	3986.88	510.81	16600.66	13124.60
W=1 ,P=0.75	19.58	401.23	1108.47	12318.75	4310.47	479.47	18138.90	14307.91
W=.0 ,P=0.9	20.96	131.66	453.76	2587.00	1156.16	154.73	4328.58	3327.15
W=.05 ,P=0.9	21.13	149.96	525.49	2903.60	1395.36	179.28	4974.40	3758.32
W=.10 ,P=0.9	21.15	151.54	530.13	2887.36	1388.03	180.53	4957.06	3749.55
W=.20 ,P=0.9	20.17	156.17	548.34	3115.12	1482.19	187.58	5301.81	4007.20
W=.30 ,P=0.9	19.39	295.97	971.56	4971.92	2416.20	338.85	8655.64	6578.29
W=.40 ,P=0.9	19.44	460.60	1437.05	7874.03	3537.77	519.12	13309.44	10290.79
W=.50 ,P=0.9	19.41	483.53	1456.97	9304.62	3808.56	549.18	15053.67	11794.29
W=.75 ,P=0.9	19.44	506.02	1472.85	13185.84	4687.55	591.26	19852.25	15755.96
W=1 ,P=0.9	19.40	464.75	1272.58	14952.85	5112.69	557.44	21802.87	17247.62
W=.0 ,P=1.0	20.96	136.16	470.05	2682.89	1214.81	160.38	4503.90	3449.47
W=.05 ,P=1.0	21.01	153.79	539.55	2995.63	1446.20	184.24	5135.16	3873.20

W=.10 ,P=1.0	21.00	154.20	540.94	3002.39	1448.62	184.63	5146.15	3882.16
W=.20 ,P=1.0	19.97	159.56	561.59	3195.56	1534.61	192.05	5451.32	4108.76
W=.30 ,P=1.0	19.33	307.81	1008.75	5262.21	2537.73	352.75	9116.50	6931.52
W=.40 ,P=1.0	19.29	480.66	1493.14	8399.68	3717.17	542.35	14090.64	10915.82
W=.50 ,P=1.0	19.29	512.06	1535.63	10084.77	4051.31	582.25	16183.77	12714.71
W=.75 ,P=1.0	19.29	540.35	1565.62	14341.67	5000.23	631.90	21447.86	17079.53
W=1 ,P=1.0	19.29	497.57	1358.81	16260.66	5474.71	597.84	23591.74	18714.87

	5-puzzle - Diameter 4							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marching Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	22.29	48.90	140.90	941.96	173.85	50.42	1305.60	1182.16
W=.05 ,P=0.0	22.29	48.90	140.90	939.64	173.85	50.42	1303.28	1179.85
W=.10 ,P=0.0	22.26	49.84	143.06	940.38	173.85	51.36	1307.12	1184.63
W=.20 ,P=0.0	21.56	69.22	188.12	959.18	173.85	70.74	1390.37	1287.25
W=.30 ,P=0.0	21.42	79.79	212.92	970.24	173.85	81.31	1436.79	1344.25
W=.40 ,P=0.0	21.42	86.56	228.78	976.82	173.85	88.08	1466.00	1380.23
W=.50 ,P=0.0	21.42	91.00	239.11	981.13	173.85	92.52	1485.08	1403.75
W=.75 ,P=0.0	21.09	116.74	305.27	1004.74	200.27	118.26	1627.01	1545.00
W=1 ,P=0.0	21.09	119.67	312.18	1007.96	200.27	121.19	1640.08	1561.00
W=.0 ,P=0.25	22.07	68.38	221.15	1064.73	504.58	71.88	1858.83	1426.13
W=.05 ,P=0.25	22.06	73.12	239.06	1116.86	597.31	77.51	2026.35	1506.54
W=.10 ,P=0.25	22.04	75.01	245.10	1128.59	621.01	79.47	2069.70	1528.16
W=.20 ,P=0.25	21.00	132.07	408.61	1350.53	937.20	138.44	2828.40	2029.64
W=.30 ,P=0.25	20.72	171.07	512.74	1512.23	1153.03	178.78	3349.06	2374.81
W=.40 ,P=0.25	20.73	187.62	556.20	1608.81	1283.48	196.09	3636.11	2548.72
W=.50 ,P=0.25	20.74	180.18	516.82	1626.79	1206.76	188.96	3530.55	2512.75
W=.75 ,P=0.25	20.77	188.36	512.23	1704.06	1161.52	197.90	3566.16	2602.54
W=1 ,P=0.25	20.79	176.37	457.81	1672.69	935.48	185.83	3242.35	2492.70
W=.0 ,P=0.5	21.74	78.11	260.58	1142.52	673.53	82.63	2154.73	1563.83
W=.05 ,P=0.5	21.85	82.35	277.05	1235.16	807.58	88.22	2402.14	1682.78
W=.10 ,P=0.5	21.79	87.08	292.79	1268.08	855.90	93.26	2503.85	1741.21
W=.20 ,P=0.5	20.36	161.41	503.85	1656.78	1347.06	171.00	3669.10	2493.04
W=.30 ,P=0.5	20.15	242.94	729.30	2127.73	1916.13	256.56	5016.08	3356.51
W=.40 ,P=0.5	20.23	286.96	848.00	2452.22	2283.50	303.51	5870.68	3890.68
W=.50 ,P=0.5	20.26	280.79	801.96	2545.98	2235.01	298.39	5863.73	3927.11
W=.75 ,P=0.5	20.25	292.05	799.03	2884.11	2436.67	313.44	6411.86	4288.63
W=1 ,P=0.5	20.31	279.63	723.46	2968.47	2283.87	302.42	6255.41	4273.97
W=.0 ,P=0.75	21.83	81.49	274.17	1197.23	763.07	86.65	2315.95	1639.54
W=.05 ,P=0.75	21.61	90.58	309.96	1339.68	970.57	97.67	2710.78	1837.89
W=.10 ,P=0.75	21.62	99.14	340.38	1388.24	1064.88	106.69	2892.64	1934.44
W=.20 ,P=0.75	19.99	195.75	609.73	1981.11	1725.85	208.50	4512.43	2995.08
W=.30 ,P=0.75	19.63	289.81	864.72	2585.69	2381.30	307.94	6121.51	4048.15
W=.40 ,P=0.75	19.71	347.72	1016.49	3037.90	2849.01	369.94	7251.12	4772.05
W=.50 ,P=0.75	19.85	355.10	1007.44	3275.41	2934.85	379.82	7572.80	5017.77
W=.75 ,P=0.75	19.81	366.54	999.67	3742.32	3218.57	396.43	8327.09	5504.95
W=1 ,P=0.75	19.89	360.09	929.21	3987.14	3222.14	393.25	8498.57	5669.68
W=.0 ,P=0.9	21.79	84.85	288.22	1229.03	821.03	90.35	2423.13	1692.45
W=.05 ,P=0.9	21.70	94.54	326.04	1387.13	1047.59	102.12	2855.29	1909.82
W=.10 ,P=0.9	21.50	99.03	340.70	1416.32	1091.97	106.92	2948.01	1962.96
W=.20 ,P=0.9	19.70	203.74	634.61	2095.40	1842.07	217.63	4775.81	3151.36
W=.30 ,P=0.9	19.43	311.19	926.77	2779.83	2592.31	331.19	6610.09	4348.97
W=.40 ,P=0.9	19.48	368.55	1076.04	3270.75	3075.49	393.04	7790.82	5108.37
W=.50 ,P=0.9	19.47	374.27	1058.10	3485.72	3115.78	401.07	8033.86	5319.15
W=.75 ,P=0.9	19.41	398.85	1084.55	4099.75	3498.78	432.25	9081.93	6015.40
W=1 ,P=0.9	19.54	387.67	998.88	4337.36	3537.31	424.37	9261.22	6148.28
W=.0 ,P=1.0	21.66	85.61	291.52	1244.23	842.98	91.27	2464.34	1712.63
W=.05 ,P=1.0	21.54	97.10	336.37	1417.51	1095.99	105.03	2946.96	1956.00

W=.10	,P=1.0	21.43	101.53	350.76	1455.31	1144.14	109.81	3051.73	2017.40
W=.20	,P=1.0	19.61	210.12	654.84	2167.97	1932.78	224.77	4965.70	3257.69
W=.30	,P=1.0	19.29	321.03	954.96	2879.00	2687.55	341.99	6842.54	4496.97
W=.40	,P=1.0	19.29	380.25	1107.05	3358.81	3151.48	405.56	7997.59	5251.67
W=.50	,P=1.0	19.29	386.43	1090.63	3613.16	3217.15	414.47	8307.36	5504.68
W=.75	,P=1.0	19.29	411.99	1119.81	4243.71	3619.29	446.82	9394.78	6222.32
W=1	,P=1.0	19.29	398.75	1026.86	4473.97	3643.61	436.87	9543.18	6336.43

	T.O.H - Diameter 2							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	82.02	136.23	412.87	3557.08	214.55	139.11	4320.72	4245.28
W=.05 ,P=0.0	81.77	135.85	411.70	3556.10	213.91	138.73	4317.56	4242.38
W=.10 ,P=0.0	81.77	135.85	411.70	3556.10	213.91	138.73	4317.56	4242.38
W=.20 ,P=0.0	80.80	137.70	417.17	3556.28	215.93	140.58	4327.08	4251.73
W=.30 ,P=0.0	73.79	188.48	570.07	3581.90	241.74	191.36	4582.19	4531.81
W=.40 ,P=0.0	72.52	248.29	750.49	3627.27	288.94	251.18	4914.99	4877.23
W=.50 ,P=0.0	72.48	274.04	828.33	3647.55	308.34	276.94	5058.26	5026.85
W=.75 ,P=0.0	71.77	319.85	966.54	3681.37	357.58	322.74	5325.33	5290.49
W=1 ,P=0.0	71.85	328.63	993.05	3688.81	363.67	331.53	5374.15	5342.01
W=.0 ,P=0.25	76.08	205.25	622.15	6569.06	1076.87	223.58	8473.32	7620.03
W=.05 ,P=0.25	76.06	209.97	637.19	6617.07	1107.72	229.02	8571.94	7693.24
W=.10 ,P=0.25	76.08	203.91	618.17	6586.46	1084.54	222.85	8493.07	7631.38
W=.20 ,P=0.25	76.29	219.47	664.49	6734.33	1155.01	239.39	8773.29	7857.67
W=.30 ,P=0.25	70.54	384.37	1163.68	8755.68	2005.59	413.67	12309.31	10717.39
W=.40 ,P=0.25	70.03	567.64	1716.17	10899.50	2808.73	602.17	15992.03	13785.47
W=.50 ,P=0.25	69.26	508.64	1537.41	9974.27	2452.34	541.23	14472.65	12561.54
W=.75 ,P=0.25	69.83	471.49	1423.15	10027.17	2135.32	500.25	14057.12	12422.06
W=1 ,P=0.25	69.91	437.67	1319.40	8920.00	1715.65	461.18	12392.72	11138.25
W=.0 ,P=0.5	74.17	298.99	904.17	8104.10	1936.21	329.64	11243.46	9636.89
W=.05 ,P=0.5	73.54	307.56	930.05	8396.04	2008.32	340.76	11641.95	9974.39
W=.10 ,P=0.5	74.56	307.37	929.39	8377.52	1999.49	340.36	11613.77	9954.64
W=.20 ,P=0.5	72.82	312.22	944.20	8356.07	2012.11	345.00	11624.60	9957.49
W=.30 ,P=0.5	68.82	584.00	1762.91	11891.27	3688.23	632.52	17926.40	14870.69
W=.40 ,P=0.5	68.15	1179.01	3577.65	19565.79	7377.16	1248.23	31699.60	25570.67
W=.50 ,P=0.5	68.25	1179.44	3583.84	20758.11	7661.20	1250.39	33182.59	26771.78
W=.75 ,P=0.5	68.54	710.49	2150.50	22790.05	6031.22	778.40	31682.25	26429.43
W=1 ,P=0.5	68.70	576.24	1737.34	19997.88	4795.49	629.97	27106.95	22941.43
W=.0 ,P=0.75	72.43	356.44	1077.08	9003.24	2495.99	394.64	12932.75	10831.40
W=.05 ,P=0.75	72.17	374.93	1130.61	9419.37	2671.42	417.03	13596.32	11341.93
W=.10 ,P=0.75	72.28	372.59	1123.89	9362.89	2643.90	414.68	13503.27	11274.05
W=.20 ,P=0.75	71.31	385.27	1162.24	9463.77	2733.19	427.79	13744.47	11439.07
W=.30 ,P=0.75	68.02	657.92	1985.32	12980.21	4306.61	713.74	19930.06	16337.19
W=.40 ,P=0.75	67.71	1721.66	5243.16	27313.18	11342.06	1812.24	45620.05	36090.23
W=.50 ,P=0.75	67.58	2173.94	6646.31	35624.55	15185.27	2283.57	59630.07	46728.36
W=.75 ,P=0.75	68.18	1241.61	3772.09	54278.99	17196.76	1386.59	76489.45	60679.28
W=1 ,P=0.75	68.52	967.30	2919.94	56196.59	17058.55	1107.56	77142.37	61191.38
W=.0 ,P=0.9	72.00	377.07	1137.43	9331.25	2725.24	418.52	13570.98	11264.26
W=.05 ,P=0.9	71.76	394.03	1186.44	9739.85	2882.19	439.16	14202.50	11759.47
W=.10 ,P=0.9	71.53	394.48	1188.53	9708.59	2871.57	439.39	14163.17	11730.99
W=.20 ,P=0.9	71.02	408.22	1229.35	9708.78	2938.31	453.88	14284.65	11800.22
W=.30 ,P=0.9	67.92	695.71	2100.09	13513.05	4644.38	755.31	20953.23	17064.16
W=.40 ,P=0.9	67.45	1879.93	5728.49	30458.74	12699.51	1978.49	50766.66	40045.64
W=.50 ,P=0.9	67.42	2723.72	8348.52	43111.15	19112.23	2850.15	73295.62	57033.54
W=.75 ,P=0.9	67.60	1598.93	4862.94	75024.67	25502.20	1793.66	106988.74	83280.20
W=1 ,P=0.9	67.70	1310.12	3955.11	87882.92	30651.17	1523.89	123799.32	94672.03
W=.0 ,P=1.0	71.45	384.91	1160.77	9481.22	2824.08	427.97	13850.97	11454.86
W=.05 ,P=1.0	71.59	406.92	1225.35	10009.65	3039.44	454.08	14681.36	12095.99

W=.10	,P=1.0	71.60	409.13	1232.16	10009.26	3046.35	456.33	14696.89	12106.87
W=.20	,P=1.0	70.74	423.38	1274.98	9971.16	3095.00	471.16	14764.51	12140.67
W=.30	,P=1.0	67.70	704.71	2127.02	13760.37	4749.14	765.59	21341.23	17357.68
W=.40	,P=1.0	67.36	1978.06	6029.41	32165.31	13481.56	2080.87	53654.33	42253.65
W=.50	,P=1.0	67.35	2903.22	8900.06	46371.47	20568.16	3037.41	78742.89	61212.14
W=.75	,P=1.0	67.35	1774.91	5400.44	85171.84	29732.23	1992.15	122079.40	94339.32
W=1	,P=1.0	67.35	1505.96	4546.15	105179.16	38414.02	1758.91	149645.27	112990.16

	T.O.H - Diameter 4							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	82.49	176.19	529.92	2835.20	327.00	178.29	3868.30	3719.58
W=.05 ,P=0.0	82.49	176.20	529.94	2834.48	327.03	178.30	3867.64	3718.91
W=.10 ,P=0.0	82.49	176.63	531.25	2834.62	328.43	178.73	3870.92	3721.21
W=.20 ,P=0.0	79.76	255.09	767.39	2886.73	382.79	257.19	4291.99	4166.39
W=.30 ,P=0.0	78.97	336.32	1011.46	2960.10	418.64	338.42	4726.52	4646.30
W=.40 ,P=0.0	78.97	355.11	1067.86	2977.35	424.69	357.21	4825.00	4757.53
W=.50 ,P=0.0	78.97	364.77	1096.82	2986.38	428.28	366.87	4876.24	4814.83
W=.75 ,P=0.0	78.84	392.62	1180.53	3011.89	459.73	394.72	5044.76	4979.75
W=1 ,P=0.0	79.15	398.44	1197.98	3017.38	463.74	400.53	5077.53	5014.33
W=.0 ,P=0.25	79.22	202.33	609.38	3285.15	945.24	209.22	5042.08	4306.07
W=.05 ,P=0.25	74.98	192.51	579.57	3277.60	908.98	199.28	4958.65	4248.96
W=.10 ,P=0.25	75.10	192.34	579.13	3293.32	917.92	199.33	4982.71	4264.12
W=.20 ,P=0.25	71.07	311.18	937.77	3696.27	1515.70	321.27	6460.90	5266.48
W=.30 ,P=0.25	70.24	506.72	1526.83	4508.58	2428.05	520.70	8970.16	7062.82
W=.40 ,P=0.25	69.08	545.64	1644.59	4671.27	2592.70	560.13	9454.19	7421.62
W=.50 ,P=0.25	68.64	500.15	1507.00	4564.94	2488.99	514.25	9061.07	7086.33
W=.75 ,P=0.25	71.46	458.52	1379.09	4554.38	2046.92	471.87	8438.91	6863.85
W=1 ,P=0.25	71.81	435.08	1307.38	4362.48	1661.83	446.58	7766.76	6551.51
W=.0 ,P=0.5	75.03	220.18	663.00	3490.96	1336.30	229.81	5710.43	4603.94
W=.05 ,P=0.5	72.20	221.64	667.44	3555.78	1418.11	232.28	5862.97	4677.14
W=.10 ,P=0.5	70.92	217.12	653.90	3530.31	1370.97	227.56	5772.30	4628.89
W=.20 ,P=0.5	68.95	370.19	1116.63	4103.78	2324.86	384.81	7915.46	5975.41
W=.30 ,P=0.5	67.74	711.78	2149.96	5780.91	4650.20	734.14	13292.85	9376.79
W=.40 ,P=0.5	67.78	899.72	2722.35	6723.90	6308.65	925.71	16654.61	11271.67
W=.50 ,P=0.5	68.13	831.79	2516.43	6836.34	6171.18	858.28	16355.73	11042.83
W=.75 ,P=0.5	68.20	622.21	1873.20	7303.86	4959.67	649.84	14758.94	10449.10
W=1 ,P=0.5	68.36	564.75	1697.79	7019.26	4285.41	590.04	13567.20	9871.83
W=.0 ,P=0.75	72.61	237.88	716.41	3638.44	1661.38	249.74	6254.10	4842.46
W=.05 ,P=0.75	69.79	235.55	709.24	3691.23	1695.34	248.30	6331.35	4884.31
W=.10 ,P=0.75	70.29	235.48	708.96	3701.94	1682.67	248.28	6329.05	4894.65
W=.20 ,P=0.75	68.61	390.77	1177.84	4245.07	2628.45	407.51	8442.12	6221.19
W=.30 ,P=0.75	67.43	861.92	2606.34	6793.55	6379.32	889.48	16641.13	11151.29
W=.40 ,P=0.75	67.45	1175.91	3564.64	8492.80	9449.60	1210.07	22682.95	14443.42
W=.50 ,P=0.75	67.39	1237.47	3754.63	9605.90	10979.84	1276.46	25577.83	15874.45
W=.75 ,P=0.75	67.45	959.89	2893.32	12667.40	11808.16	1013.76	28328.76	17534.36
W=1 ,P=0.75	67.74	867.83	2608.81	12855.74	11628.14	921.05	27960.51	17253.43
W=.0 ,P=0.9	71.24	238.55	718.29	3661.06	1712.01	250.92	6329.90	4868.81
W=.05 ,P=0.9	69.66	241.62	727.09	3764.01	1806.40	255.44	6539.11	4988.15
W=.10 ,P=0.9	70.01	245.68	739.47	3751.88	1825.10	259.44	6562.13	4996.46
W=.20 ,P=0.9	68.20	391.46	1180.17	4223.20	2638.40	408.49	8433.22	6203.31
W=.30 ,P=0.9	67.46	930.88	2817.75	7224.73	7230.71	960.49	18204.06	11933.84
W=.40 ,P=0.9	67.37	1317.80	3997.43	9377.39	11001.24	1355.99	25693.85	16048.61
W=.50 ,P=0.9	67.37	1473.41	4473.05	11054.21	13480.61	1518.64	30481.28	18519.30
W=.75 ,P=0.9	67.38	1142.98	3446.01	15276.09	15231.29	1208.88	35096.36	21073.95
W=1 ,P=0.9	67.39	1072.39	3224.01	17134.94	17927.62	1146.04	39358.95	22577.37
W=.0 ,P=1.0	71.07	242.31	729.57	3700.14	1771.57	255.14	6443.58	4927.15
W=.05 ,P=1.0	69.56	245.59	738.94	3788.59	1861.58	259.83	6634.69	5032.94

W=.10	,P=1.0	69.53	246.45	741.64	3774.71	1853.63	260.53	6616.42	5023.33
W=.20	,P=1.0	68.16	398.57	1201.49	4261.54	2737.78	416.07	8599.37	6277.66
W=.30	,P=1.0	67.47	967.91	2929.56	7491.35	7602.41	998.85	18991.22	12387.66
W=.40	,P=1.0	67.35	1380.40	4188.27	9834.97	11723.52	1420.59	27127.15	16824.22
W=.50	,P=1.0	67.35	1571.39	4770.69	11677.81	14397.64	1619.32	32417.52	19639.20
W=.75	,P=1.0	67.35	1266.35	3819.25	17233.06	17969.63	1341.30	40288.28	23659.94
W=1	,P=1.0	67.35	1206.40	3626.72	19609.61	21468.04	1291.59	45910.76	25734.31

	KL-2000 - Diameter 2							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	12.75	73.99	1166.43	3622.30	296.70	76.05	5159.41	4938.76
W=.05 ,P=0.0	12.78	74.42	1175.11	3600.63	297.40	76.49	5147.55	4926.64
W=.10 ,P=0.0	12.78	74.42	1175.11	3600.63	297.40	76.49	5147.55	4926.64
W=.20 ,P=0.0	12.72	85.48	1350.92	3606.83	301.93	87.55	5345.16	5130.78
W=.30 ,P=0.0	12.39	132.73	2071.17	3632.00	315.56	134.79	6151.45	5970.68
W=.40 ,P=0.0	12.05	162.54	2421.26	3653.41	344.14	164.61	6581.34	6401.81
W=.50 ,P=0.0	11.95	192.09	2776.64	3682.05	377.31	194.15	7028.07	6844.92
W=.75 ,P=0.0	11.36	323.36	4465.01	3796.22	499.32	325.42	9083.90	8910.00
W=1 ,P=0.0	11.45	344.97	4708.18	3814.36	513.76	347.03	9381.25	9214.52
W=.0 ,P=0.25	11.80	144.29	1725.85	5465.14	2901.06	158.06	10236.34	7493.34
W=.05 ,P=0.25	11.81	182.82	2081.60	6779.20	4191.58	203.10	13235.19	9246.70
W=.10 ,P=0.25	11.81	182.80	2081.56	6779.19	4191.46	203.08	13235.01	9246.63
W=.20 ,P=0.25	11.46	197.46	2274.81	7109.72	4441.85	219.16	14023.83	9801.14
W=.30 ,P=0.25	10.86	364.94	4261.49	11034.54	7587.77	400.08	23248.74	16061.04
W=.40 ,P=0.25	10.25	707.42	8176.01	19634.70	13793.53	769.39	42311.64	29287.50
W=.50 ,P=0.25	10.34	831.25	9806.35	21692.79	14898.80	898.31	47229.18	33228.69
W=.75 ,P=0.25	10.39	1130.84	13409.07	29174.78	19480.21	1220.62	63194.89	44935.30
W=1 ,P=0.25	10.55	1132.51	13743.97	29638.67	21098.56	1222.61	65613.70	45737.75
W=.0 ,P=0.5	11.55	146.89	1726.22	5744.45	3241.71	162.66	10859.26	7780.21
W=.05 ,P=0.5	11.51	197.19	2121.74	7432.55	4926.02	222.14	14677.48	9973.60
W=.10 ,P=0.5	11.51	197.11	2121.56	7432.52	4925.62	222.06	14676.80	9973.24
W=.20 ,P=0.5	11.29	202.92	2308.69	7928.88	5123.71	229.17	15564.20	10669.66
W=.30 ,P=0.5	10.70	378.06	4304.76	13001.46	8980.02	421.36	26664.28	18105.63
W=.40 ,P=0.5	10.07	767.76	8644.74	23467.86	16115.44	843.34	48995.79	33723.69
W=.50 ,P=0.5	10.08	919.41	10551.50	26951.14	18038.14	1004.44	56460.18	39426.48
W=.75 ,P=0.5	10.13	1282.18	14566.93	37111.10	24222.74	1398.71	77182.94	54358.91
W=1 ,P=0.5	10.25	1325.24	15281.23	39433.47	27997.45	1447.56	84037.38	57487.49
W=.0 ,P=0.75	11.58	153.05	1765.00	6026.37	3606.17	170.73	11550.59	8115.14
W=.05 ,P=0.75	11.34	197.80	2143.88	8077.98	5386.72	225.54	15806.37	10645.19
W=.10 ,P=0.75	11.32	197.52	2141.60	8075.11	5387.40	225.21	15801.63	10639.43
W=.20 ,P=0.75	11.34	208.54	2307.56	8471.98	5672.56	237.96	16660.63	11226.03
W=.30 ,P=0.75	10.63	374.81	4242.69	13563.55	9423.86	421.25	27604.90	18602.29
W=.40 ,P=0.75	9.99	798.98	8914.08	25299.29	17469.29	881.73	52481.63	35894.07
W=.50 ,P=0.75	9.97	969.88	10985.92	29709.27	19810.37	1064.49	61475.43	42729.55
W=.75 ,P=0.75	10.03	1365.59	15149.85	41237.62	26614.64	1496.07	84367.69	59249.11
W=1 ,P=0.75	9.99	1393.83	15643.66	43434.20	30490.03	1529.87	90961.72	62001.56
W=.0 ,P=0.9	11.57	156.36	1802.62	6136.79	3691.56	174.44	11787.32	8270.20
W=.05 ,P=0.9	11.24	201.42	2164.08	8276.86	5628.95	230.60	16271.30	10872.95
W=.10 ,P=0.9	11.26	200.38	2162.06	8267.56	5614.20	229.54	16244.19	10859.54
W=.20 ,P=0.9	11.18	212.87	2332.19	8741.60	5908.29	243.80	17194.93	11530.45
W=.30 ,P=0.9	10.57	382.30	4302.23	14070.05	9770.50	430.67	28525.08	19185.24
W=.40 ,P=0.9	9.92	804.86	8939.72	25915.38	17818.54	889.91	53478.49	36549.86
W=.50 ,P=0.9	9.91	986.66	11085.87	30685.80	20380.45	1084.69	63138.77	43843.01
W=.75 ,P=0.9	9.96	1393.48	15354.36	42872.38	27614.10	1529.56	87234.31	61149.77
W=1 ,P=0.9	9.92	1425.33	15857.42	45397.14	31917.60	1567.76	94597.49	64247.64
W=.0 ,P=1.0	11.56	156.87	1799.20	6212.70	3776.50	175.45	11945.26	8344.21
W=.05 ,P=1.0	11.21	204.42	2183.44	8388.55	5750.30	234.17	16526.70	11010.57

W=.10	,P=1.0	11.21	203.93	2182.03	8386.53	5747.98	233.68	16520.46	11006.16
W=.20	,P=1.0	11.10	216.15	2359.77	8852.31	6057.65	247.69	17485.87	11675.91
W=.30	,P=1.0	10.53	381.38	4260.56	14195.28	9897.02	430.55	28734.24	19267.77
W=.40	,P=1.0	9.90	808.92	8976.68	26301.94	18044.87	895.37	54132.40	36982.90
W=.50	,P=1.0	9.87	997.54	11160.70	31178.35	20707.04	1097.28	64043.62	44433.86
W=.75	,P=1.0	9.87	1395.48	15342.72	43284.19	27822.12	1533.11	87844.50	61555.49
W=1	,P=1.0	9.87	1441.17	15959.31	46273.14	32538.61	1586.66	96212.22	65260.26

	KL-2000 - Diameter 4							
	Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
W=.0 ,P=0.0	11.60	440.58	6729.35	3578.70	1043.00	442.03	11791.63	11190.66
W=.05 ,P=0.0	11.54	439.50	6705.27	3533.11	1042.09	440.95	11719.96	11118.82
W=.10 ,P=0.0	11.54	446.48	6788.73	3536.44	1042.09	447.93	11813.74	11219.58
W=.20 ,P=0.0	11.42	563.38	8205.94	3611.99	1042.09	564.83	13423.39	12946.13
W=.30 ,P=0.0	11.29	595.71	8579.85	3635.92	1042.09	597.16	13853.57	13408.63
W=.40 ,P=0.0	11.14	612.92	8736.85	3645.69	1042.09	614.37	14037.55	13609.83
W=.50 ,P=0.0	11.13	619.91	8774.03	3650.51	1036.80	621.36	14081.24	13665.79
W=.75 ,P=0.0	10.86	701.10	9723.67	3724.81	1101.09	702.55	15250.67	14852.13
W=1 ,P=0.0	10.82	706.49	9789.85	3731.17	1110.71	707.94	15338.22	14935.45
W=.0 ,P=0.25	11.21	425.59	6302.73	3837.05	4195.35	430.01	14760.72	10995.38
W=.05 ,P=0.25	10.97	443.13	6291.84	4300.85	9895.28	452.72	20931.10	11488.54
W=.10 ,P=0.25	10.90	444.85	6293.50	4310.24	10018.19	454.50	21066.77	11503.09
W=.20 ,P=0.25	10.46	654.77	8661.90	4926.51	15279.32	668.78	29522.49	14911.96
W=.30 ,P=0.25	10.32	840.06	10588.32	5422.98	19507.31	857.64	36358.67	17709.00
W=.40 ,P=0.25	10.09	938.79	11486.31	5700.54	22034.52	958.45	40160.15	19084.08
W=.50 ,P=0.25	10.17	999.94	12168.13	5762.39	22317.22	1019.73	41247.68	19950.19
W=.75 ,P=0.25	10.16	1164.93	13901.60	6194.15	26287.09	1187.67	47547.76	22448.34
W=1 ,P=0.25	10.23	1179.44	14130.92	6155.21	25971.59	1201.72	47437.16	22667.29
W=.0 ,P=0.5	11.10	415.11	6125.96	3859.61	4649.58	419.99	15050.25	10820.66
W=.05 ,P=0.5	10.90	448.57	6326.22	4522.68	12504.82	460.46	23802.29	11757.93
W=.10 ,P=0.5	10.88	457.94	6438.18	4543.14	12618.37	469.94	24057.63	11909.20
W=.20 ,P=0.5	10.26	651.24	8599.32	5178.06	18255.23	667.95	32683.84	15096.56
W=.30 ,P=0.5	10.17	853.01	10636.50	5789.37	23790.30	874.30	41069.17	18153.18
W=.40 ,P=0.5	10.02	972.13	11698.47	6155.23	27159.69	996.27	45985.51	19822.09
W=.50 ,P=0.5	10.00	1031.06	12312.68	6235.98	27686.20	1055.55	47265.92	20635.26
W=.75 ,P=0.5	10.04	1222.12	14291.99	6836.29	33338.27	1251.11	55688.66	23601.50
W=1 ,P=0.5	10.03	1229.67	14452.09	6812.54	33680.26	1258.40	56174.55	23752.69
W=.0 ,P=0.75	11.05	412.45	6081.57	3892.52	5063.22	417.72	15449.76	10804.26
W=.05 ,P=0.75	10.80	447.05	6252.54	4629.65	13608.57	460.07	24937.81	11789.30
W=.10 ,P=0.75	10.79	452.51	6325.37	4662.65	13945.98	465.81	25386.50	11906.33
W=.20 ,P=0.75	10.22	657.26	8637.56	5322.69	19836.55	675.40	34454.05	15292.90
W=.30 ,P=0.75	10.09	848.59	10513.48	5957.43	25761.73	871.69	43081.22	18191.18
W=.40 ,P=0.75	9.90	976.52	11674.46	6392.58	29722.49	1003.13	48766.04	20046.68
W=.50 ,P=0.75	9.93	1049.82	12406.37	6518.76	30678.79	1077.11	50653.73	21052.06
W=.75 ,P=0.75	9.92	1244.10	14400.78	7183.31	36976.72	1276.58	59804.90	24104.76
W=1 ,P=0.75	9.92	1255.89	14562.38	7176.51	38192.60	1288.22	61187.37	24282.99
W=.0 ,P=0.9	11.00	409.78	6041.87	3890.77	5071.36	415.05	15413.78	10757.46
W=.05 ,P=0.9	10.76	446.52	6245.61	4671.65	14116.87	460.02	25480.65	11823.80
W=.10 ,P=0.9	10.74	453.18	6313.78	4692.98	14264.66	466.82	25724.60	11926.75
W=.20 ,P=0.9	10.26	665.14	8719.71	5416.76	20870.96	684.19	35672.56	15485.79
W=.30 ,P=0.9	10.06	852.12	10529.93	6034.50	26548.22	876.01	43964.76	18292.55
W=.40 ,P=0.9	9.89	983.88	11728.99	6486.18	30701.98	1011.41	49901.02	20210.45
W=.50 ,P=0.9	9.87	1052.87	12408.29	6611.73	31705.79	1081.11	51778.67	21153.99
W=.75 ,P=0.9	9.88	1251.39	14426.21	7313.72	38236.07	1285.18	61227.39	24276.49
W=1 ,P=0.9	9.89	1272.87	14659.54	7373.46	40414.20	1307.11	63720.06	24612.97
W=.0 ,P=1.0	11.01	411.92	6072.68	3898.39	5134.70	417.24	15517.68	10800.22
W=.05 ,P=1.0	10.76	447.74	6251.20	4691.21	14297.15	461.43	25687.29	11851.56

W=.10 ,P=1.0	10.71	453.57	6316.19	4711.89	14473.15	467.40	25954.79	11949.04
W=.20 ,P=1.0	10.20	663.03	8704.41	5424.10	20987.70	682.19	35779.23	15473.73
W=.30 ,P=1.0	10.02	848.56	10479.42	6055.54	26868.36	872.71	44251.88	18256.23
W=.40 ,P=1.0	9.87	984.03	11715.17	6526.59	31113.34	1011.98	50339.12	20237.76
W=.50 ,P=1.0	9.87	1058.51	12441.73	6673.98	32242.62	1087.35	52416.83	21261.56
W=.75 ,P=1.0	9.87	1259.17	14474.31	7395.99	39045.89	1293.76	62175.36	24423.23
W=1 ,P=1.0	9.87	1277.32	14684.86	7441.38	41223.58	1312.23	64627.13	24715.78

C.4 FOURTH EXPERIMENT

		Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Level)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
		Blocks World							
V3	D = 2	9.92	526.83	2553.81	48588.75	3365.64	780.20	55035.02	52449.58
	D = 4	9.92	383.61	1448.82	12581.53	4657.07	458.10	19071.03	14872.06
V4	D = 2	9.92	386.52	1992.94	29041.83	4384.94	555.38	35806.23	31976.67
	D = 4	9.92	333.85	1287.82	9797.70	6174.90	391.82	17594.25	11811.18
V5	D = 2	9.92	480.04	2740.86	33780.49	5715.70	724.89	42717.08	37726.27
	D = 4	9.92	316.32	1158.91	9264.28	5581.50	371.97	16321.01	11111.47
		Permutation							
V3	D = 2	4.71	332.20	4089.56	22811.85	1869.62	454.27	29103.22	27687.87
	D = 4	4.71	168.57	1374.45	3807.26	2809.17	200.01	8159.44	5550.28
V4	D = 2	4.71	233.41	3054.87	13352.92	6908.64	317.59	23549.84	16958.79
	D = 4	4.71	168.41	1371.95	3803.42	7732.10	199.85	13075.87	5543.62
V5	D = 2	4.71	226.34	3070.57	13970.07	6845.38	320.62	24112.35	17587.59
	D = 4	4.71	167.72	1344.99	3823.70	7274.64	199.58	12611.05	5535.99
		5-puzzle							
V3	D = 2	19.29	881.22	2762.55	29708.41	3291.64	1037.77	36643.80	34389.94
	D = 4	19.29	489.92	1533.77	5832.23	2274.44	533.68	10130.36	8389.59
V4	D = 2	19.29	821.47	2573.39	25904.10	5175.11	959.75	34474.07	30258.71
	D = 4	19.29	470.04	1467.51	5395.02	4226.02	510.26	11558.59	7842.83
V5	D = 2	19.29	850.77	2895.83	26212.71	5725.59	1024.53	35684.90	30983.84
	D = 4	19.29	432.59	1314.16	4897.15	3645.92	470.53	10289.81	7114.43
		T.O.H							
V3	D = 2	67.35	4649.96	13910.51	91407.27	14811.18	4851.87	124778.92	114819.61
	D = 4	67.35	1909.64	5707.77	16095.01	9140.49	1970.46	32852.90	25682.87
V4	D = 2	67.35	4488.80	13428.76	88327.62	26602.86	4684.80	132848.04	110929.97
	D = 4	67.35	1865.20	5575.19	15733.52	18566.51	1924.60	41740.41	25098.51
V5	D = 2	67.35	4337.11	12958.83	76885.35	25594.16	4552.74	119775.44	98734.02
	D = 4	67.35	1683.60	5030.96	13750.92	15819.83	1738.16	36285.30	22203.63
		KL-2000							
V3	D = 2	9.87	1630.59	24018.41	252847.29	10059.06	2048.23	288555.34	280544.51
	D = 4	9.87	1083.99	12296.71	19821.41	15586.53	1163.56	48788.64	34365.67
V4	D = 2	9.87	1294.52	19127.53	162406.62	22770.79	1588.15	205599.44	184416.81
	D = 4	9.87	1029.50	11820.36	17990.73	23208.20	1100.57	54048.78	31941.16
V5	D = 2	9.87	1296.37	18908.94	149421.95	21564.78	1613.22	191192.04	171240.48
	D = 4	9.87	1004.53	11591.61	16288.10	19460.63	1068.88	48344.86	29953.11

C.5 FIFTH EXPERIMENT

		Solution Length	Nodes Expanded	Arcs Traversed	Caching (Current Levels)	Marking Down	Looking Up	Work (Marking Down)	Work (Looking Up)
					Blocks World				
V3	D = 2	47.46	595.37	2319.74	22839.75	6503.48	707.59	32258.33	26462.44
	D = 4	47.46	404.54	1409.29	8255.29	8305.53	460.32	18374.65	10529.44
V4	D = 2	47.46	561.19	2205.31	20130.38	9208.93	661.99	32105.81	23558.87
	D = 4	47.46	399.04	1391.41	7748.46	9852.43	450.72	19391.33	9989.62
V5	D = 2	47.46	485.48	1857.99	18257.81	8102.45	580.66	28703.72	21181.93
	D = 4	47.46	386.54	1298.16	7194.86	8439.35	434.00	17318.89	9313.55
					Permutation				
V3	D = 2	19.94	465.57	6185.24	12650.22	2465.94	537.86	21766.96	19838.88
	D = 4	19.94	292.34	1670.83	2405.34	5549.16	317.14	9917.65	4685.64
V4	D = 2	19.94	441.05	5775.52	11081.89	11761.68	504.02	29060.13	17802.47
	D = 4	19.94	292.18	1669.57	2399.93	8168.58	316.86	12530.26	4678.54
V5	D = 2	19.94	413.62	5253.20	10567.15	10315.29	477.62	26549.25	16711.58
	D = 4	19.94	291.14	1644.69	2395.14	7526.96	316.03	11857.93	4647.00
					5-puzzle				
V3	D = 2	101.91	717.57	2220.10	12847.14	3126.05	795.89	18910.85	16580.69
	D = 4	101.91	469.49	1407.53	4085.82	2247.48	501.56	8210.31	6464.40
V4	D = 2	101.91	707.15	2182.91	12340.15	5575.05	782.32	20805.25	16012.52
	D = 4	101.91	466.63	1395.87	4015.45	4464.63	497.98	10342.58	6375.93
V5	D = 2	101.91	645.70	1986.02	11007.56	4911.07	718.50	18550.34	14357.77
	D = 4	101.91	432.33	1250.50	3583.73	3582.26	459.84	8848.81	5726.39
					T.O.H				
V3	D = 2	356.15	3708.53	11411.33	61024.17	15479.89	3860.36	91623.92	80004.38
	D = 4	356.15	1834.79	5582.32	14120.43	9754.72	1890.32	31292.26	23427.86
V4	D = 2	356.15	3655.44	11245.66	59812.39	27297.00	3804.26	102010.48	78517.74
	D = 4	356.15	1826.56	5557.14	14002.24	18827.37	1881.51	40213.30	23267.44
V5	D = 2	356.15	3355.13	10284.41	50140.92	24595.02	3496.45	88375.48	67276.91
	D = 4	356.15	1670.23	5070.08	12254.83	16075.10	1720.24	35070.23	20715.37
					KL-2000				
V3	D = 2	31.03	1503.18	16731.74	47681.44	24358.48	1650.08	90274.83	67566.43
	D = 4	31.03	1239.62	15033.37	7612.34	38260.86	1276.29	62146.17	25161.61
V4	D = 2	31.03	1476.18	16547.91	44936.75	33462.01	1614.58	96422.85	64575.42
	D = 4	31.03	1238.21	15029.04	7541.62	40549.58	1274.14	64358.44	25083.00
V5	D = 2	31.03	1415.52	15993.87	37982.13	27747.46	1536.55	83138.96	56928.05
	D = 4	31.03	1228.43	14978.96	7231.58	35703.40	1261.07	59142.36	24700.03