National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services Branch

Direction des acquisitions et des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

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.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

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.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canadä

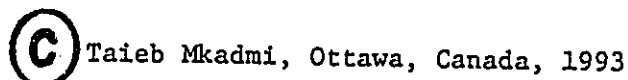
# SPEEDING-UP STATE-SPACE SEARCH BY AUTOMATIC ABSTRACTION

*By*

Taieb Mkadmi

Thesis Submitted
to the School of Graduate Studies
in partial fulfilment of the requirements
for the Master degree in Computer Science
under the auspices of the Ottawa-Carleton
Institute for Computer Science

UNIVERSITY OF / UNIVERSITÉ D'
OTTAWA


© Taieb Mkadmi, Ottawa, Canada, 1993

National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services Branch

Direction des acquisitions et des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-83830-2

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

*To my lovely mother*

*Mannoubia*

# Abstract

Most existing abstraction algorithms are sensitive to the initial problem formulation. Given two different descriptions of the same space, they will produce different abstractions, of which one might be efficient for problem-solving while the other might be inefficient.

This thesis presents a completely automated approach to generating and using abstractions for problem solving in state-spaces. The strategy to overcome the problem of sensitivity is called the *graph relabelling strategy*. The abstraction algorithms used are all based on that strategy and on a theoretical study of the complexity to abstract and to search using an abstraction. This study presents theorems and compares analytical results to some known graph algorithms.

Extensive experiments confirm that our abstractions can be quickly computed and greatly reduce problem-solving time in state-spaces, especially those with invertible operators.

# Acknowledgments

I am very grateful to my supervisor Dr. Robert C. Holte. He was always a good source of ideas and insights. I thank him so much for the plenty of time he gave me, for his "exaggerated" patience and for his tremendous support. I am really proud to be his student. I would like to thank Dr. Nejib Zaguia for the fruitful discussions we had and for his continuous encouragement.

I also would like to express my gratitude to my country Tunisia which provided me the opportunity to come and study in Canada. I gratefully acknowledge the financial support by the Natural Sciences and Engineering Research Council of Canada.

Finally, I thank my family and all my friends for their endless love and support.

# Table Of Contents

**Chapter 1**

**Introduction** .......... **1**

1.1 Search and efficiency .......... 1

1.1.1 State-space .......... 2

1.2 Speedup of state space search .......... 5

1.3 Specification of this work .......... 9

1.3.1 Contributions .......... 9

1.3.2 Organization of the thesis .......... 10

**Chapter 2**

**Definitions** .......... **11**

2.1 Basic definitions of state-space .......... 11

2.2 Abstraction .......... 14

2.3 Abstraction: a kind of change of representation .......... 14

2.4 Various useful definitions .......... 19

2.4.1 Partition of a set .......... 19

2.4.2 Equivalence relation .......... 20

2.4.3 Isomorphism .......... 21

2.4.4 Homomorphism .......... 22

2.4.5 Congruence .......... 23

**Chapter 3**

**Problem-solving by Abstraction/Refinement** . . . . . . . . . . 25

3.1 Abstraction = Homomorphism . . . . . . . . . . 26

3.2 The use of abstraction for problem solving . . . . . . . . . . 27

3.2.1 Refinement and refinability . . . . . . . . . . 27

3.2.2 Searching . . . . . . . . . . 31

3.3 Initial algorithms and their deficiencies . . . . . . . . . . 33

3.3.1 The class extension algorithm . . . . . . . . . . 33

3.3.2 The chain extension algorithm . . . . . . . . . . 36

3.3.3 Deficiencies of initial algorithms . . . . . . . . . . 38

3.4 Solution: graph relabelling . . . . . . . . . . 41

3.4.1 Formal discussion of graph relabelling . . . . . . . . . . 42

**Chapter 4**

**Complexity analysis of using abstraction** . . . . . . . . . . 44

4.1 Analysis of problem-solving efficiency . . . . . . . . . . 44

4.1.1 Examples of the "work" w . . . . . . . . . . 51

4.1.1.1 Example 1 . . . . . . . . . . 51

4.1.1.2 Example 2 . . . . . . . . . . 52

4.1.2 Discussion of the "work" w . . . . . . . . . . 53

4.1.3 Examples . . . . . . . . . . 54

4.2 General approach to abstract . . . . . . . . . . 55

4.3 Complexity to abstract and store . . . . . . . . . . 56

4.3.1 Complexity to abstract . . . . . . . . . . 56

4.3.2 Complexity to store . . . . . . . . . . 58

**Chapter 5**

**Specific algorithms** . . . . . . . . . . 60

5.1 Introduction . . . . . . . . . . 60

5.2 The primitive inverse case . . . . . . . . . . 63

5.2.1 The *Algorithm ........ 64
5.2..1.1 Complexity analysis ........ 65
5.2.1.2 Searching and storing the result when using the *Algorithm ........ 68
5.2.2 The Absorb *Algorithm ........ 68
5.2.2.1 Complexity analysis ........ 69
5.2.3 The double *Algorithm ........ 70
5.2.3.1 Complexity analysis ........ 72
5.3 The general case ........ 73
5.3.1 The general approach for the general case ........ 73
5.3.2 The strong *Algorithms ........ 74
5.3.3 The strong edge algorithms ........ 77

**Chapter 6**

**Experimental work** ........ 78
6.1 Motivation ........ 78
6.2 Parameters measured and their significance ........ 79
6.3 Results and discussion ........ 81
6.3.1 The primitive inverse case ........ 81
6.3.2 The general case ........ 91
6.4 Conclusion ........ 96

**Chapter 7**

**Literature review** ........ 97
7.1 General Approaches dealing with abstraction ........ 97
7.1.1 The use of abstractions for problem solving ........ 97
7.1.2 The generation of abstractions for problem solving ........ 99
7.2 Closely related work ........ 101
7.3 Conclusion ........ 103

**Chapter 8**
**Conclusion** .......... **105**
8.1 Summary of the thesis .......... 105
8.2 Limitations and weaknesses .......... 106
8.3 Future work .......... 107
8.3.1 Work for the near future .......... 107
8.3.2 Work for the distant future .......... 108

**Appendix A**
**Puzzles** .......... **109**
A.1 Towers of Hanoi .......... 109
A.2 The Navigation Problem .......... 109
A.3 The 5-Puzzle .......... 110
A.4 The Arrow Puzzle .......... 111
A.5 The Missionaries and Cannibals .......... 111
A.6 The Water Jug Problem .......... 113

**Appendix B**
**Tabular results** .......... **114**
B.1 The Prim Inv Towers of Hanoi .......... 114
B.2 The Arrow puzzle .......... 116
B.3 The 5-Puzzle .......... 118
B.4 The Prim Inv Navigation Problem .......... 118
B.5 The Missionaries and Cannibals .......... 121
B.6 The Non-Prim Inv Navigation Problem .......... 122
B.7 The Non-Prim Inv Towers of Hanoi .......... 124
B.8 The Water Jug Problem .......... 125

**Appendix C**

**Raw Program Output** .......... **127**

C.1 The Prim Inv Towers of Hanoi .......... 127

C.2 The Arrow Puzzle .......... 130

C.3 The 5-Puzzle .......... 131

C.4 The Non-Prim Inv Navigation Problem .......... 133

C.5 The Missionaries and Cannibals .......... 135

C.6 The Navigation Problem .......... 136

C.7 The Non-Prim Inv Towers of Hanoi .......... 137

C.8 The Water Jug Problem .......... 139

**Bibliography** .......... **141**

# List of Figures

Figure 1.1 The navigation problem . . . . . . . . . . 2
Figure 1.2 A portion of the graph for the navigation problem state space . . . . . . . . . . 3
Figure 1.3 The start and goal states . . . . . . . . . . 4
Figure 1.4 The abstract space for the navigation problem . . . . . . . . . . 7
Figure 2.1 Initial and Goal states in the Towers of Hanoi (TOH3) . . . . . . . . . . 12
Figure 2.2 State space of the 2-disk Tower of Hanoi . . . . . . . . . . 13
Figure 2.3 An adequate representation . . . . . . . . . . 17
Figure 2.4 An inadequate representation . . . . . . . . . . 17
Figure 2.5 Correspondence between levels of abstractions . . . . . . . . . . 18
Figure 3.1 The standard abstraction of the 3-disk Tower of Hanoi . . . . . . . . . . 28
Figure 3.2 TOH2 . . . . . . . . . . 30
Figure 3.3 Refinement . . . . . . . . . . 31
Figure 3.4 Hierarchical search . . . . . . . . . . 32
Figure 3.5 The labelled TOH2 . . . . . . . . . . 34
Figure 3.6 An example of abstraction of TOH2 . . . . . . . . . . 35
Figure 3.7 An example of abstraction on TOH2 . . . . . . . . . . 36
Figure 3.8 The chains in a state space . . . . . . . . . . 38
Figure 3.9 An example of a graph . . . . . . . . . . 39
Figure 3.10 Abstraction of the previous space . . . . . . . . . . 39
Figure 3.11 The standard abstraction of TOH2 . . . . . . . . . . 40
Figure 4.1 The notion of diameter . . . . . . . . . . 45
Figure 4.2 The essential parameters in Standard abstraction of TOH2 . . . . . . . . . . 46

Figure 4.3 The essential parameters in a particular abstraction of TOH2 . . . . . . . . . . . . . 47
Figure 4.4 The process of refining . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
Figure 4.5 The essential parameters in Standard abstraction of TOH2 . . . . . . . . . . . . . . 52
Figure 4.6 The essential parameters in a particular abstraction of TOH2 . . . . . . . . . . . . . 53
Figure 4.7 A line . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
Figure 5.1 Primitive inverse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
Figure 5.2 Non primitive inverse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
Figure 5.3 The strongly connected algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
Figure 5.4 The abstract space of TOH2 using the *Algorithm . . . . . . . . . . . . . . . . . . . 64
Figure 5.5 The adjacency matrix . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
Figure 5.6 The "inside rectangles" are only visited once . . . . . . . . . . . . . . . . . . . . . . . 66
Figure 5.7 The "absorb" phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
Figure 5.8 The abstraction of TOH2 using "Absorb *Algorithm" . . . . . . . . . . . . . . . . . . 70
Figure 5.9 The "Double *Algorithm" strategy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
Figure 5.10 The abstraction of TOH2 using the "Double *Algorithm" . . . . . . . . . . . . . . 72
Figure 5.11 The non primitive inverse TOH2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
Figure 6.1 Number of arcs traversed vs Disk number (1) . . . . . . . . . . . . . . . . . . . . . . . 84
Figure 6.2 Number of arcs traversed vs Disk number (2) . . . . . . . . . . . . . . . . . . . . . . . 84
Figure 6.3 Search time vs Disk number (1) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
Figure 6.4 Search time vs Disk number (2) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86
Figure 6.5 Abstraction time vs Disk number (1) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
Figure 6.6 Abstraction time vs Disk number (2) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
Figure 6.7 Average solution length vs Disk number (1) . . . . . . . . . . . . . . . . . . . . . . . . 88
Figure 6.8 Average solution length vs Disk number (2) . . . . . . . . . . . . . . . . . . . . . . . . 89
Figure 6.9 Ratio of the solution length by the optimal solution length . . . . . . . . . . . . . . 89
Figure 6.10 Number of arcs traversed vs Disk number . . . . . . . . . . . . . . . . . . . . . . . . . 93
Figure 6.11 Search time vs Disk number . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
Figure 6.12 Abstraction time vs Disk number . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95

# Chapter 1

# Introduction

## 1.1 Search and efficiency

The concept of search is central to AI as well as to other areas of computer science. Search techniques are of central importance and significance to AI not only because most AI systems are built around them, but because it is through deep understanding and exploration of search algorithms that we can foresee and predict what kinds of AI problems are solvable in practice. Search problems include

- Games and puzzles
- Route finding
- Language parsing and interpretation
- Logic programming
- Computer vision and pattern recognition
- Expert systems
- Machine learning

### 1.1.1 State-space

The idea of searching for something implies moving around examining things and making decisions about whether the sought object has yet been found [Tanimoto, 1990]. State-space is a way of representing or depicting the information defining a problem. It shows a "network of information" of a certain problem. A state is a vector or set of measurements of objects having certain properties. For example the *configuration* of a chess board is a state. A configuration could be described, for instance, as a vector of 64 elements where each element includes information like the coordinates of the correspon-ding position in the board, what kind of a chess piece is on it and whether it is white or black, etc... After each move of one player, we get a new configuration or a new state. State-space is normally depicted as a graph where nodes correspond to states and arcs correspond to transitions or moves that go from one state to another. These transitions are also known as operators (e.g. move the black queen from b7 to b1 on the board).

**Example** The navigation problem

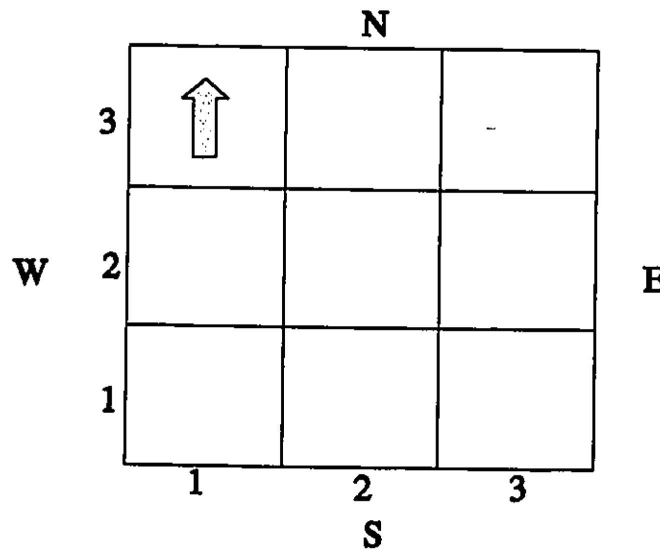



**Figure 1.1** The navigation problem

Suppose in a 3×3 board, we have a robot that has two actions:

Action M = move forward
Action R = rotate 90 degrees counterclockwise.

A state is the configuration of the board at any time. It contains the coordinates of the robot's location and the direction it faces. Figure 1.1 represents the state [1,3,N] because the robot is at position (1,3) in the board, and faces North. All possible states are all possible combinations of location and direction. In all, there (3*3)*4 = 36 states. Figure 1.2 shows a portion of the state-space. □

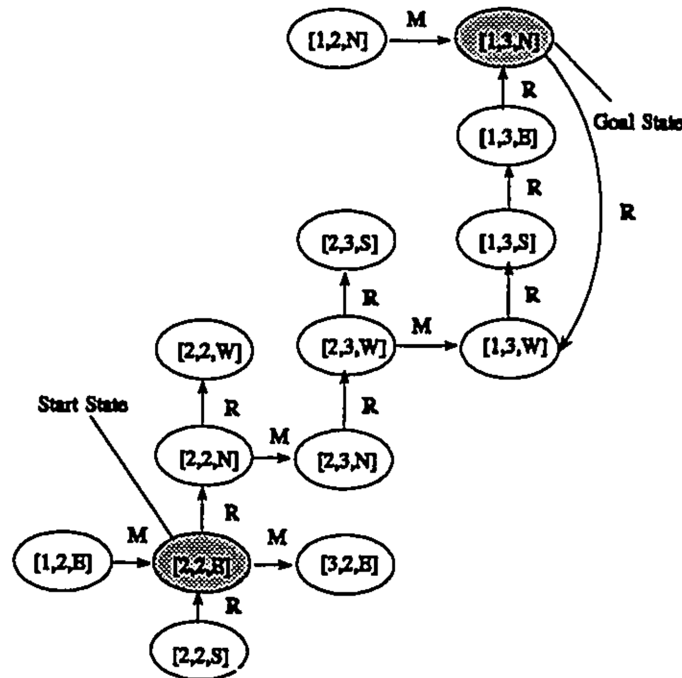



**Figure 1.2** A portion of the graph for the navigation problem state space

A *search problem* is defined by an initial (or start) state and a goal state. To solve the problem is to find a set of operators that, when applied to the initial state, leads to the goal state. In the example mentioned above, if the initial state is [2,2,E] and the goal state is [1,3,N] (see Figure 1.3), then a possible solution would be:

R M R M R R R.

which means: rotate the robot once: get to [2,2,N], move forward once: [2,3,N], rotate once: [2,3,W], move again: [1,3,W], and then rotate three times: [1,3,S] then [1,3,E] and finally [1,3,N]. The solution could be written also:

[2,2,E] R [2,2,N] M [2,3,N] R [2,3,W] M [1,3,W] R [1,3,S] R [1,3,E] R [1,3,N].

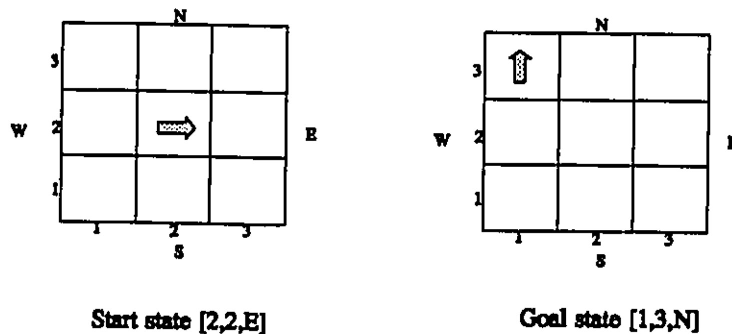



**Figure 1.3** The start and goal states

□

It is mandatory to choose or to "invent" the right search method for a problem because a poor choice could be so inefficient as to be infeasible.

**Example**

Suppose we take the example of the navigation problem. We want to find a path from some initial state, say S, to some given goal state G. One way of doing this (the most naive way) is to generate all possible paths and then select one among them. This is known as the *British Museum Procedure.* At each node in the graph we have 2 alternatives to choose: either apply R or M (if possible). If we construct a tree by applying the algorithm, the tree will have a root S and the first level of the tree will have 2 nodes, the second will have $2^2$ and etc... This means that at depth d, the number of nodes is $2^d$. Using the navigation problem in a larger board and more operators (e.g. rotating both counterclockwise and clockwise) would even worsen the number. In general, in a small graph, we can have the example of b = 10 and d = 12, the number of nodes will be then $10^{12}$, i.e one thousand billion nodes to search through and to store !

□

Various strategies and techniques for effective search have emerged from the fields of mathematics and computer science. The main contributions of AI are the concept of *heuristic* and *control knowledge*, techniques for improving the efficiency of search. As an example of control knowledge, the navigation problem can be directly solved by subtracting the X-coordinates and the Y-coordinates of the start state from the goal state to get all the M's to apply (plus one R), and finally "subtracting the angles" of the start state from the goal state to get the R's. Notice, that no search was involved.

## 1.2 Speedup of state space search

Most AI approaches to speeding up search are "manual". By contrast, the approach taken in this thesis is to automatically change the search space in order to increase speed.

Solutions are found more efficiently in the new space than in the original one. The efficiency of search is measured by the number of operators applied in the course of solving a problem. In the worst case, one blindly moves through states.

In a manual approach, the state-space and method of search are created by humans. The search system uses them in problem-solving, but is unable to change or create its search space and method. This is, for example, the case of GPS [Newell and Simon, 1972]. On the other hand some systems (e.g ABSTRIPS [Sacerdoti, 1977]) change state spaces automatically. In the case of ABSTRIPS, it automatically produces a three-level "abstraction hierarchy" and further levels come from the user. This will be discussed further in Chapter 7.

A general strategy for automatically increasing the speed of search is to conduct search in some space other than the original. Searching in a state-space directly is replaced by three different stages:

- Mapping the original state-space (OSS) to a new state-space (NSS)
- Problem-solving in NSS
- Translating the solution into OSS

**Example** The navigation problem: nav33 (33 means that the board is 3×3).

Suppose we want to solve the same problem seen previously, which is to find a *path* from the start state [2,2,E] to the goal state [1,3,N]. Solving in OSS is searching in the original space using any basic search algorithm. One way of abstracting OSS is to consider all the states with the same coordinates (but possibly different directions) as one

state in the abstract space[1].

In this example the 3 stages will be:

- Mapping OSS to NSS: collect all the states with the same coordinates. Notice that this will cause R to be the identity operation on all the states of NSS (Figure 1.4).
- Solving in NSS: Use the same search technique as was used in OSS.
- Translating the solution back to OSS: *Refining* the solution obtained in NSS. Refining will be explained in detail in Chapter 3. Briefly, it means fleshing out the solution at the abstract level so that it becomes a solution in the original space.

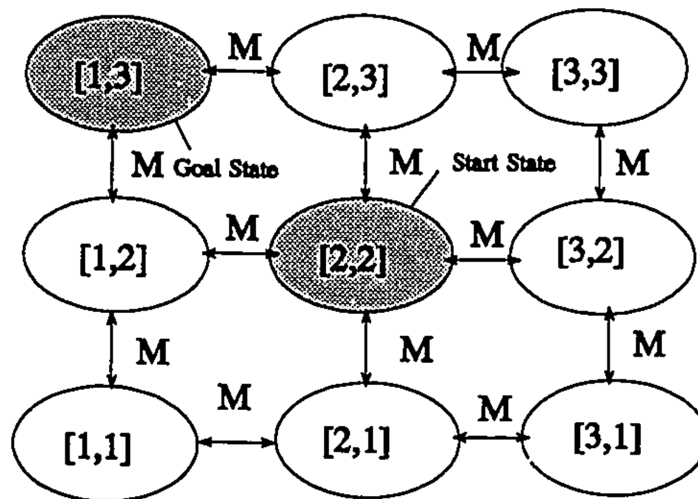



**Figure 1.4** The abstract space for the navigation problem

At the abstract level NSS, a solution is:

[1] This means we ignore direction.

[2,2] --M--> [2,3] --M--> [1,3]

This is not a solution in the original space (applying M twice to the start state). If we apply M to [2,2,E] we get to [3,2,E] where M cannot be applied. However, it can be made into a solution by inserting zero or more R's between each M move:

At the original level, it becomes:

[2,2,E] --R--> [2,2,N] --M--> [2,3,N] --R--> [2,3,W] --M--> [1,3,W] --R--> [1,3,5]
--R--> [1,3,E] --R--> [1,3,N].

□

Recall the strategy to abstract:

- Mapping the original state-space (OSS) to a new state-space (NSS)
- Solving in NSS
- Translating back the solution in OSS

We have to note here that the three stages mentioned do not necessarily reflect the number of actual stages to be done in the whole process of solving the problem. We mean by this the possibility of having a *hierarchy* that preserves the same structure. To solve NSS itself we may need to replace it by three more stages, exactly as we did for OSS. In this way, we can reach a situation or a level where solving in NSS is trivial because of its size. This will be clarified in the next chapter.

## 1.3 Specification of this work

The goal of this thesis is to present a strategy for abstraction and prove its usefulness for automatically speeding up search in state-spaces. Given a graph, we "abstract" it until a "very simple" space is found. This process is done only once and is "problem independent". This means that after doing so, we can solve the problem for any given "start" and "goal" states. The abstraction of a state-space is a hierarchy of state-spaces, and the solution to a given problem is found through that hierarchy and translated back to the original state-space. Our overall aims are:

- The construction of the abstraction hierarchy should be efficient
- Solving a problem instance using the hierarchy should be faster on average than solving the problem in the original space.
- The solutions found using the hierarchy may be longer than the solutions found in the original space, but they should not be "excessively" large.
- Storing the hierarchy in the memory should be efficient.

A final important aim is for our abstraction algorithm to be insensitive to the exact manner in which the original space is described. Most existing algorithms are sensitive. Given two different descriptions of the same space, they will produce different abstractions, one may be very good, one may be bad. This issue will be discussed in detail in Chapter 3 (section 3.3.3).

### 1.3.1 Contributions

The primary contributions of the thesis are the discovery of the graph relabelling strategy, the theorems related to the complexity analysis, the abstraction algorithms and their complexities (time and space) and finally the implementation and the empirical

demonstration of the automatic abstraction generation and its use.

The graph relabelling strategy is our "key" to overcome the sensitivity problem. The theorems show how efficiently an abstraction hierarchy can be created and present the complexity of using an abstraction. The abstraction algorithms are variations on one main algorithm called the *algorithm (Chapter 5) and their complexities for both time and space are proved to be of order $N^2$, where N is the number of nodes in the graph. The algorithms have all been implemented in the functional language ML, and used in an extensive set of experiments to confirm their ability to speedup search.

### 1.3.2 Organization of the thesis

Definitions of the basic elements of state-space and problem solving as well as some useful algebraic definitions are given in Chapter 2. These definitions are used to explain our approach to problem-solving. Chapter 3 describes in detail our approach to problem-solving. It covers three major topics: How we abstract, how we search and the initial algorithms we used. Chapter 4 describes a general approach to abstraction and analyzes its complexity. Chapter 5 presents the main algorithms and theorems. Two separate cases are studied: the "primitive inverse case" and the general case. The primitive inverse case concerns undirected graphs, whereas the general case deals with directed graphs. Chapter 6 describes experiments on several standard AI puzzles. These puzzles include the Towers of Hanoi, the navigation problem, the arrow puzzle, the missionaries and cannibals, the 5-puzzle and the water jug problem. The experiments are needed to confirm the prediction of the theory and also to guide us in improving our algorithms. Chapter 7 presents a survey of related work. Finally the conclusion, in Chapter 8, presents the limitations and weaknesses of our approach as well as some "horizons" open to future work.

# Chapter 2

# Definitions

## 2.1 Basic definitions of state-space

A state-space is a directed graph. Each node corresponds to a particular world state, and each edge to a particular operator which transforms a state into another. Both nodes and edges may be labelled. We disregard node labels. Operators are state to state functions that describe state changes. A problem is a pair of states, <start,goal>. When a sequence of operators is applied to the start state and results in the goal state, it is called a solution plan for the problem. Problem-solving is therefore equivalent to path-finding in a state-space graph.

**Example** The Towers of Hanoi problem.

This puzzle requires moving a pile of various-sized disks from one peg to another with the use of an intermediate peg. Only one disk at a time can be moved, a disk can only be moved if it is the top disk on a peg, and a larger disk can never be placed on a smaller one. An example of initial and goal states of the three-disk puzzle is shown in Figure 2.1. There are many ways to describe the states and moves of this problem. We adopt the following classical one: a state is represented by a list of N digits (1,2 or 3)

such that the $i^{th}$ position of the list indicates the peg on which is sitting the $i^{th}$ smallest disk. The smallest disk is 1, and the biggest disk is N.

The pegs are ordered: 1, 2 and 3. There are 2N operators: -N,...,-1,1,..,N. Applying the operator i means: "move disk i clockwise", and applying -i means "move disk i counterclockwise", where i is positive. If disk 2 is on peg 1, then apply -2 means move it to peg 3. The graph representation of the state-space for the 2-disk Towers of Hanoi (TOH2) is given in Figure 2.2. In the figures the states are not represented with a list, they are, instead, represented by the sequence of list elements (e.g. [1,1] is abbreviated 11).

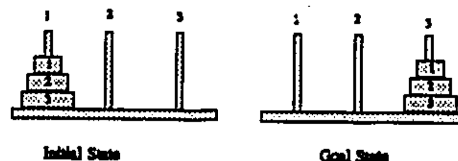


**Figure 2.1** Initial and Goal states in the Towers of Hanoi (TOH3)

□

The following definitions of the basic elements of Problem Solving are brought from [Holte et al, 1992]. Only the definition of "solution" is not standard.

- A **state** is an atomic object.
- An **action** or **operator** is a partial function mapping states to states.
- A **plan** or **path** is a sequence of actions.
- A **problem** is a pair consisting of an initial state and a goal state.
- A **state-space** is a pair consisting of a set of states and a set of actions.

- A **solution** (of a problem <$S_0$,$S_n$>) is a sequence $S_0$-$A_1$-$S_1$-$A_2$-...-$S_{(n-1)}$-$A_n$-$S_n$ Where $S_i$ is a state and $A_i$ is an action mapping $S_{(i-1)}$ to $S_i$.
- A **solution path** (of a problem <S,G>) is a sequence of actions mapping S to G.

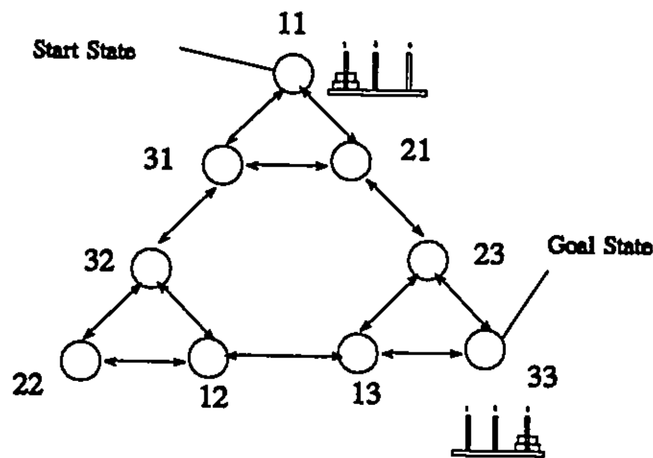



**Figure 2.2** State space of the 2-disk Tower of Hanoi

**Example:** The 2-disk Tower of Hanoi (TOH2)

State-Space =

({[1,1],[3,1],[2,1],[3,2],[2,2],[1,2],[2,3],[1,3],[3,3]}, {-2,-1,1,2})

Let S = [1,1]

G = [3,3] (see Figure 2.2)

Then one solution could be = [1,1] --1--> [2,1] --(-2)--> [2,3] --1--> [3,3]

## 2.2 Abstraction

Abstraction is just a special case of change of representation (see section 2.3). According to [Giunchiglia-Walsh, 1992], abstraction is defined as follows:

1. The process of mapping a representation of a problem, called the "*ground*" representation, onto a new representation , called the "abstract" representation, which:
2. helps deal with the problem in the original search space by preserving certain desirable properties and
3. is simpler to handle as it is constructed from the ground representation by "throwing away details".

In our terminology, the term "ground" representation is the original representation. The properties we preserve are mainly that states get mapped to states and actions to actions, which, also, means that a state-space gets mapped to a state-space. The details we "throw away" are mainly the actions' names or in other words the labelling of the graph. An example was given in chapter 1 where "throwing away" details was by ignoring the direction of the robot in the abstract space.

## 2.3 Abstraction: a kind of change of representation

The following definition is from [Holte and Zimmer, 1989]. In general, a representation is a relation between two *domains*. A domain consists of entities and functions (partial or total). If $D_1 = \langle E_1,F_1 \rangle$ and $D_2=\langle E_2,F_2 \rangle$ are domains, then $\langle R_E,R_F \rangle$ is a representation of $D_1$ by $D_2$ if the following conditions hold:

(1) $R_E$ is a relation between the entities in $E_1$ and $E_2$, and $R_F$ is a relation between the functions in $F_1$ and $F_2$. i.e. $R_E \subseteq E_1 \times E_2$ and $R_F \subseteq F_1 \times F_2$. Define $E_R = \{e_1 \in E_1 \mid \exists\, e_2 \in E_2\ (R_E\ e_1\ e_2)\}$ and $F_R = \{f_1 \in F_1 \mid \exists\, f_2 \in F_2\ (R_F\ f_1\ f_2)\}$.

(2) The fidelity requirement.

$\forall\ f \in F_R\ \forall\ x \in E_R\ \forall\ g \in F_2\ \forall\ y \in E_2$:

$(R_F\ f\ g)\ \&\ (R_E\ x\ y)\ \&\ (f\ x)$ exists $\Rightarrow (g\ y)$ exists $\&\ (R_E\ (f\ x)\ (g\ y))$

(3) A "consistent interpretation" requirement: if $e_1$ and $e_2$ are "co-represented", then every Z that represents $e_1$ must also represent $e_2$.

$\forall e_1 \in E_1\ \forall e_2 \in E_1\ (\exists z\ (R_E\ e_1\ z)\ \&\ (R_E\ e_2\ z)) \Rightarrow (\forall z\ (R_E\ e_1\ z) \Leftrightarrow (R_E\ e_2\ z))$,

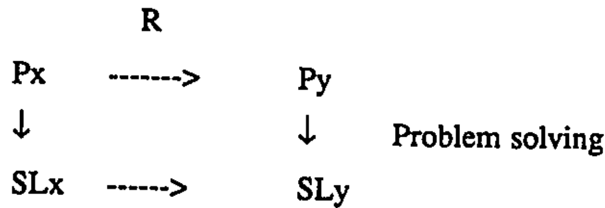
and similarly for functions,

$\forall f_1 \in F_1\ \forall f_2 \in F_1\ (\exists z\ (R_F\ f_1\ z)\ \&\ (R_F\ f_2\ z)) \Rightarrow (\forall z\ (R_F\ f_1\ z) \Leftrightarrow (R_F\ f_2\ z))$.

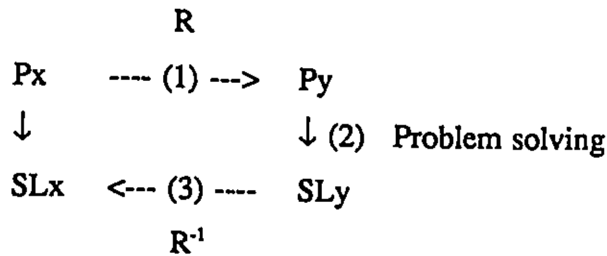
□

In this thesis, we are interested in state-space representation where the original state-space OSS is represented by a new state-space NSS.

Let the domains X and Y be state-spaces. From the previous definition of a state-space, X = {Sx, Ax} and Y = {Sy, Ay}, where Sx (respectively Sy) is the set of states of X (respectively Y) and Ax (resp Ay) is the set of actions (or operators) of X (resp Y). In this case, entities are the states and functions are the actions. Y is a representation of X if there is a mapping R such that if, for any problem Px = <Ix, Gx> in X, R maps Px to some Py = <Iy, Gy> in Y and if SLx is a solution (plan) of Px then there exists SLy a solution (plan) in Py such that R maps SLx to SLy. Let SLx = $A_1$-$A_2$-...-$A_n$ and SLy = $B_1$-$B_2$-...-$B_m$, this means that for each $A_i$ ($1 \leq i \leq n$) $\exists$ $B_j$ ($1 \leq j \leq m$) such that R maps $A_i$ to $B_j$ ($B_j$ may be the identity operation). A particular case, which we treat, is to have $m \leq n$. This is shown in the following diagram.




What we do, practically, is to find R, apply it to X, get Y. For any problem Px in X, apply R to Px, get Py, then solve Py in Y, get SLy and finally apply $R^{-1}$ (this inverse mapping is called refinement, to be discussed in Chapter 3) to get the solution of Px in X called SLx.




Rephrasing that differently:

(1). Translating a given problem Px into another problem in Y

(2). Solving Py (solution is SLy)

(3). Translating SLy back into X (solution is SLx)

The process of translating SLy to SLx is called refining the solution SLy. If a refinement is always guaranteed to exist, then the representation is called *adequate* or refinable. An example of an adequate representation is shown in Figure 2.3. If we map any problem in X to a problem in Y, every solution in Y can always be mapped back to X.

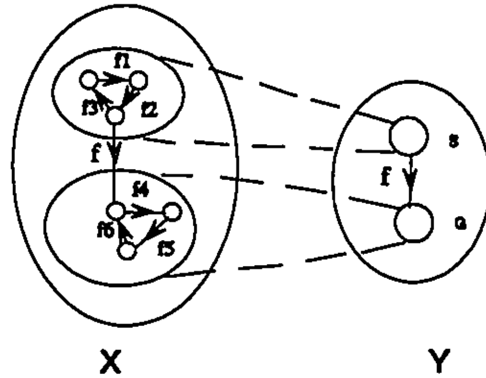



**Figure 2.3** An adequate representation

An example of an inadequate representation is shown in Figure 2.4. There is a solution in Y, namely S-f-G, which is not refinable, since there is no path going from s1 to g in X. This implies that the representation is inadequate. This will be further explained in Chapter 3.

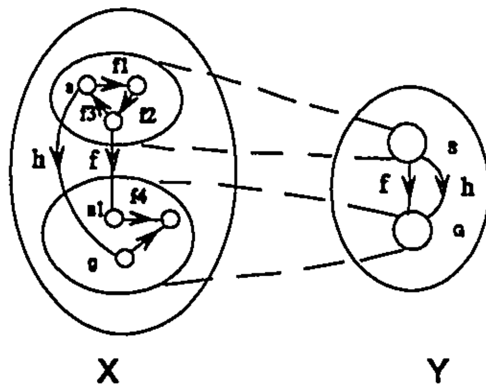
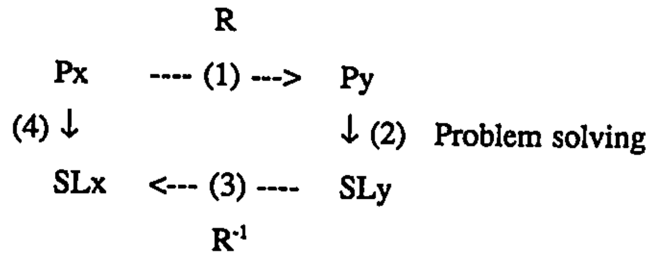



**Figure 2.4** An inadequate representation

A representation R that maps X to Y is said to be *efficient* if the sum of costs of the translations (1), (2) and (3) is less than the cost of step (4), shown in the following

diagram.




The strategy we follow in this thesis, is to apply this change of representation recursively. This is also known as "hierarchical problem-solving" [Knoblock, 1991]. The first step of change of representation is applied many times until a "simple to solve" problem space is created. It is usually a *trivial* space, i.e where the problem-solving activity is the identity. A clear example describing this is given in Chapter 3. The next step is to translate the solution back "traversing" all spaces until getting to the original problem. Figure 2.5 shows one step of representation for state-space.

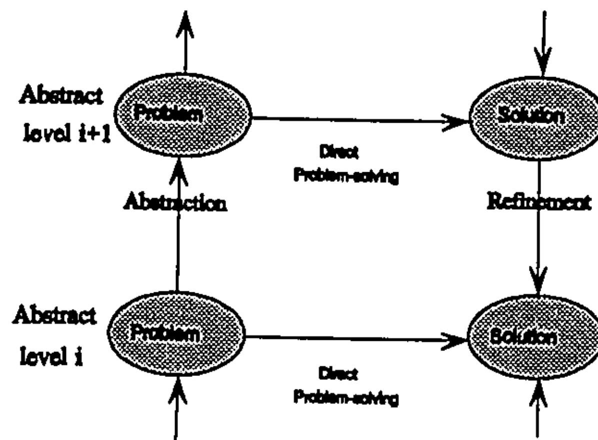
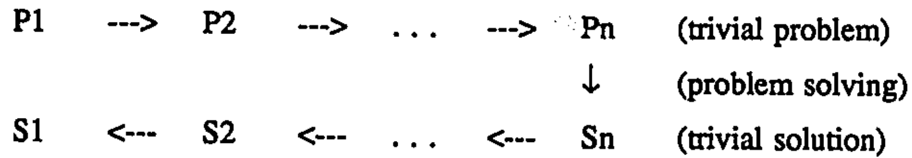



**Figure 2.5** Correspondence between levels of abstractions




When done hierarchically, the cost is the sum of all costs to transform P1 to Pn plus all costs to transform Sn to S1. Notice here that the cost to solve a trivial problem is zero (solve Pn to get Sn). To have an efficient representation, this sum must be less than solving directly P1 to get S1.

Remark:

We will use the terminology: good and bad abstractions. A **good** abstraction is an abstraction where the cost to abstract plus the cost to solve plus the cost to translate the solution back is less than the cost of direct solving. A **bad** abstraction is an abstraction which is not good. This corresponds to an efficient representation.

## 2.4 Various useful definitions

### 2.4.1 Partition of a set

Let I be a non-empty set, finite or infinite, and suppose that, for each i in I, we are given a set $X_i$. Then we say that we have a family of sets, written

$$F = \{X_i \mid i \in I\}$$

and we refer to I as the index set. A *partition* of a set X is a family $\{X_i \mid i \in I\}$ of non-empty subsets of X such that

(i) X is the union of the sets $X_i$ ($i \in I$),
(ii) each pair $X_i$, $X_j$ ($i \neq j$) is disjoint.

The subsets $X_i$ are called the *parts* of the partition.

**Theorem** (A classical result)

Let S(n,k) denote the number of partitions of an n-set X into k parts, where $1 \le k \le n$. then

$$S(n,1) = 1, \quad S(n,n) = 1,$$
$$S(n,k) = S(n-1,k-1) + k\,S(n-1,k) \qquad (2 \le k \le n-1)$$

□

This shows that the number of partitions grows very rapidly with the size of set to be partitioned.

## 2.4.2 Equivalence relation

Suppose that $\{X_i \mid i \in I\}$ is a partition of the set X. We can define R a relation on X.

$$x\,R\,x' \quad (x \text{ is equivalent to } x'),$$

whenever x and x' are in the same part $X_i$. If x, y, z are any members of X, not necessarily different, then we have the following statements, which are known by the names indicated:

| | |
|---|---|
| $x\,R\,x$ | (the *reflexive* property) |
| $x\,R\,y \Rightarrow y\,R\,x$ | (the *symmetric* property) |
| $x\,R\,y$ and $y\,R\,z \Rightarrow x\,R\,z$ | (the *transitive* property) |

R is said to be an *equivalence relation* if it is reflexive, symmetric, and transitive.

In this thesis, a *trivial* equivalence relation is one that maps X into a singleton say {s} (the partition is the whole of X, R = {X}), or the identity (the partition is the set of all elements of X, $R = \{\{x_1\},...,\{x_n\}\}$ where $\{x_1,...,x_n\} = X$).

Remark:

A partition P induces an equivalence relation:

$x \approx y \Leftrightarrow$ x and y are in the same part of P.

### 2.4.3 Isomorphism

In general, given two domains: A and B such that A has states $a_1,a_2,...,a_n$ and operations $g_1,g_2,...,g_m$, B has states $b_1,b_2,...,b_n$ and operations $h_1,h_2,...,h_m$.
An isomorphism $\Phi$ from A to B is a bijective map that satisfies the following conditions:

| | |
|---|---|
| • $\forall\ a_i\ \ \Phi(a_i)$ is defined | Total |
| • $\forall\ b_i\ \ \exists\ a_j$ such that $b_i = \Phi(a_j)$ | Surjective |
| • $\forall\ a_i, a_j\ \ \Phi(a_i) = \Phi(a_j) \Rightarrow a_i = a_j$ | Injective |
| | |
| • $\forall\ g_i\ \ \Phi(g_i)$ is defined | Total |
| • $\forall\ h_i\ \ \exists\ g_j$ such that $h_i = \Phi(g_j)$ | Surjective |
| • $\forall\ g_i, g_j\ \ \Phi(g_i) = \Phi(g_j) \Rightarrow g_i = g_j$ | Injective |
| | |
| • $\forall\ a, g_i\ \ \Phi(g_i(a)) = (\Phi(g_i))(\Phi(a))$ | Commutes |

**Example** (from [Holte, 1988])

A = {Objects = sorted finite lists (no duplicate elements),
Operations = merge}
B = {Objects = Finite sets
Operations = union}

An example of isomorphism from A to B is:

$\Phi(L) = S$, where L is a sorted list without duplicates, and S is the set of all elements of L.
$\Phi(\text{merge}) = \text{union}$.
$\Phi(\text{merge } L_1 \; L_2) = \text{union } (\Phi L_1) \; (\Phi L_2)$.

## 2.4.4 Homomorphism

An homomorphism $\Phi$ from A to B has to satisfy the following conditions:

| | |
|---|---|
| • $\forall \; a_i \;\; \Phi(a_i)$ is defined | Total |
| • $\forall \; b_i \;\; \exists \; a_j$ such that $b_i = \Phi(a_j)$ | Surjective |
| • $\forall \; g_i \;\; \Phi(g_i)$ is defined | Total |
| • $\forall \; h_i \;\; \exists \; g_j$ such that $h_i = \Phi(g_j)$ | Surjective |
| • $\forall \; a, g_i \;\; \Phi(g_i(a)) = (\Phi(g_i))(\Phi(a))$ | Commutes |

The difference from an isomorphism is that an homomorphism is not necessarily a bijection. Injectivity is not required ( instead of one-to-one, we allow many-to-one)

**Example** (from [Holte, 1988])

A = {Objects = sorted lists (may have duplicates),
Operations = Concatenation, merge}
B = {Objects = Finite sets
Operations = union}

$\Phi$, as it will be defined , is an example of homomorphism from A to B:

$\Phi$(L) = S, where L is an unsorted finite list that may have duplicates, and S is a set formed by all elements of L.
$\Phi$(merge) = $\Phi$(concatenation) = union.
$\Phi$(merge $L_1$ $L_2$) = union ($\Phi L_1$) ($\Phi L_2$).
$\Phi$(concatenation $L_1$ $L_2$) = union ($\Phi L_1$) ($\Phi L_2$).

## 2.4.5 Congruence

Given a domain X with states $x_1,x_2,...,x_n$ and operations $f_1,f_2,...,f_m$. First, we define the terms: *defined* and *undefined*. Given $x \in \{x_1,x_2,...,x_n\}$, and $f \in \{f_1,f_2,...,f_m\}$, f is defined on x if there exists $y \in \{x_1,x_2,...,x_n\}$ such that f x = y, and otherwise f is undefined on x.

A relation R is a congruence on X if and only if:

1) R is an equivalence relation on X.

2) $\forall\ x_i, x_j \in \{x_1, x_2, \ldots, x_n\}$,

$x_i \ R \ x_j \Rightarrow \forall\ f \in \{f_1, f_2, \ldots, f_m\}$,

either (f is undefined on $x_i$ or f is undefined on $x_j$)

or (f is defined on $x_i$ and f is defined on $x_j$ and $(f\ x_i)\ R\ (f\ x_j)$).

Remark: A congruence induces a homomorphism.

# Chapter 3

# Problem-solving by Abstraction/Refinement

Recall from chapter 1, the task we are investigating is:

Given: a state-space $P = <S,I>$

Find: A hierarchy of state-spaces (an abstraction hierarchy) $P_0 \ldots P_k$ such that $P = P_0$ and $P_{i+1}$ is an "abstraction" of $P_i$ ,where $0 \leq i \leq k\text{-}1$.

Our overall aims are:

1) Constructing the hierarchy $P_0 \ldots P_k$ should be efficient.
2) Solving a problem instance using the hierarchy should be faster on average than solving the problem instance in P itself (i.e the hierarchy should be a good abstraction).
3) The solutions found using the hierarchy may be longer than the solution found in P itself, but they should not be "excessively" large.
4) Storing the hierarchy into the memory should be efficient.

In this chapter, we give a detailed description of our approach to this task which

1) Defines the kind of abstraction we will use.

2) Defines how we can use an abstraction hierarchy to solve a problem instance. In particular, gives a detailed discussion of refinement.

3) Sketches our initial algorithms for constructing an abstraction hierarchy.

## 3.1 Abstraction = Homomorphism

Given the original state-space $P = <\{S_1,...,S_n\}, \{i_1,...,i_m\}>$, we have to find a partition $Q = \{C_1,...,C_p\}$ on P that satisfies the following constraints:

1) Viewed as a relation, Q is a congruence on P (inducing a homomorphism, say, $\Phi$)

2) $Q = \{C_1,...,C_p\}$ is non-trivial, i.e $1<p<n$.

3) Refinability: Given a problem Pr in P, if $\Phi(Pr)$ has a solution $\Sigma_Q$ in $<Q,\{\Phi(i_1),...,\Phi(i_m)\}>$, then there exists a solution $\Sigma_P$ for Pr in P such that

$$\Phi(\Sigma_P) = \Sigma_Q.$$

To construct a hierarchy, we repeat this process until there is no partition with all three properties.

When abstracting a state space, we want always to avoid - if possible - trivial abstractions or trivial equivalence relations. There are two possibilities of trivial abstractions:

1. When the abstract state space is isomorphic to the original one. This means that each state is by itself an equivalence class. Surely this abstraction does not improve problem solving speed since the state space is the same.

2. When the abstract state space has only one state. This means that all the states map to a single equivalence class. This abstraction is not needed, because when refining the solution, we will be solving the entire problem in the original space.

# 3.2 The use of abstraction for problem solving

Search has two purposes:

- Searching in the abstract space
- Refining the abstract plan

For all search, we use the classical algorithm Breadth First Search.

## 3.2.1 Refinement and refinability

Refinement is the final part in the whole process of solving a problem. First, a problem is abstracted through successive levels of a given abstraction hierarchy. Second, search is done in the most abstract space to find an initial solution. Third, the solution is translated back down the hierarchy, one level at a time, to create a solution at every level including, in the end, the original level.

**Example** The 3-disk Tower of Hanoi (TOH3)

Suppose that the abstraction is done as follows:

- Homomorphism 1: Space1 (the original space) → Space2
  - All states connected by the operators -1 or 1 → Same single state
  - Operators -1 and 1 → Identity
  - Operators -3,-2,2 and 3 → Themselves

- Homomorphism 2: Space2 → Space3
  - All states connected by the operators -2 or 2 → Same single state
  - Operators -2 and 2 → Identity
  - Operators -3 and 3 → Themselves

This abstraction is the "standard" abstraction of TOH3. Each level of abstraction corresponds to "ignoring" the smallest disk at that level (see Figure 3.1).

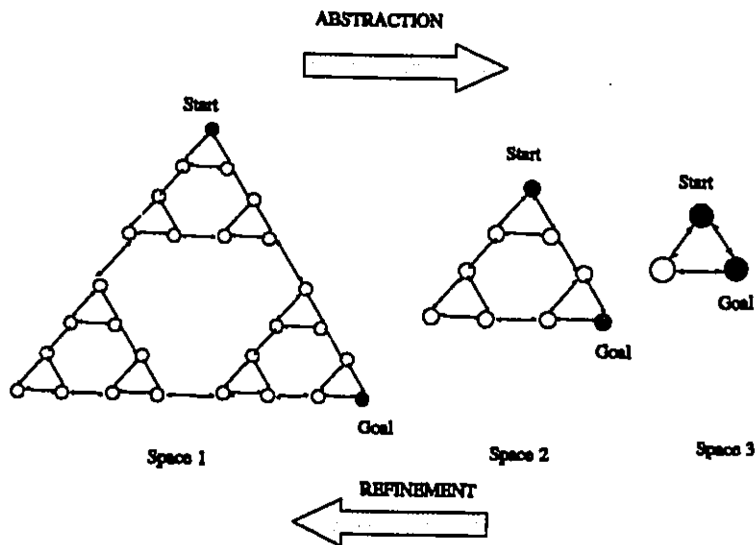



**Figure 3.1** The standard abstraction of the 3-disk Tower of Hanoi

A given problem is translated into start and goal states in each level. Problem-solving starts with the most abstract level and goes to the original space. The solution, written with operators only, is found as follows:

(1) First, the solution is calculated in space3: An identity followed by (-3) (i.e move disk3 counterclockwise) followed by another identity. $I_2$ (-3) $I_2$

(2) Second, Refining that solution in space2 gives an identity followed by operator (2) followed by an identity followed by operator (-3) followed by an identity followed by operator (2) and finally followed by an identity. $I_1$ (2) $I_1$ (-3) $I_1$ (2) $I_1$.

(3) Finally, Refining the solution, found in space2, in the original space1 is actually determining the values of identities, which gives: (-1) (2) (-1) (-3) (-1) (2) (-1).

Which, simply, means (move disk1 to peg3) (move disk2 to peg2) (move disk1 to peg2) (move disk3 to peg3) (move disk1 to peg1) (move disk2 to peg3) (move disk1 to peg3).

□

In this example, if we have a solution plan at abstraction level i, then it is always possible to refine it into a solution at level i-1. Refinement is applied until we reach the original state space. The solution was given, first, at the highest level of abstraction where the number of states is 3, then refined at the next level where the number of states is 9 and finally refined at the base level where the number of states is 27. During this process, there was no *backtracking* at all. Backtracking means: if a solution is found at the abstract level but has no refinement at the current level then we look for another abstract solution and try to refine it again. This could happen recursively at many levels and thus cause significant inefficiency.

Refinability means: any solution S (for problem P) at abstraction level i, can always be refined into a solution for P at level i-1. If refinability is guaranteed at every level of abstraction then this will eliminate backtracking in search.

We want to have conditions on the way we abstract that guarantee refinability. Stronger conditions may eliminate good abstractions. The ultimate goal is to find the

weakest conditions possible to fulfil this. If we require, for instance, that we only accept an equivalence relation such that for all classes C and for all operators f, either f is defined on all members of C or f is undefined on all members of C. This satisfies the refinability condition because any abstract plan can always be refined. However this is a strong condition that "asks a lot" and may prevent us from finding good abstractions where this "identical functionality" is not satisfied but where the refinability condition is satisfied.

**Example**

The standard abstraction of TOH2 (Tower of Hanoi with 2 disks) does not satisfy the "identical functionality" because the operator "move disk2" (i.e, operators (-2) and (2)) is not defined at the "corner" states. In Figure 3.2, operator "move disk2" applies only to states which are marked with (2).

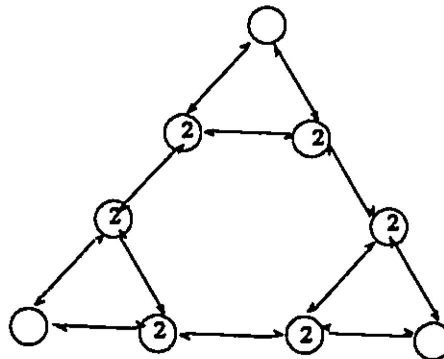



**Figure 3.2** TOH2

□

Solution:

The condition we impose is that, any two states that belong to the same

equivalence class have to be "reachable" from each other (i.e there is a path from the first state to the second and a path from the second to the first). Formally:

Given a state-space $P = <\{x_1,...,x_n\}, \{i_1,...,i_m\}>$. For all $x, y \in \{x_1,...,x_n\}$

| | |
|---|---|
| if | $x \approx y$ |
| then | $\exists$ a sequence of operators from $\{i_1,...,i_m\}$, say $op_1...op_k$ such that $(op_1...op_k)\ x = y$, (i.e $op_k(...(op_1 x)) = y$). |
| and | the same for y. |

## 3.2.2 Searching

Search has two purposes:

- Normal problem-solving search at the most abstract level
- Refinement at the current level of the solution found at the abstract level

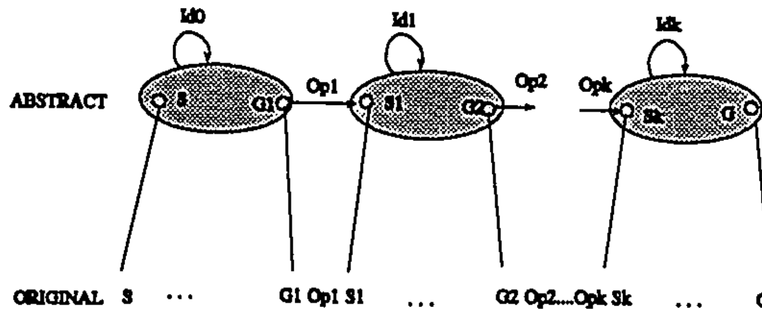



**Figure 3.3** Refinement

Suppose that we are looking for a path from a start state "S" to a goal state "G" (see Figure 3.3). In the abstract space, we start from the class containing the start state S (which is the start state in the abstract space), and search to find a solution plan in the abstract space, say:

$$SP_a = Id_0\text{-}Op_1\text{-}Id_1\text{-}Op_2\text{-}\ldots\text{-}Op_k\text{-}Id_k \text{ ( the solution is } Sol_a = S_{a0}\text{-}Op_1\text{-}S_{a1}\text{-}Op_2\text{-}\ldots\text{-}Op_k\text{-}S_{ak})$$

Where $Op_i$'s are the operators in the abstract solution plan, and $Id_i$'s are identities which might be translated into non-identities with respect to the current space. Refining this solution means that we have to look for a state in the class $S_{a0}$ where we can apply $Op_1$. Once found, say $G_1$, we search for a path from S to $G_1$, apply $Op_1$ to get to $S_1$ in the class $S_{a1}$ and then look for a state in that class where we can apply $Op_2$. Again, once found, say $G_2$, we search for a path from $S_1 = Op_1(G_1)$ to $G_2$ and so on ...We keep on going until we reach $G_k$ in class $S_{a(k-1)}$ and then, finally, we search from $S_k = Op_k(G_k)$ to G. Notice that $S_i = Op_i(G_i)$ and $G_{i+1}$ both belong to class $S_{ai}$ (it is a class with respect to the current space and a state with respect to the abstract space).

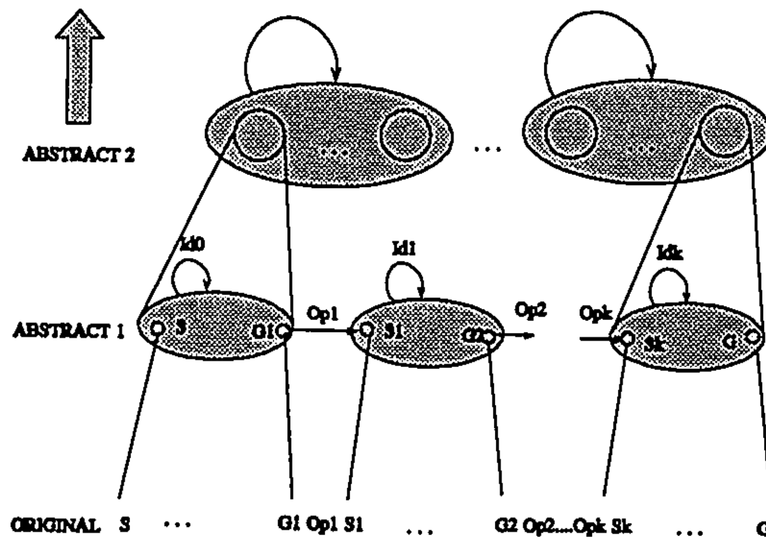



**Figure 3.4** Hierarchical search

At the highest level search could be trivial (no search when the start state is equal to the goal state), as is the case when the whole space becomes one state. Figure 3.4 shows how search is done hierarchically.

## 3.3 Initial algorithms and their deficiencies

To abstract a state space, we, initially, used two algorithms. The second algorithm is faster than the first. Those algorithms are:

- Class extension
- Chain extension

The system was first built using the first algorithm, and because of problems of efficiency and the need to find good abstractions, it was necessary to explore new alternatives on how to abstract and how to be very efficient when doing so. That was the motivation for the algorithms described in Chapter 5.

### 3.3.1 The class extension algorithm

Let the set of states = $\{S_1,...,S_k, x, y\}$. We choose two states, x and y, and compute the "smallest" congruence in which $x \approx y$. The smallest congruence is the one in which fewest states are related to one another. The algorithm is as follows:

Let Done = $[\{S_1\},...,\{S_k\}, \{x,y\}]$ be a partition of states which will be successively updated until it is the solution. Let Ready = $[\{x,y\}]$, a list of equivalences requiring processing.

Repeat until Ready is empty

1) Remove a set $\{C_1,...,C_m\}$from Ready.

2) For all operators f, create the set $C_f = \{f\ C_1,...,f\ C_m\}$

3) If $C_f$ is a subset or equal to any set in Done or Ready then discard it. Otherwise find all sets $A_1,...,A_n$ on Done or Ready that intersect with $C_f$. Delete them. Add $A_1 \cup ... \cup A_n$ to Done and Ready

**Example** TOH2

For the sake of clarity, we only show the "positive" operators in Figure 3.5. The negative operators are the inverses. For example, to get from state 11 to state 21, we apply operator 1 (as shown in the figure), which means that to get from 21 to 11, apply operator -1.

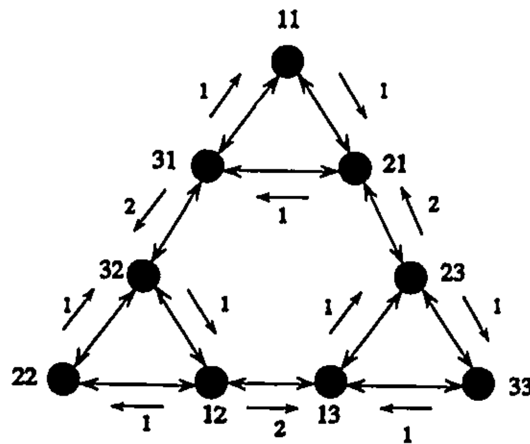



**Figure 3.5** The labelled TOH2

Suppose that the 2 first states chosen are 12 and 13, then

Done = [{11},{31},{21},{32},{23},{22},{33},{12,13}], Ready = [{12,13}]

| | |
|---|---|
| Apply -2 | maps 12 to nothing and 13 to 12, so $C_{(-2)}$ = {12} |
| | ⇒ Discard |
| Apply -1 | maps 12 to 32 and 13 to 33, so $C_{(-1)}$ = {32,33} |
| | ⇒ Delete {32} {33} from Done, add {32,33} to Done and Ready |
| Apply 1 | maps 12 to 22 and 13 to 23, so $C_1$ = {22,23} |
| | ⇒ Delete {22} {23} from Done, add {22,23} to Done and Ready |
| Apply 2 | maps both 12 to 13 and 13 to nothing, so $C_2$ = {13} |
| | ⇒ Discard |

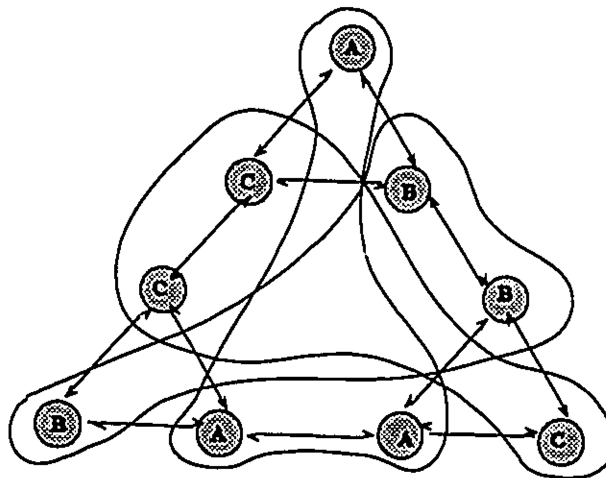



**Figure 3.6** An example of abstraction of TOH2

Doing the same thing to the pair {32,33} (i.e applying -2, -1, 1 and 2) will give respectively: {31}, {22,23}, {12,13}, {}. They are all discarded, as they are already on Done. Again, doing this for the pair {23,22} will give: {}, {13,12}, {33,32}, {21}. Those, too, are all discarded. At this point, the following partition has been constructed: Done =

{{12,13},{32,33},{23,22},{21},{31},{11}}. Ready is empty, so Done is the final partition, which corresponds to a congruence. This is the first level of abstraction of TOH2. Suppose to proceed to the second level we choose the 2 first states, say {12,13} and {11}, then the result obtained will be {{11,13,12},{31,33,32}{21,23,22}} (see Figure 3.6). If we proceed further, the whole space will collapse into a single state.

□

The way this has been implemented is such that the two first states are chosen to be connected states (i.e there is an arc between the two nodes). That will reduce the number of choices to be checked, but at the same time will assume always that the congruence is the result of two connected states, whereas we can have some congruences where none of the states is directly connected to the other states within the same class. See the classes (A,B,C) in Figure 3.7. But, later, we will see this is a reasonable restriction.

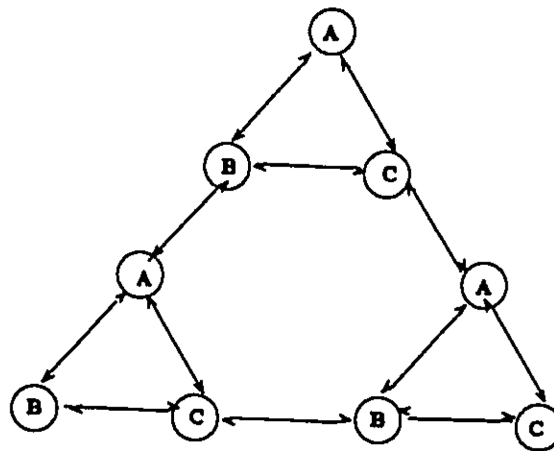



**Figure 3.7** An example of abstraction on TOH2

## 3.3.2 The chain extension algorithm

The idea in this algorithm comes from exploring the fundamental property of a

congruence. We start with any state S from which we can apply at least one operator, say f.

The first two states S and fS are supposed to be in the same equivalence class i.e $S \approx fS$. Applying f from both sides gives $fS \approx f^2 S$ which in turn implies that $S \approx f^2 S$ because of the transitivity property. Repeating this process (applying f) gives the final chain: S, f S, $f^2$ S,...,$f^n$ S such that if we apply f to $f^n$ S we get a state that belongs to the chain itself (creating in this manner a cycle with f only), or f cannot be applied to $f^n$ S. We conclude that all the states of this chain are in the same equivalence class, say the class C, and f is an identity on C.

Further, if $f^k$ S maps to $f^r$ S with a function g (different from f), then, like f, g must be an identity on C. This is true because of the following:

Let $y = f^k S$ and $z = f^r S$. Since both of them belong to the f-chain they are equivalent (they both belong to the same equivalence class C). We have then $y \approx z$. Knowing that from y we go to z by applying g, i.e $z = gy$ and $y \approx z \Rightarrow y \approx gy$. We are now in the same situation as S and fS. Another chain y, gy, $g^2$ y, ... can be built and has to belong also to the same class C. Also, if one state in the f-chain is connected to another state in the g-chain with some operator h, we can build another h-chain following the same reasoning. Figure 3.8 illustrates this approach.

Repeating that process until no possible identities are found will allow us to find the first equivalence class C. C could then be "extended" (as in the first algorithm) to get a congruence. In this way, we made a great improvement compared to the first algorithm. Instead of the extension of two states only, we extend a bigger class.

This was implemented and had showed a considerable gain in time compared to the class extension algorithm.

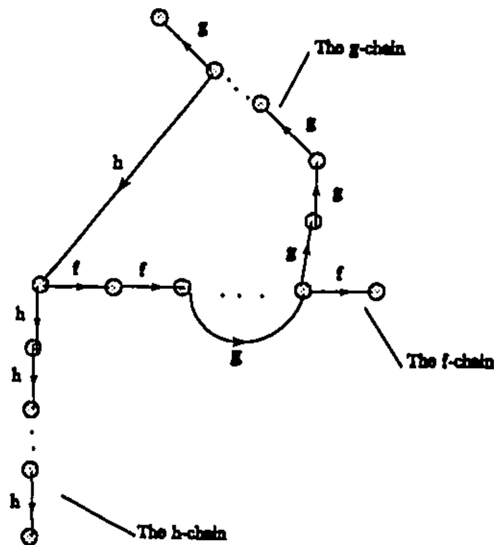



**Figure 3.8** The chains in a state space

### 3.3.3 Deficiencies of initial algorithms

The two algorithms share the same kinds of problems. Those problems are summarized in the following points:

1) Sensitivity to initial formulation: Consider the graph in Figure 3.9. If the two first chosen states are 1 and 2 then the following will happen:

| | |
|---|---|
| Apply a | maps 1 to 2 and 2 to nothing, so the result is {2} |
| Apply b | maps 1 to 3 and 2 to 4, so the result is {3,4} |
| Apply c | maps 1 to nothing and 2 to 3, so the result is {3} |

Then, doing the same with {3,4}

| | |
|---|---|
| Apply a | maps 3 to 1 and 4 to 3, so the result is {1,3} (*) |
| Apply b | maps 3 to 4 and 4 to nothing, so the result is {4} |

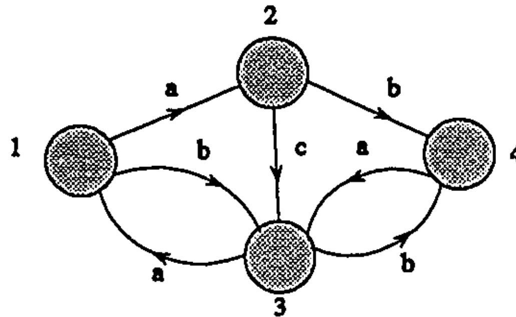



**Figure 3.9** An example of a graph

The line indexed by (*) will cause the following:

$1 \approx 3$ and since, from the beginning, $1 \approx 2$ then (transitivity of $\approx$) $1 \approx 2 \approx 3$ and since also, from second line, $3 \approx 4$, then $1 \approx 2 \approx 3 \approx 4$, and the whole collapses into a single class. Note, however, that if the arrow mapping 3 to 1 has a different name, say d, then we obtain the following nontrivial congruence: $\{\{1,2\}, \{3,4\}\}$ (see Figure 3.10).

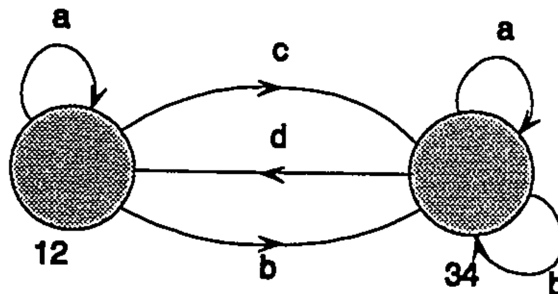



**Figure 3.10** Abstraction of the previous space

2) The algorithms are slow: This is due to their "generate and test" nature. They can exert a big effort to "extend" a class which finally collapses into a trivial abstraction. Then another two states would be chosen and the algorithms would "start from scratch" etc...Also, refinability is not automatically guaranteed, it is satisfied by the "generate and test" strategy.

3) No control over the contents of most classes: It is hard to foresee the effects of the choice of the two starting points. One choice may lead to total collapse, another to an abstraction with one class containing only these two states and other states in classes by themselves. Another choice may produce some classes with many states and all other classes with only few, whereas a different choice of starting points may produce perfectly "balanced" classes.

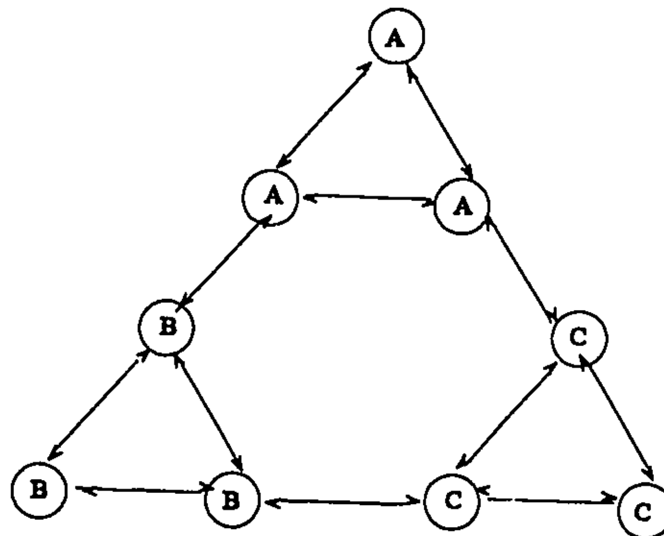



**Figure 3.11** The standard abstraction of TOH2

The two first states chosen to be together usually have a great impact on the "goodness" of the abstraction to be obtained. In the example shown previously, if the two first states chosen to be together were, for instance, 11 and 21 (Figure 3.5), then following the same

process, we will end up with a good abstraction, the standard abstraction of the Towers of Hanoi (Figure 3.11). In Chapter 6, we show empirically that the standard abstraction of Towers of Hanoi is a very good abstraction (probably the best). Another drawback is that we cannot predict whether the abstraction to be obtained will be good or bad.

## 3.4 Solution: graph relabelling

The graph representing the state-space will be relabelled so that each edge has a unique label in the graph. This will guarantee the following advantages:

1) Insensitivity to initial formulation.

2) Total control over the contents of all classes of an abstraction.

<u>1) Insensitivity</u>

When a graph is relabelled, it is treated exactly as an unlabelled graph. Recall from the previous section that initial formulation of the problem has influence on the resulting congruence. After relabelling, as we will prove below, any partition is a congruence. This means that regardless of how we formulate the problem initially, we can always abstract the way we like and thus are completely insensitive to initial formulation.

<u>2) Abstraction control:</u>

Since any partition is a congruence, we can construct any one we like. We will be able to choose the size of classes, the number of levels in the hierarchy of abstractions, the number of classes at each level, etc... This flexibility permits us to construct good abstractions and avoid the inefficiency of a "generate and test" method and also facilitates the satisfaction of the refinability constraint.

### 3.4.1 Formal discussion of graph relabelling

Given a state-space $P = <\{S_1,...,S_n\}, \{i_1,...,i_m\}>$, define the following mapping $\Psi$ (called the relabelling mapping) which has the properties:

$\Psi(P) = RP = <\{S_1,...,S_n\}, \{j_1,...,j_k\}>$ where

1) $k \geq m$

2) For all $a \neq b$, $j_a \neq j_b$ $(1 \leq a,b \leq k)$

3) For all $i \in \{i_1,...,i_m\}$, For all $x \in \{S_1,...,S_n\}$, if i is defined on x, say $ix = y$, then $\exists$ a unique $j \in \{j_1,...,j_k\}$ such that $jx = y$ and for all $h \in \{j_1,...,j_k\}-\{j\}$, h is not defined on x and finally for all $z \in \{S_1,...,S_n\}-\{x\}$, j is not defined on z.

Consequence:

Suppose we are given two states x and y that belong to the same equivalence class, C. For any $j \in \{j_1,...,j_k\}$, there are three and only three possibilities that could happen:

1) jx is defined and jy is not defined

2) jx is not defined and jy is defined

3) Both jx and jy are not defined

This implies that j (C) is either empty or a singleton.

**Claim**

After relabelling, every partition induces a homomorphism (is a congruence).

**Proof**

Let RP = <$\{S_1,...,S_n\}$, $\{j_1,...,j_k\}$> be the relabelled space, and $\{C_1,...,C_p\}$ any partition of $\{S_1,...,S_n\}$. Define a map $\Phi$ such that

1) $\Phi(S_a) = C_b$ ($1 \le a \le n$ and $1 \le b \le p$) if and only if $S_a \in C_b$, and

2) $\Phi(j_r) = J$ ($1 \le r \le k$) defined on $\{C_1,...,C_p\}$ such that

   If the unique state where $j_r$ is defined is x, say $j_r x = y$, let c1 and c2 $\in \{C_1,...,C_p\}$ such that $x \in c1$, $y \in c2$, then J(c1) = c2 and for all $c \in \{C_1,...,C_p\}-\{c1\}$, J is undefined on c. Say, in all we will have $\{J_1,...,J_q\}$.

Want to prove that $\Phi$ is a homomorphism. Recall, from chapter 2, that $\Phi$ has to be total, surjective (for both states and operators) and commute.

1) $\forall x \in \{S_1,...,S_n\}$, there is a unique element c in $\{C_1,...,C_p\}$ such that $x \in c$. This is a straight deduction from the definition of a partition. So $\Phi x$ is defined and unique, so $\Phi$ is total on $\{S_1,...,S_n\}$.

2) $\forall c \in \{C_1,...,C_p\}$, $c \neq \{\}$ because $\{C_1,...,C_p\}$ is a partition. So $\exists$ at least one $x \in \{S_1,...,S_n\}$ such that $x \in c$, i.e $c = \Phi x$, so $\Phi$ is surjective on $\{C_1,...,C_p\}$.

3) By definition, $\{J_1,...,J_q\} = \Phi(\{j_1,...,j_k\})$, so $\Phi$ is total on $\{j_1,...,j_k\}$, and surjective on $\{J_1,...,J_q\}$.

4) $\forall x \in \{S_1,...,S_n\}$, $\forall j \in \{j_1,...,j_k\}$. Want to prove that if jx is defined then $\Phi(jx) = (\Phi(j))(\Phi x)$.

If jx is defined, say jx = y, both x and y must belong to some element(s) in the partition, say $x \in c1$, $y \in c2$ (i.e $\Phi x = c1$ and $\Phi y = c2$). This implies that $(\Phi(j))(c1) = c2$, i.e $(\Phi(j))(\Phi x) = \Phi(jx)$, so $\Phi$ commutes.

<u>Conclusion:</u>

$\Phi$ is a homomorphism. □

# Chapter 4

# Complexity analysis of using abstraction

The ultimate goal of changing representation is the improvement of problem-solving efficiency. That is the cost of abstracting (as many times as we need to) plus the cost of solving a problem should be less expensive than solving directly in the original space. Analyzing the problem-solving activity using abstract problem spaces will guide us to look for the best way one should abstract. The analysis, discussed in this chapter, is very useful for evaluating abstraction. It will, also, guide us directly to select the more efficient among a set of possible abstractions. This analysis deals with a connected graph. If the graph is disconnected then we treat each connected subgraph separately.

## 4.1 Analysis of problem-solving efficiency

Recall from Chapter 2 that, when we have a hierarchy of abstractions, solving the problem is mainly refinement. For the sake of clarity and avoiding complexities to solve the problem, we make the following assumptions in this analysis [Holte et al, 1992]:

1) The hierarchy includes the original space and excludes the last trivial space (the one

with a single state only).

2) Each class at any level of abstraction has the same number of states.

3) The number of states in the abstract space is strictly less than the number of states in the original space.

Let A be the number of non-trivial state-spaces including the original. For example, if we abstract only once and the abstract space is not trivial, A is then equal to 2. Let c be the number of states within a class at any level of abstraction. Let N be the number of states in the original space. N is, then, equal to $c^A$. Notice that c is different from 1, otherwise the abstract space will also have N states, and thus violate assumption (3). This implies that $A = (\ln N) / (\ln c)$ where ln is the natural logarithm. The diameter of a class is the maximum *distance* between 2 states in the class. The distance between two nodes R and Q in a graph is the number of states including both endpoints in the shortest path from R to Q.

**Example:** See Figure 4.1

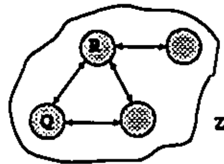



**Figure 4.1** The notion of diameter

The distance from R to Q is 2.

The distance from R to R is 1.

The diameter of Z is 3.

□

For the purpose of analysis, we assume that the diameter of all classes at all levels of abstraction is the same. Let d denote the diameter.

**Example** TOH2: N = 9, c = 3, A =2. d = 2.

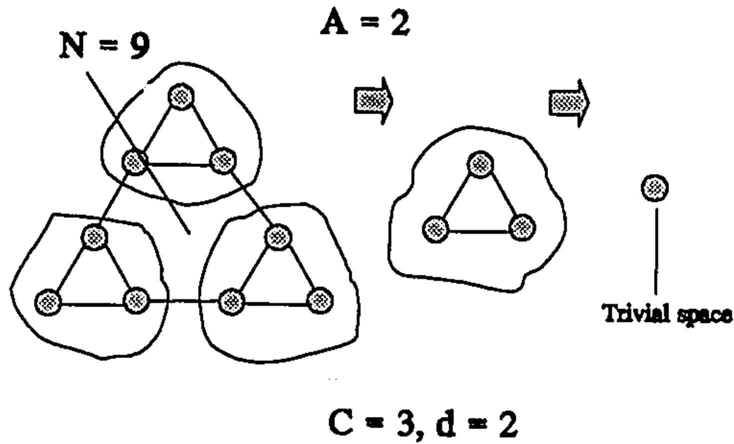



**Figure 4.2** The essential parameters in Standard abstraction of TOH2

□

Remark:

It is possible for the diameter of a class to be greater than the number of states in the class, i.e d > c. Figure 4.3 shows an example of such a situation. If we consider, for instance, the class of B's in the first abstraction, it is clear that the distance between the two "non-corner" states is equal to 4, and the distance to the "corner" state is equal to 3, and thus d = 4. But c, in this case is equal to 3.

Define a *compact* class Q as a class where, given any two states S1 and S2 in Q, there exists a solution Sol = $s_1$ $Op_1$ $s_2$ ... $Op_{k-1}$ $s_k$, to the problem <S1=$s_1$,S2=$s_k$>, such that $s_1,s_2,\ldots,s_k$ all belong to Q.

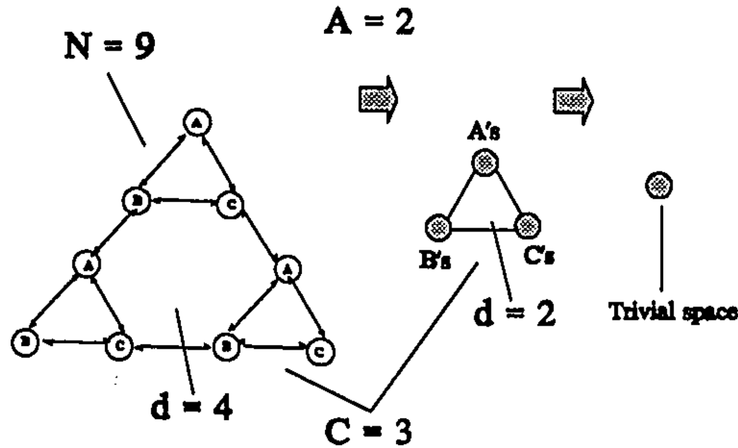



**Figure 4.3** The essential parameters in a particular abstraction of TOH2

Remark: The diameter d is always less than or equal to c for compact classes.

The *expansion factor*, denoted x, is defined as the average ratio of the length of a refinement to the length of the solution from which it was derived [Stefik and Conway, 1982]. This means that at any level i, where $1<i\leq A$, if $Sol_i$ is the solution at that level and $Sol_{i-1}$ is the refinement of $Sol_i$, which is in turn the solution of the same problem at level i-1, then the ratio is (length $Sol_{i-1}$)/(length $Sol_i$) (Note that level 1 corresponds to the original space and thus $Sol_1$ is the final solution). Suppose at the level i the solution is $Sol_i = S_{1i}\ Op_{1i}\ S_{2i}\ Op_{2i} \ldots Op_{(k-1)i} S_{ki}$. The expansion factor x is such that, the refinement of this solution is $Sol_{i-1} = S_{11(i-1)}\ Op_{11(i-1)} \ldots Op_{1(x-1)(i-1)}\ S_{1x(i-1)}\ Op_{1i} \ldots Op_{(k-1)i}\ S_{k1(i-1)} \ldots S_{kx(i-1)}$. In other words, it is the length of the "part" of the solution that "expands" one abstract state. In the example given, the abstract state $S_{1i}$ expands to $S_{11(i-1)}\ Op_{11(i-1)} \ldots Op_{1(x-1)(i-1)}\ S_{1x(i-1)}$. The maximum value of x is then d, the diameter of the class "being expanded". Figure 4.4 shows an example where the expansion factor, x, is equal to 3. At the trivial level, the solution, S, has a length of 1. At the next level (level A), S is expanded to produce a solution of length 3. Each state in this solution is expanded into 3 states at

level (A-1) to produce a solution of length 9.

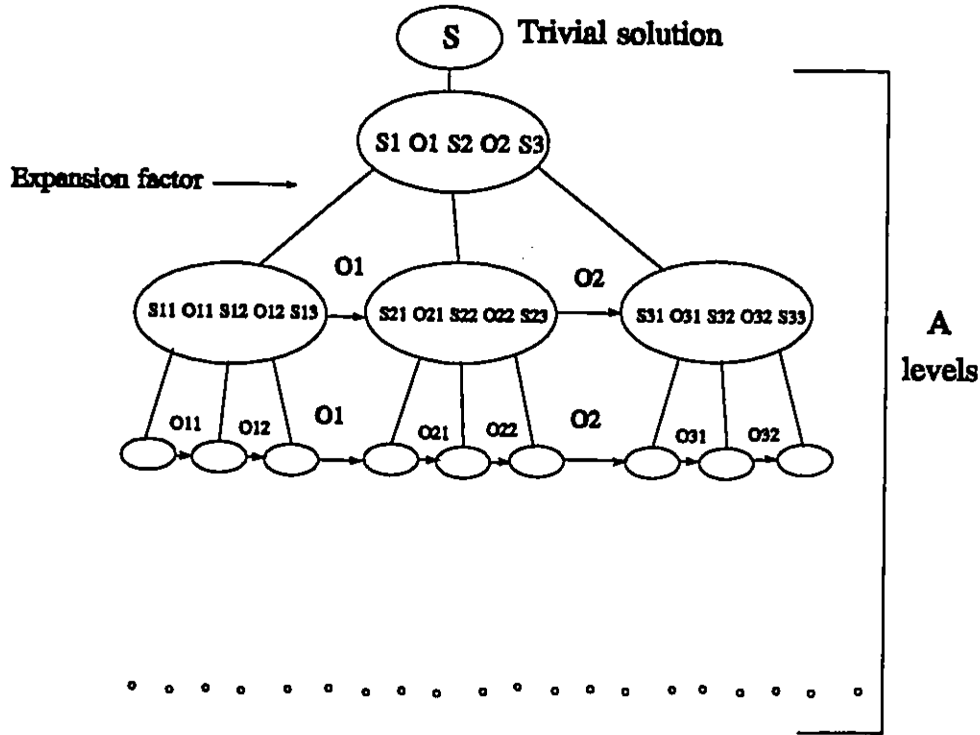



**Figure 4.4** The process of refining

Finally, let w denote the amount of "work" or cost required to "refine a single class" (e.g. to expand $S_{11}$ into $S_{11(i-1)}\ Op_{11(i-1)} \ldots Op_{1(x-1)(i-1)}\ S_{1x(i-1)}$ ). In other words, w is the cost to find a path from a start state to a goal state within the same class. In Figure 4.4, w is, for instance, the cost to find the path from S11 to S13 (i.e S11 O11 S12 O12 S13) within the class S1 at the level A-1.

To proceed with the analysis, here are all the assumptions:

(1) $c \neq 1$

(2) c is constant

(3) d is constant

(4) x is constant

(5) w is constant

Assumptions (1)-(3) were already mentioned previously. Assumption (4) means that the "refinement of each class" of a solution has the same length for all classes at every level, and (5) means that the cost to do refinement is the same everywhere.

Let TW denote the total work required to refine the most abstract solution down to the original level. At the highest level where the solution is trivial the work needed to refine it is just w, at the second highest level the solution has a length of x, the expansion factor, so to refine this solution, the work needed is w times x, at the level after, the work is w times $x^2$, and so on until we get to the original level.

$$
\begin{aligned}
TW &= w + x\,w + \dots + x^{(A-1)}\,w \\
&= w\,(1 + x + \dots + x^{(A-1)}) \qquad \text{(* Equation 1 *)} \\
&= w\,(x^A - 1)/(x - 1) \qquad \text{(when } x \neq 1\text{)} \\
&\approx w\,x^{(A-1)}
\end{aligned}
$$

Since x is bounded by d then

$$TW \leq w\,d^{(A-1)}$$

Knowing that A = (ln N)/(ln c), it follows that

$$TW \leq w\,d^{((\ln N)/(\ln c))-1} \qquad \text{(* Equation 2 *)}$$

To minimize the term $d^{((\ln N)/(\ln c))-1}$, we should **minimize d** and **maximize c**.

If we use the above lemma, $TW \leq w/d\ N^{(\ln d)/(\ln c)}$ (* Equation 3 *)

<u>Remark</u>

If there is no abstraction at all, i.e $A = 1$ and $c = N$.

$$
\begin{aligned}
TW &\leq w\ d^{((\ln N)/(\ln c))-1} \\
&= w\ d^{1-1} \\
&= w
\end{aligned}
$$

In this case, w is the work required to solve a problem in the original space.

## 4.1.1 Examples of the "work" w

The "work" w is very difficult to estimate analytically, because it depends on exactly which subproblems (<start,goal> pairs) arise during refinement. For the sake of illustration, we will assume that all subproblems are equally likely, i.e that during refinement of class Z, all pairs of states in Z are equally likely to arise as subproblems. We will also assume that we are using breadth first search to refine a single class.

### 4.1.1.1 Example 1

As a first example, consider the standard abstraction of the 2-disk Towers of Hanoi puzzle (see Figure 4.5). This is one of the few abstractions in which c and d are actually constant. The work w is equal to the average "effort" (amount of search) of solving the problem to get from any start state to any goal state within a "triangle". In all we have 9 possible pairs (start,goal) in a triangle, this includes the pairs where the start

and goal states are the same. To get from one state to itself, the "effort" is equal to 1. To get from state 1, for instance, to state 2, the effort is either 2 or 3, depending on the order in which we search. If the effort is 2, then to visit state 3, the effort is 3 and if the effort is 3 then visiting state 3 will require an effort of 2. In all, the sum of "efforts" to get from state 1 to itself, to state 2 and to state 3, is equal to $1 + 2 + 3 = 6$. This exactly the same if we solve from state 2 to others and also if we solve from state 3 to others. In all we have to exert an effort of $6 + 6 + 6 = 18$ to solve the 9 possible pairs of problems. On average, w is then $18 / 9 = 2$.

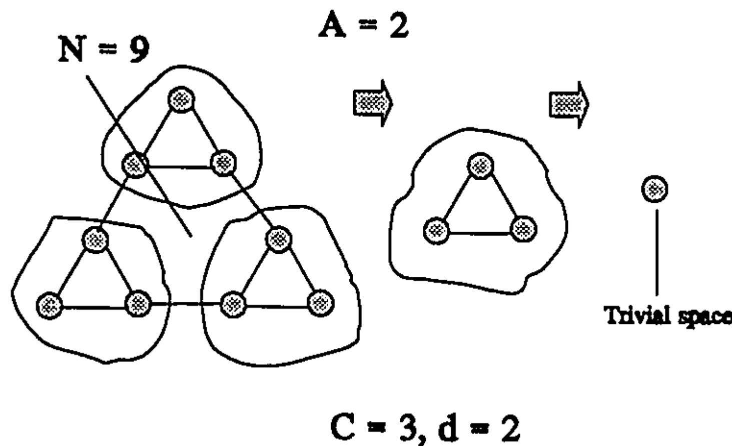



**Figure 4.5** The essential parameters in Standard abstraction of TOH2

#### 4.1.1.2 Example 2

In this example, (see Figure 4.6), d is not the same at every level. Let's calculate the value of w for the first level of abstraction. If we take the class of A's, for instance, then call the one in the top state 1, the one on the left state 2 and the last one state 3. To get from a state in this class to itself we require an "effort" of 1. To get from state 1 to 2, the effort is either 4 (in which case getting to 3 will require an effort of 5) or 5 (and thus getting to 3 requires 4), and this depends on the order in which we search. To get

from state 2 to state 1, the effort varies between 5,6,7 and 8 (average is 6.5). Finally, to get from 2 to 3, the effort varies between 9, 10 and 11 (average is 10). It is the same when starting with state 3. In all there are 9 paths, and an average value of w will be: ((1+4+5 + 2*(1+6.5+10))/9 = 5.

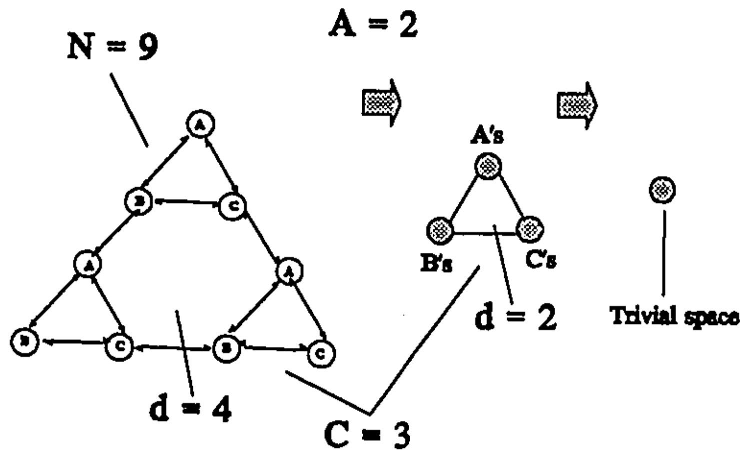



**Figure 4.6** The essential parameters in a particular abstraction of TOH2

## 4.1.2 Discussion of the "work" w

Recall from the analysis (Equation 2), that $TW \leq w\ d^{((\ln N)/(\ln c))-1}$. In this section, we consider how to minimize w. This parameter depends on the way we abstract and the searching algorithm we use. Here are some different cases:

(1) Suppose that when we abstract we build a lookup table for each class, and do refinement, when problem-solving, by table lookup. In this case w is equal to a constant.

(2) Suppose that when we abstract, each class is represented by a graph. We know that breadth first search (and most searches, e.g. depth first search) has a time complexity of

O(N+E) for a graph with N nodes and E edges [Moore, 1959]. Search is needed to refine a single class (i.e solve the problem <S,G> where S and G are both in the same class). To minimize w, we need to search in "small portions" of the graph. If we are using, for instance **compact** classes, then the time complexity to search within the class will never exceed $O(c^2)$. It is $O(c^2)$ when the "local" graph represented by the compact class is dense (Knowing that in a dense graph, E is of order $O(N^2)$, so $O(N+E) = O(N^2)$). Notice, also, that for many search algorithms, including breadth first search, w is influenced by d; larger values of d can give rise to larger values of w.

To conclude, when using compact classes, to **minimize w**, we should **minimize** c and **d**.

**Figure 4.7** A line

### 4.1.3 Examples

Consider the special case when c=d. Using compact classes, c=d means that the states in a class form a "line" in the graph (see Figure 4.7). If we use BFS to search, then the work required to solve a problem in the original space, with no abstraction, will vary between O(N) and $O(N^2)$ depending on whether the original graph is dense or not. If our search confines itself to paths within a class, then $w \leq c^2$. From the analysis (Equation 2):

$$\begin{aligned} TW &\leq w\, d^{A-1} \\ &\leq c^2\, d^{A-1} \end{aligned}$$

$= c \, c^{A}$

$= c \, N$

Thus, abstraction with c=d will only speedup search if the original graph is dense.

**Corollary**

If classes are all size 2 and each two states within a class are directly connected then no speedup is obtained if the original graph is not dense, and there is a speedup if it is.

**Proof**

This is just a particular case where d = c =2.

□

## 4.2 General approach to abstract

The strategy to follow when abstracting will be to use compact classes. In this manner we have a control over the total complexity. If, instead, we don't impose this restriction on classes, then it may happen that to solve from a start state to a goal state within the same class, we search through the whole graph, and thus allowing w to be of the same complexity as to solve in the original space.

The total work required to refine the most abstract solution down to the original level is, according to Equation 2 is, $TW \leq w \, d^{((\ln N)/(\ln c))-1}$. Using compact classes, the first term of the equation, w, requires that d and c should be minimized. The second term of

the equation requires that d should be minimized and c maximized. A common goal is clearly to minimize d. For c, we must look for an optimal value.

The approach to optimizing c that we will use in this thesis is to use algorithms that maximize c subject to a severe limit on d (typically $d \leq 3$). Our rationale is that optimizing d is the primary concern, and that the w term is in most cases small for classes with very small diameters. This strategy is a heuristic, it is not guaranteed to minimize TW. But we expect it to do well.

# 4.3 Complexity to abstract and store

## 4.3.1 Complexity to abstract

To abstract efficiently, not only should we care about the size of d and c, but also about the algorithm we use for abstraction. It should be of reasonable complexity. Recall that, in our approach, we don't look for the shortest path, so we would like our algorithm to be faster and require less space than algorithms for computing a shortest path.

Suppose that we have an algorithm that, at each level of abstraction, creates classes that each contain at least 2 states. Then the number of states of the abstract space is at most N/2. The next abstract space will have at most N/4 states and so on...

Suppose that the algorithm to do one level of abstraction has a complexity f(N), then the total abstraction complexity TAC, will be

$$TAC \leq f(N) + f(N/2) + f(N/4) + ... + f(1)$$

Our aim is for TAC to be at most $O(N^2)$. If f(N) is of order $O(N^2)$ then, from the

definition of the O-notation: there exists a constant b and $N_0$, such that for any $N \geq N_0$, $f(N) \leq b\,N^2$. This will imply

$$
\begin{aligned}
TAC &= f(N) + f(N/2) + f(N/4) + \dots + f(N/N) \\
&\leq b\,N^2 + b\,(N/2)^2 + b\,(N/4)^2 + \dots + b\,(N/N)^2 \qquad \text{for all } N \geq N_0 \\
&= b\,N^2\,(1 + 1/4 + 1/16 + \dots + 1/N^2) \\
&< b\,N^2\,(1 + 1/4 + 1/16 + \dots + 1/N^2 + \dots) \quad \text{(the infinite sequence)} \\
&< b\,N2\,(1 + 1/2 + 1/4 + 1/8 + 1/16 + \dots) \\
&= b\,N^2\,(1/(1-1/2)) = 2b\,N^2
\end{aligned}
$$

Thus <u>TAC is of order $N^2$.</u>

<u>Remark:</u>

The algorithms for creating a single level of abstraction will be described next chapter. Some are proved to be $O(N^2)$, others are conjectured to be. We have just shown that this is sufficient to guarantee that TAC, total cost of creating an abstraction hierarchy of height log N is $O(N^2)$.

### 4.3.2 Complexity to store

In All pairs shortest paths, to store the results, we need to store all paths. Knowing that we have $O(N^2)$ such paths, in the most "naive" way to store all the paths, the total memory space needed will be of order $O(N^2)$ times the average length of a path. The length of a path is considered (in this thesis) to be the number of states that constitute the path. It could be at maximum of order O(N). This implies that the total space complexity is of order $N^3$. There are better ways to do this, and which require $O(N^2)$ only to store all paths. However, in our approach, we don't store the paths. We store the abstraction

hierarchy and compute the paths as needed. The hierarchy of abstractions is - normally - represented in this way:

Level i: A list of operators and states, and the adjacency matrix for this level.

Recall that, the original space has N states and the abstract space at level i has $N/c^i$ states

The matrix at level i needs $(N/c^i)^2$ memory space.

The total space needed is:

$N^2(1 + 1/c^2 + 1/c^4 + \ldots)$, which is of $O(N^2)$.

If the graph is sparse (i.e $O(E) = O(N)$), then, using a list representation:

The total space needed is:

$N(1 + 1/c + 1/c^2 + \ldots)$, which is of $O(N)$.

Conclusion:

The space complexity to store the hierarchy of abstractions is of the same order as to store the original state-space only.

# Chapter 5

# Specific algorithms

## 5.1 Introduction

Recall, from the previous chapter, that our aim is to minimize the value ln(d)/ln(c) where c is the number of states within a class, and d is its diameter. The analysis was based on abstractions which provide compact classes, i.e classes where states are clustered together and thus, when searching, the path stays in the class. There are two cases studied in this chapter: the *primitive inverse* case and the *general* case. The primitive inverse case involves state spaces where all operators have primitive inverses. An operator f has a primitive inverse if and only if for all states S, if fS = S' then there exists an operator g such that gS' = S. For example, the formulation of the Tower of Hanoi puzzle, described in Chapter 2, is a primitive inverse state-space. The general case involves state-spaces which are not necessarily primitive inverse spaces. Abstracting a primitive inverse state-space is not as complicated as a general state-space. This is because, in a primitive inverse state-space, the distance from a state S1 to a state S2 is equal to the distance to get from S2 to S1, i.e the distance is symmetric. This means that once we know the distance from S1 to S2 we don't have to look for the distance from S2 to S1 since it is the same. In this case compact classes having small diameter are constructed more easily (see section 5.2). For example if S1 is directly connected to S2, then the distance is equal

to 2 in both directions. See Figure 5.1.

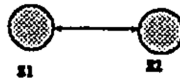


**Figure 5.1** Primitive inverse

However, in the general case, the distance is not symmetric and hence constructing compact classes having small diameter is more difficult. For example, if a state S1 is directly connected to a state S2, then the distance is equal to 2 in one direction but could be much greater than 2 in the other direction, as illustrated in Figure 5.2.

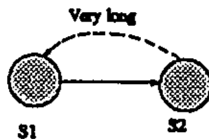



**Figure 5.2** Non primitive inverse

As mentioned in Chapter 4, the refinability constraint is satisfied when any two states within the same class are "reachable" from one another. Taking refinability together with the constraint that classes are compact, it follows that classes must be *strongly connected* subgraphs. As a first step in our abstraction algorithm, we therefore apply the strongly connected algorithm [Aho al, 1983].

The strongly connected algorithm is a classic application of depth first search. It partitions a given directed graph into its maximal strongly connected components, which are "portions" of the graph where all nodes within a portion are pairwise reachable. This means that for any two given nodes X and Y that belong to the same strongly connected

component, there exists a path from X to Y and a path from Y to X within the component itself (Figure 5.3). One way to find the strongly connected components of a graph G = (N,E) uses the transpose of G, which is $G^T = (N,E^T)$, where $E^T = \{(u,v) : (v,u) \in E\}$. That is, $E^T$ consists of the edges of G with directions reversed (Figure 5.3). The strongly connected algorithm has a time complexity of O(N+E).

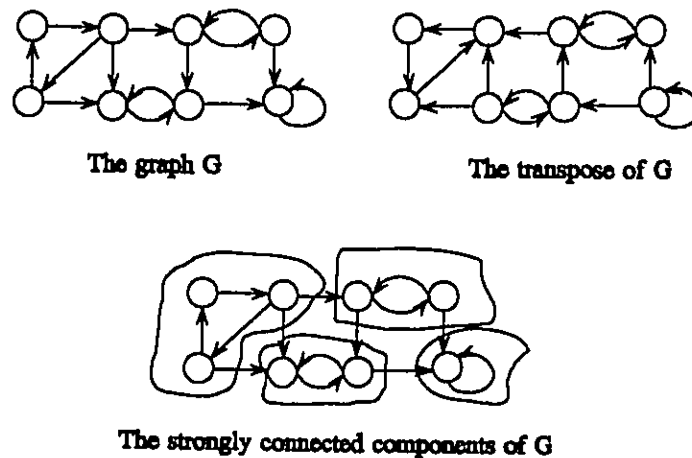



**Figure 5.3** The strongly connected algorithm

Using $G^T$, there are 3 major steps in the algorithm. Those steps include:

1) Applying depth first search to G.

2) Computing $G^T$

3) Applying depth first search to $G^T$ (in some different order) see [Cormen et al, 1990].

The final output will be a list of sublists where each sublist contains the strongly connected nodes, thus each sublist represents a strongly connected component of the

graph.

It is important to note, that for the primitive inverse case, the graph, representing the state space, is either strongly connected or disconnected. For the general case (or also for a disconnected graph that corresponds to a primitive inverse state space), after applying the strongly connected algorithm, the state space is now split into sub-spaces, each "acting" as a separate state-space, and will be abstracted separately.

All the algorithms described in this chapter use the graph relabelling strategy. The first thing to be done is change all the labels in the graph so they all differ from each other. If we have for instance k operators: $Op_0,...,Op_{k-1}$, and t arcs in the graph then all the arcs will hold the labels: $a_1,...,a_t$ (all integers) where if $a_i$ mod k = s then the arc labelled by $a_i$ represents actually the original label $Op_s$.

# 5.2 The primitive inverse case

According to the analysis in Chapter 4, we need an algorithm for abstraction that:
- is fast ( at most $O(N^2)$)
- strives to minimize d and optimize c

In the next sections, three algorithms will be shown with their advantages and disadvantages. Those algorithms are: the *Algorithm, the absorb *Algorithm and the double *Algorithm (the last two are Holte's).

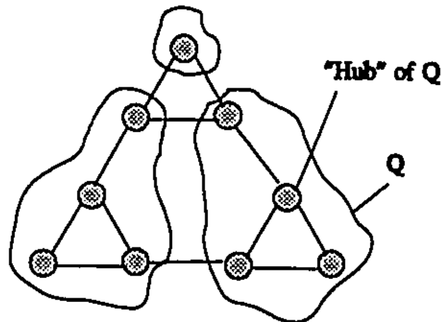



**Figure 5.4** The abstract space of TOH2 using the *Algorithm

## 5.2.1 The *Algorithm

The idea is to look for the most "dense" area in the graph and make it into a class. Recall that in a directed graph, the out-degree of a node means the number of edges diverging from that node, and the in-degree means the number of edges converging to that node. We use the term degree without distinguishing between out-degree or in-degree: they are the same in the primitive inverse case. A class will contain a "hub" state and all the states connected to it. The name star comes from the fact that the "hub" state and its neighbours will look like a star in a graph representation. Constructing classes in this way guarantees that d is 3. Any two states are reachable either directly (distance =2) or through the hub (distance=3). To try to optimize c, we choose as hubs the nodes with maximum degree. Here is the algorithm in detail:

(1) Choose the node with maximum degree in the graph
(2) Put that node and all of its neighbours in a class
(3) Remove all the elements of that class from the graph
(4) Check: if the graph is not empty then go to (1)
(5) Stop.

**Example** The 2-disk Tower of Hanoi (TOH2). See Figure 5.4.

□

### 5.2.1.1 Complexity analysis

Recall that the algorithm is applied many times until we reach a trivial space (or more precisely until we reach the space just before it). In this section we will show the complexity of the *Algorithm for building one level of abstraction only, and then present a conjecture for the whole complexity of the algorithm.

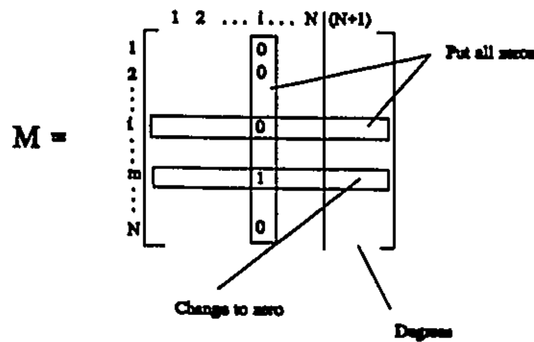



**Figure 5.5** The adjacency matrix

**Theorem**

The complexity to abstract a state space with N states is of order $N^2$.

**Proof**

If we choose as a representation of the graph the *adjacency matrix*, then this means that the whole graph is represented by matrix, say M, such that M is a square

matrix of size N×N that only has zeros and ones, and for all $a_{ij} \in M$ ($1 \leq i,j \leq N$):

$a_{ij} = 1$ if the ith node is connected to the jth node, ($a_{ij}=0$) if it is not.

In the primitive inverse case, the matrix is symmetric, but for the sake of simplicity we will consider the whole matrix. We will also augment M with a column where we put degrees and update them each time we look for a new "star", so M will be N×(N+1). See Figure 5.5.

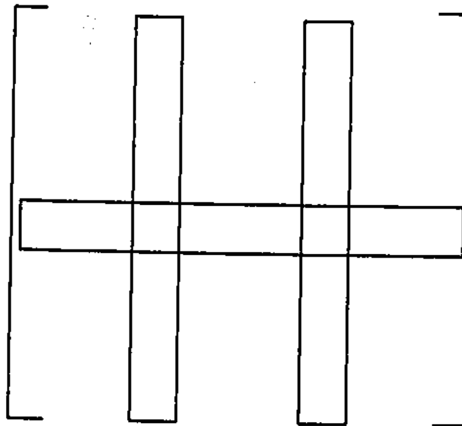


**Figure 5.6** The "inside rectangles" are only visited once

Calculating the maximum degree for the first star is just searching for the maximum number at the last column of M, it is of order N. Suppose that node m ($1 \leq m \leq N$) has the maximum degree, so node m will be the hub of the first star and the other elements of this star are just the neighbours of node m. We move along the row m from left to right and do the following in each column:

- if we encounter a "0" in column i just ignore it and proceed to i+1.

- if we encounter a "1" in column i then

  - Add node i to the list of nodes of the m-star (the star that has m as a hub).
  - Move along the column i and if there is a "0" in row j, ignore it and proceed to the next row. If there is a "1" in row j (i.e $a_{ji}$=1) then set it to zero and subtract 1 from the column of degrees at level j (i.e $a_{(N+1)j} := a_{(N+1)j} - 1$). This is to update the degrees (Figures 5.5 and 5.6).
  - Move along the row i (from 1 to N+1) and put zeros all over, so that node i is removed from the graph.

- Set the degree of node m (the hub) , $a_{m(N+1)}$ to zero.

Using a sparse matrix representation for M, "setting to zero" actually means deleting from M, therefore, all the "entries" visited in one star-calculation step won't exist for the next step. In all we have O(E) (E is the number of edges in the graph) entries, so the complexity is of O(E), which is no worse than $O(N^2)$. In this proof, the stopping-condition is when the whole matrix becomes nil.

□

In Chapter 5, we've mentioned that if f(N) (the cost to abstract once) is of $O(N^2)$ then so is TAC (total cost to abstract), but this is provided that the number of classes decreases proportionally to N, i.e there exists a constant k > 1 such that, if $N_i$ is the number of classes at the abstraction level i, then $N_{i+1} \leq N_i/k$ (i =1 corresponds to the level of the original space). For example, if we are sure that all the classes have at least 2 states, then the number of classes of the abstract space won't exceed N/2. In the *Algorithm, this is not obvious, because we could have some *orphans*. An orphan is a class that contains only one state from the original space (see Figure 5.7). However we believe that the number of abstractions needed to get from the original space to the trivial

space (with only one state) is a logarithmic function of N. Since we don't have a proof for that, we will leave it as a conjecture.

**Conjecture**

If the original space has N states then, after O(log N) abstractions, it will have 1 state.

□

#### 5.2.1.2 Searching and storing the result when using the *Algorithm

Storage requirements, in this case, are similar to those described for other algorithms (in Chapter 4). The only difference is to indicate the "hub" of each class at any level. This is linear in N, for storage.

Searching is the same as described in Chapter 5, except when it comes to search for a path from S to G where both S and G are in the same class. We look if there is a direct connection (to make it shorter) so the path is just SG, otherwise, we, use the path SHG, where H is the hub of the class containing S and G.

### 5.2.2 The Absorb *Algorithm

The motivation behind this algorithm is to make sure that the total abstraction complexity (TAC) is $O(N^2)$. The only problem with the *Algorithm is the existence of orphans that could make the number of abstractions relatively big and hence TAC could exceed $O(N^2)$. The absorb *Algorithm applies the *Algorithm first and then all the orphans are added to their neighbour classes. All of this during one abstraction. This will force all classes to contain two or more states guaranteeing the total complexity to

abstract to be $O(N^2)$. The only disadvantage of this strategy is that by introducing orphans to classes the diameter could increase from 3 to 4 or even to 5. See Figure 5.7.

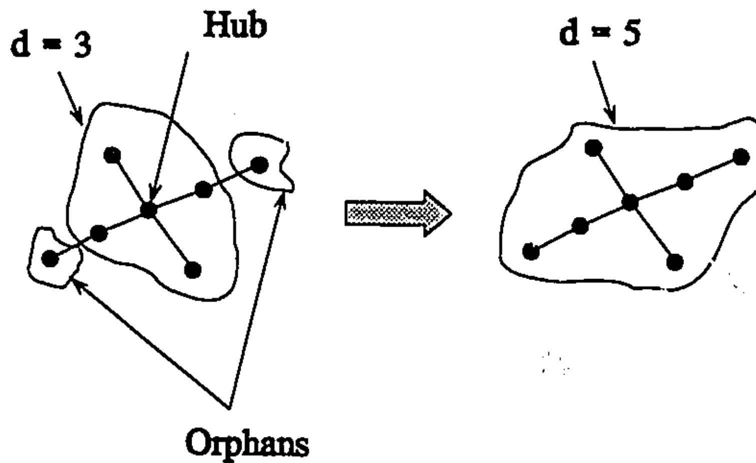



**Figure 5.7** The "absorb" phase

Here is the algorithm:

- Apply the *Algorithm, get a set of classes.
- Add every orphan to one of its neighbour classes.

**Example**

Applying the Absorb *Algorithm on TOH2 gives the result shown in Figure 5.8.

□

### 5.2.2.1 Complexity analysis

The algorithm has two steps: the first one was analysed in the previous section, and the second one is linear in the number of classes obtained. Since that number cannot

exceed N, the total complexity is just the *Algorithm's. TAC in this case is surely $O(N^2)$.

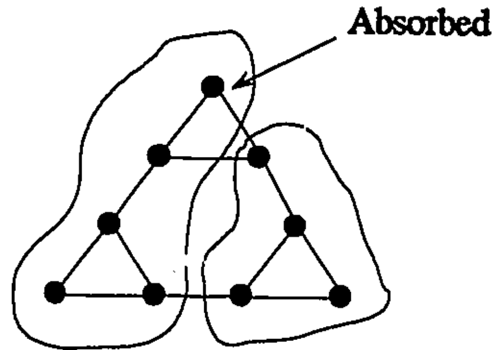



**Figure 5.8** The abstraction of TOH2 using "Absorb *Algorithm"

## 5.2.3 The double *Algorithm

The motivation behind this algorithm is the same as for the previous one, which is always to make sure that the number of states within any class at the abstract level is at least 2, so that TAC won't exceed $O(N^2)$. Although this was provided by the previous algorithm, we still want to minimize the value of d (recall that we want to minimize ln(d)/ln(c)).

The idea in this algorithm is to collect edges (which means two nodes connected to each other, call it the hub edge and call those nodes the "focii") and all the other nodes connected to the focii (Figure 5.9).

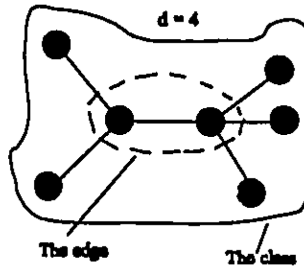



**Figure 5.9** The "Double *Algorithm" strategy

In this case we also look for the dense parts of the graph. The hub, in this algorithm, is the edge to which is connected the maximum number of nodes. This technique guarantees a diameter of at most 4.

Here is the algorithm for one abstraction:

(1) Generate a set of O(N) edges (see below)

(2) Choose the edge with the highest sum of degrees (the degrees of its two focii).

(3) Put in class the two focii and their neighbours

(4) Remove all the elements of that class from the graph

(5) Check: if the graph is not empty then go to (1)

(6) Stop.

<u>Explanation of (1):</u>

We choose any edge, remove it and its focii from the graph, and then we store the edge and its focii in a list. Then we do the same thing to the resulting graph. We repeat this process until the graph is empty.

**Example**

Applying the "Double *Algorithm" to TOH2 gives the result shown in Figure 5.10.

□

### 5.2.3.1 Complexity analysis

Normally, in step(1), we would generate all possible edges. But this could be of order $O(N^2)$ (recall that in a dense graph G = (N,E), E is $O(N^2)$). This will result in a complexity of order $O(N^3)$ to calculate the maximum degree at the first-step abstraction. It is for this reason that we only consider O(N) edges. In this case, the complexity analysis is similar to that of the *Algorithm. TAC is then $O(N^2)$.

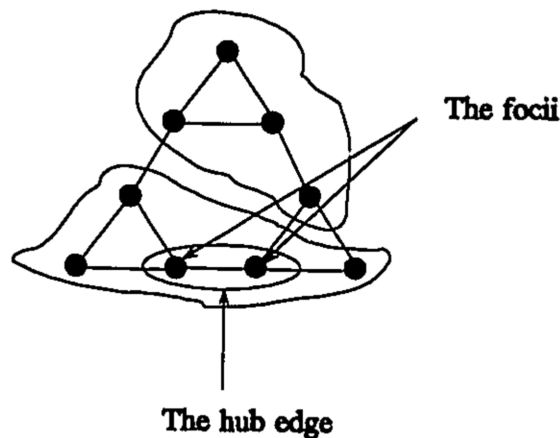



**Figure 5.10** The abstraction of TOH2 using the "Double *Algorithm"

## 5.3 The general case

As mentioned previously, the algorithm for abstraction must be "fast" and the abstract space has to be formed by classes, each having the maximum of states (the parameter c) and with the minimum diameter (the parameter d). The algorithms in this section are modifications of the *Algorithm. As before, the first step to take when abstracting is to apply the Strongly Connected Algorithm. This is done to ensure refinability. An abstraction is then formed by partitioning each sub-space. Our aim is to find the partition that minimizes ln(d)/(ln(c). The problem, in this case, is not as simple or straight forward as for the primitive inverse case. Putting two states together in the same class will normally require us to look for the distance between them in both directions and not only to see whether they are connected as in the *Algorithm.

### 5.3.1 The general approach for the general case

When putting two states into the same class, we must be sure that the distance between them, in both directions, is not large. The idea is to collect a set of closely connected states (at least in one direction), and then apply the strongly connected algorithm on this set. This produces a set of subclasses, some of which could be orphans. But if a class is not an orphan, it contains states that are all "close" to each other in both directions and thus guarantees a small value for d. The worst case happens when all the subclasses are orphans. Then the abstract space will be just the original and so there is need to proceed to the next level of abstraction. The "greedy" ways to follow this strategy are shown in the following algorithms.

## 5.3.2 The strong *Algorithms

The first algorithm is called the **basic strong *algorithm**. Although, practically (see Chapter 6), it is not very helpful, it is the starting point for the subsequent algorithms. Not only this, but when the graph is "dense" or many arcs have inverses, the algorithm behaves very well. On the other hand when the graph is not "dense" or almost free of cycles, the algorithm terminates very quickly without abstracting the graph. The idea is very similar to the primitive inverse case with the exception of applying the strongly connected algorithm to every star. Here is the algorithm in detail:

(1) Choose the node with maximum out-degree in the graph
(2) Separate that node and all of its neighbours and all edges connecting them from the graph
(3) Apply the strongly connected algorithm to this local part of the graph (i.e all the nodes and their interconnections) => get classes.
(4) Remove all the elements of each class from the graph
(5) Check: if the graph is not empty then go to (1)
(6) Stop.

Recall that this is one step of abstraction. Before moving to the next level of abstraction we first count the number of states: if they are equal to those of the current space being abstracted, then we stop. Notice that this algorithm could produce many levels of abstraction. A method to overcome this problem will be explained after showing the other algorithms since they all face this problem.

There is a better version of the strong *algorithm which increases the chance of constructing a non-trivial abstraction. That is, when abstracting, the probability of getting classes which are not orphans is greater than when using the basic strong *algorithm. This

algorithm is called the **in-out strong *algorithm.** It is the same as the previous one except that it considers both in and out degrees in choosing the hub of a class and its neighbours. Notice now that the neighbours of a node will be anything directly connected to it, whether with an arc converging to the node or with a diverging one.

**Example** The non-primitive inverse Towers of Hanoi problem.

The only difference in the formulation of the problem is when dealing with N disks we only need N operators: 1,2,...,N instead of 2N operators: -N,...-1,1,...,N in the primitive inverse case. The move of any disk will be possible in one direction: clockwise. The graph obtained is strongly connected, but no arc has a primitive inverse. See Figure 5.11.

Applying the basic strong *algorithm to TOH2 (the non primitive inverse) will leave the state space unchanged. The result we get, after applying the in-out strong *algorithm to TOH2 , is exactly the standard abstraction of TOH2.

□

The third algorithm within the strong * family is the **double in-out strong * algorithm.** The idea, here, is to move 2 levels out from a node to count its degree. That is, instead of considering its neighbours only - to construct a class -, it considers also the neighbours of its neighbours (the second-level neighbours), and counts all of them to find the "degree" of the node. Then, as for the previous algorithms, the strongly connected algorithm is applied to the class constituting of a hub (having the maximum in-out degree), its neighbours and its second-level neighbours.

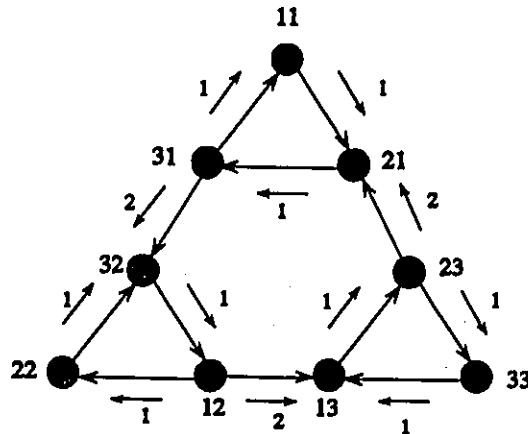



**Figure 5.11** The non primitive inverse TOH2

## Complexity analysis

The analysis is very similar to that for the primitive inverse case. There are two major things to take into consideration. The first is the local application of the strongly connected algorithm within each class. This doesn't change the order complexity of the algorithm since the strongly connected algorithm itself is linear in the number of states and arcs. Within a class, visiting all its states plus applying the strongly connected algorithm on them will be the **same**, in terms of order of complexity, as just visiting them. The second difference is that the number of levels of abstraction could be linear in the number of states N. For this reason, when we abstract, we "move" from a level to another only when the number of states (classes) of the abstract space is less than half the number of states of the current space. We do not "move" to new level if the majority of states have been left as orphans. If this happens, we try to expand the abstraction at this level. To do this, it is necessary to consider only the states "directly" connected to those already affected at the current abstraction level. Thus the total complexity to abstract, TAC, is "believed" to be of order $N^2$. It is the same as the * algorithm (for the primitive

inverse case), which wasn't given a precise proof.

### 5.3.3 The strong edge algorithms

There are also three of these. The first one, called **the strong edge algorithm,** is just the double *algorithm as for the primitive inverse case with the addition of applying the strongly connected algorithm on the classes formed. In this case, only out-degrees are considered. The second algorithm: **the in-out strong edge** has the extra feature of considering both in and out degrees. Finally, the third algorithm, which is the **double in-out strong edge** considers two levels when including the neighbours of the hub edge focii into a class. The complexity is of the same order as the strong *algorithms.

Remark

Notice that all complexity results in chapters 4 and 5 are true if we change $O(N^2)$ by $O(E)$.

# Chapter 6

# Experimental work

## 6.1 Motivation

Experimental study is needed for many reasons. First, the actual abstractions do not satisfy the premises of our analysis such as c and d constant for all classes. Second, our heuristics do not guarantee the best choice of c and d. Third, the complexity of the *algorithm was not proven. Finally, the experimentation led to new approaches which could be studied and analyzed and then theoretically proved. For example, the idea of using the double *algorithm was mainly brought about because the previous approaches were not good enough when experimentation was done.

The state-spaces used in this experiment are those corresponding to a set of known puzzles: the Towers of Hanoi, the Navigation Problem, the Five Puzzle, the Arrow Puzzle, the Missionaries and Cannibals and the Water Jug Problem. The variety of spaces will help comparing algorithms to each other. Those puzzles cover almost all state-space possibilities described in this thesis which are: primitive inverse and non-primitive inverse, connected and disconnected graph representation. They all share one important feature, which is that their graphs are such that O(E) = O(N). For example, the graph representing the Towers of Hanoi has the following properties. If the number of disks is

D, then the number of nodes $N = 3^D$ and the number of edges is $3(3^D-1)$ (in the primitive inverse case). This means that $E = 3(N-1)$ and thus $O(E) = O(N)$. The time complexity to search in the original space is thus of order N for all puzzles. The formulations of the puzzles are given in Appendix A.

# 6.2 Parameters measured and their significance

An experiment consists of creating a hierarchy for a given puzzle. Then a hand-picked state plus three other randomly chosen states from the original space, are all considered to be initial states and the problems to be solved in this experiment are those where the initial state is one among the four states mentioned, and the goal states are all possible states in the space. This means that when the space has N states, we solve 4N problems.

To summarize the results of an experiment we measure the following six parameters[2]:

**1) Dec-time:**

The time it takes to create the abstraction hierarchy. This measures the computational complexity of an abstraction algorithm.

[2] We also measure two other parameters but which are not as important as the rest. Those are sumlength: the sum lengths of all 4N solutions, and ratioTsum: the ratio of Arcs_traversed to sumlength. Those are shown in Appendix C.

**2) Search-time:**

The time it takes to solve all the 4N problems. This indicates whether the abstraction hierarchy is good or bad.

**3) Arcs-traversed:**

This parameter indicates the total number of arcs traversed (in finding a solution to a problem) when solving all the 4N problems. This is a more accurate measurement of the "goodness" of an abstraction hierarchy. Search-time is not as accurate because it could include some extra overheads and programming inefficiencies.

**4) Average-Solution-length:**

Each solution has a certain length. We have 4N solutions in all and the average of their lengths is calculated. Shorter solutions are preferable.

**5) The average c:**

This parameter shows the average value of c for all levels of abstraction.

**6) The "Shape" of the hierarchy of abstractions:**

This measures the number of levels of abstraction, the number and size of classes for each level (the size is in terms of number of states that belong to the original space). Although, practically, this won't help much in deciding whether an abstraction is good or bad, it does indicate how "balanced" our abstractions are.

Recall that usually (not only in this thesis), complexity analysis is based on assumptions; e.g., classes are of equal size, the number of classes at each level is the same, etc... By measuring shape, we can see how close we are to satisfying those assumptions. The other advantage is that the "shape" could indicate how to improve our algorithms. For instance, when we first applied our algorithms for the general case, we noticed through the shape that the number of levels was too big. So we added a constraint forcing the number of states to be less than half of the number of states at the previous level. The bad or good results (directly measured from searchtime, Arcs-traversed or average-solution-length) can often be explained by analyzing the "shape" of the hierarchy.

## 6.3 Results and discussion

This section is divided into two sections: the first describes the results obtained in the primitive inverse case, the second deals with the general case.

### 6.3.1 The primitive inverse case

In this case, the state-spaces used in the experiments are for the following puzzles: the Towers of Hanoi, the second version of the Navigation Problem (in this version, the robot can turn in both directions: clockwise and counterclockwise. It can, also, move forward and backward), the Five Puzzle, the Arrow Puzzle and the Missionaries and Cannibals. For the sake of clarity, we will only discuss, in this chapter, the results produced by the Towers of Hanoi. All results, for all puzzles, are shown in Appendix B. Appendix C shows one example for each puzzle.

In this experiment, we used a set of algorithms mainly belonging to the star family. Those algorithms are:

**1)The *algorithm:**

Described in Chapter 5, named "Star" in the graphs which follow.

**2) The absorb *algorithm:**

Described in Chapter 5, named "Star &Abs" in the graphs which follow.

**3) The double *algorithm:**

Described in Chapter 5, named "Double Star" in the graphs which follow.

**4) The random star:**

Same as the *algorithm, except that instead of choosing as the hub the state with the maximum degree, it chooses hubs randomly. This algorithm is used to check whether maximizing c is beneficial. It is named in the graphs as "Rand Star".

**5) The random edge:**

Same as double *algorithm, except that instead of choosing as the hub the edge with the maximum degree, it chooses hubs randomly. Named in the graphs as "Rand Edge".

**6) The edge algorithm:**

This algorithm abstracts by just collecting connected pair of states together at each level of abstraction. it is included to see what happens when c=d=2. It is named "Edge" in the graphs.

**7) Original:**

The original, unabstracted space. It is named "Original" in the graphs.

**8) Standard:**

This only applies to Towers of Hanoi. It is the standard, hand-crafted abstraction (so the time to abstract for this case and also for the original space is to be ignored). The goal is to compare our general-purpose abstraction heuristics to the best known abstraction for Towers of Hanoi. It is named "Standard" in the graphs.

<u>Explanation of the graphs</u>

There are four different pairs of graphs and a single last one. In all of the graphs, the x-axis is the number of disks (from 1 to 6), which is $\log_3 N$ (N = the number of states). The y-axis of the first pair of graphs measures the number of arcs traversed. The second measures search time. The third measures abstraction time. The fourth measures the average solution length. Finally, the last graph measures the ratio of the solution length to the optimal solution length. The two graphs in each pair give results for different algorithms, but otherwise give exactly the same information. The first in each pair includes the algorithms "Star & Abs", "Rand Edge", "Double Star", "Original", "Rand Star" and "Star". The second includes again "Star & Abs" and "Original" and also "Edge" and "Standard". This is to avoid having too many curves in the same graph and at the same time to separate those relatively "extreme" cases from the first graph.

In Figures 6.1 and 6.2 the y-axis has units of one million (arcs traversed). The curves for all abstraction algorithms are below the one plotted by original. This means that problem-solving using abstraction always involves traversing fewer arcs than problem-solving in the original space. Most of the star algorithms are "close" to each other and much better than "Original". Furthermore, the "gap" between them and "Original" gets wider as the number of disks increases. This means that ratios are not constant, and thus they don't have the same complexity. The best among them is "Star & Abs", it is quite close to "Standard", the best known abstraction for Towers of Hanoi.

The algorithm called "Edge" also shows better performance than "Original". Recall that it represents a particular case where c=d=2.

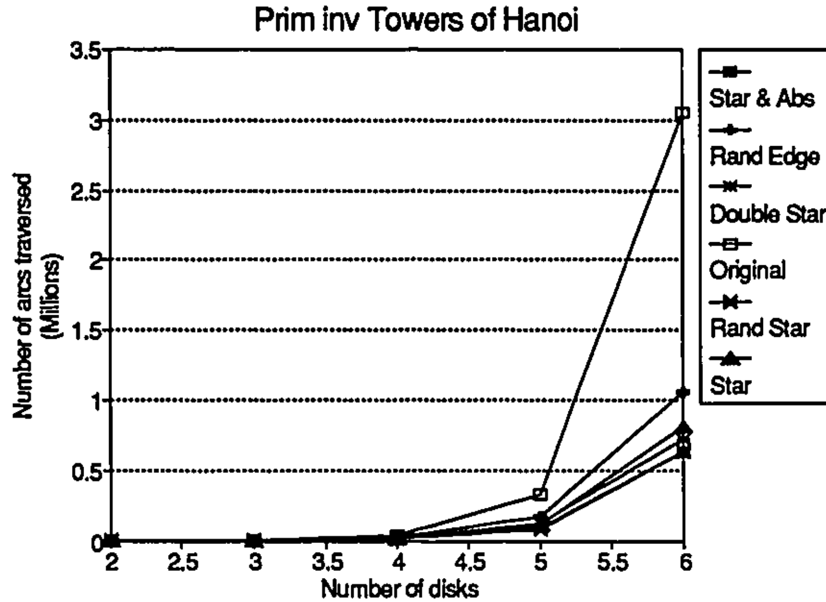



**Figure 6.1** Number of arcs traversed vs Disk number (1)

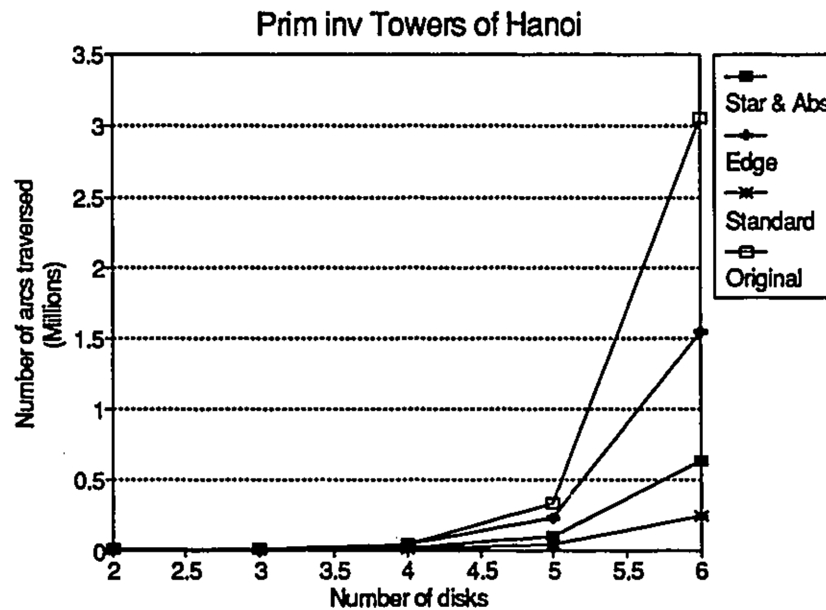



**Figure 6.2** Number of arcs traversed vs Disk number (2)

In the two graphs showing the time to search, the scale of the y-axis is not the same. It is 1 second in Figure 6.3 and 1 thousand seconds in Figure 6.4. Generally, the graphs are similar to the ones showing the number of arcs traversed. The differences in order are due to extra overheads of the system and also to some programming inefficiencies. For example, "Edge" looks worse than "Original" in Figure 6.4. This is due, apart from programming inefficiencies, to the fact that "Edge" builds a hierarchy of many levels compared to other algorithms, so in this case the overhead of translating a solution from one level to another becomes significant.

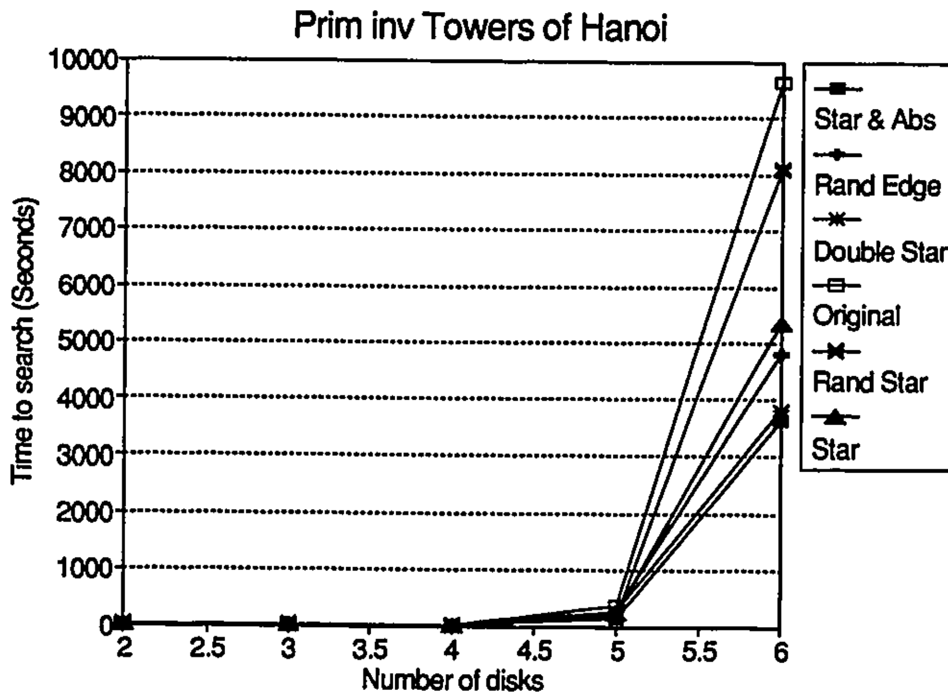



**Figure 6.3** Search time vs Disk number (1)

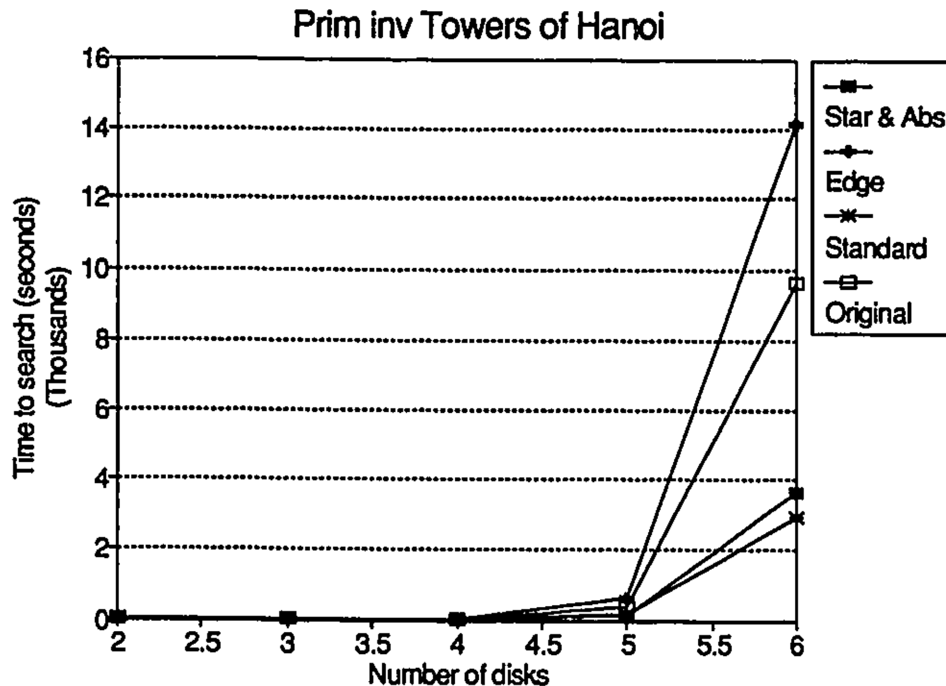



**Figure 6.4** Search time vs Disk number (2)

Abstraction time varies according to the algorithm used (see Figures 6.5 and 6.6). The more complex heuristics we add (e.g calculating the maximum degree), the more time the system takes to abstract a state space. Notice, in the figures that "Original" has an abstraction time different from zero. This is due to the process of renaming all edges in the graph. This was needed for programming purposes (keeping the same data structure for all graphs used). "Double Star" takes an unacceptably long time to abstract, but this is due to programming inefficiencies, not its inherent complexity. This time ought to be similar to the time of "Rand Edge".

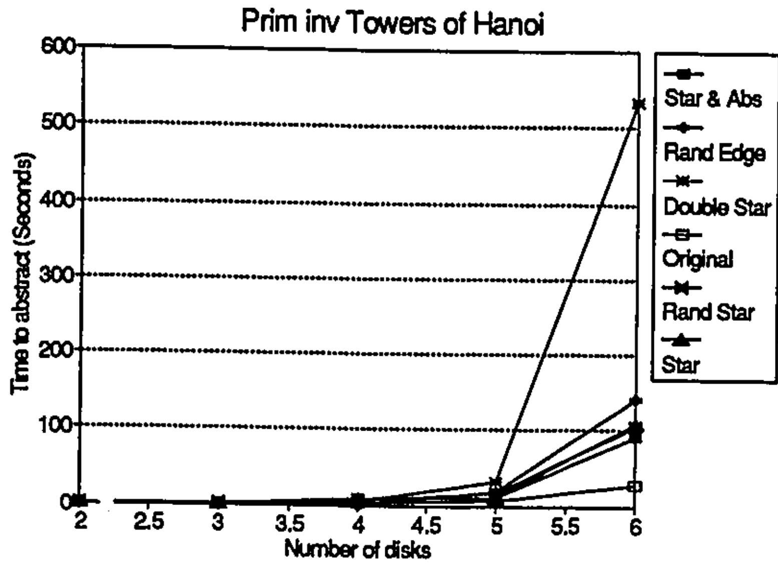



**Figure 6.5** Abstraction time vs Disk number (1)

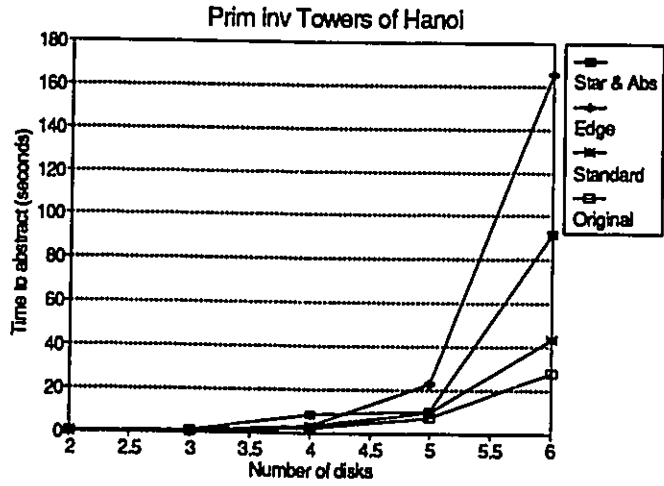



**Figure 6.6** Abstraction time vs Disk number (2)

Finally, the solution lengths are longer than optimal (obtained when using "Original"). See Figures 6.7 and 6.8. Nevertheless the solutions found are not excessively long. The ratio of the average length to the optimal length is shown in Figure 6.9. An increasing ratio indicates that solutions grow worse and worse as the size of the space increases. This is undesirable, but most of the algorithms have this property. Judging from this data, the algorithms with the most slowly growing ratio are "Star &Abs" and "Double Star".

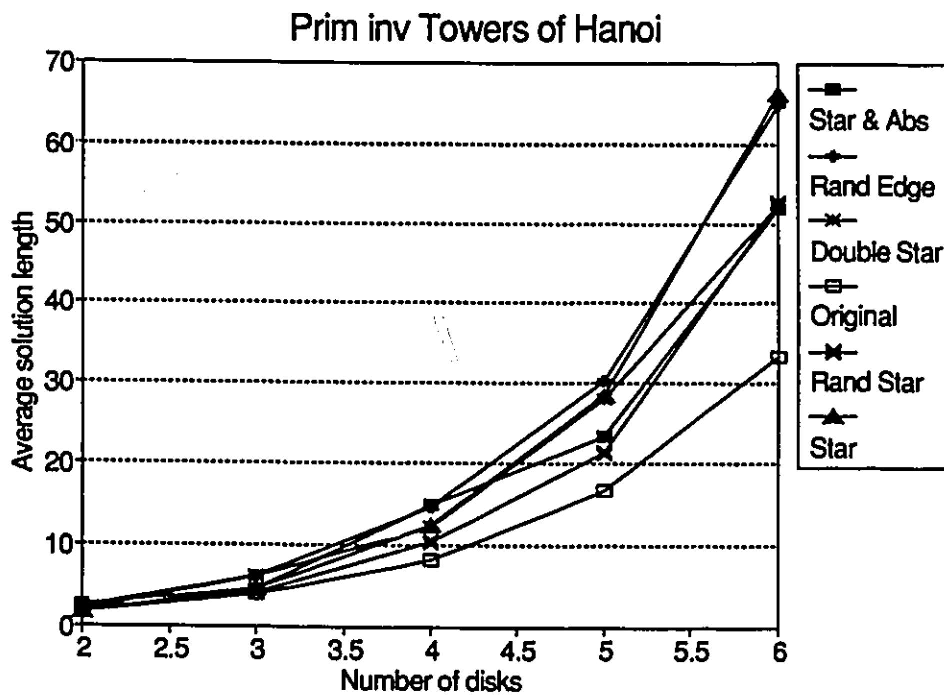



**Figure 6.7** Average solution length vs Disk number (1)

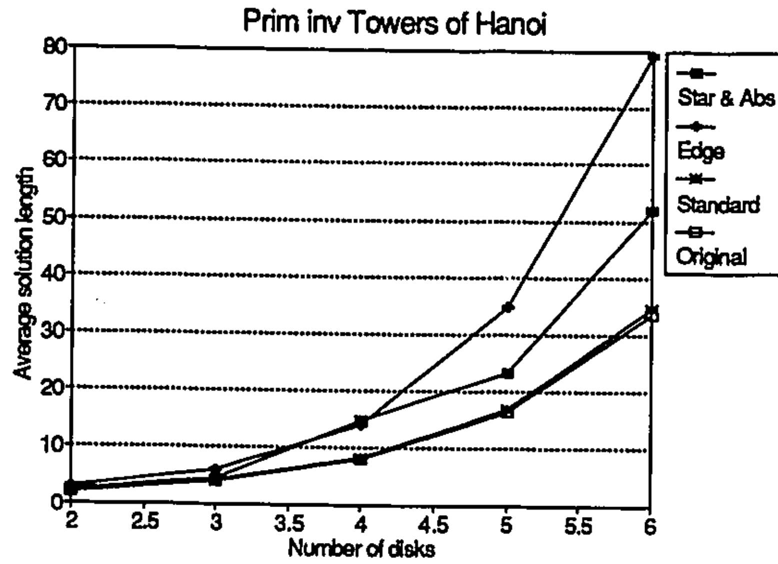



**Figure 6.8** Average solution length vs Disk number (2)

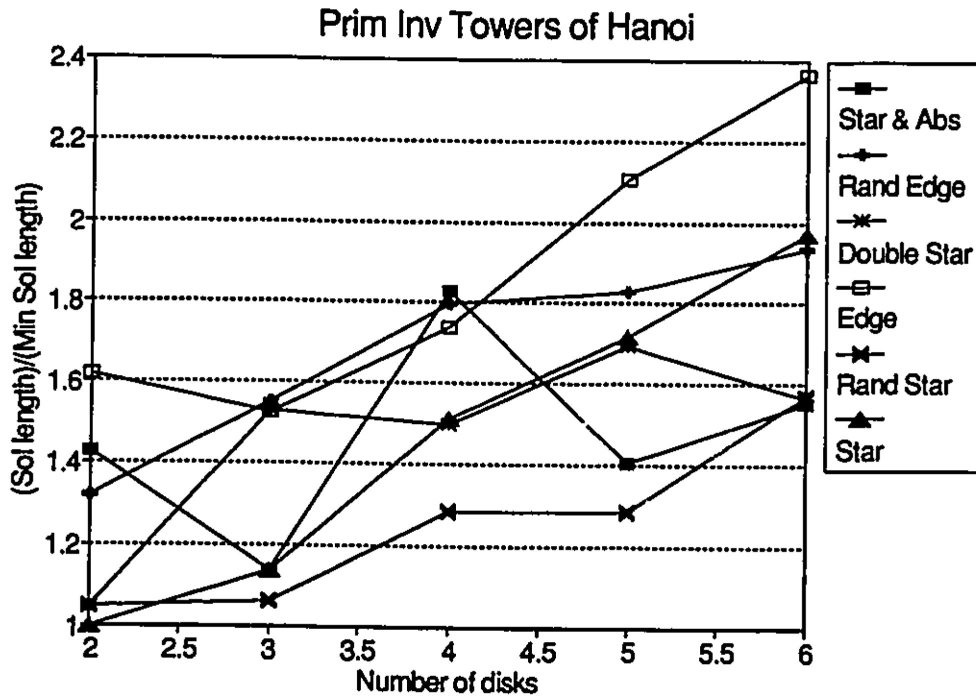



**Figure 6.9** Ratio of the solution length by the optimal solution length

If we consider, for example, "Double Star", the ratio grows from 1.05 to 1.53, then decreases to 1.50, and then increases (slower) to 1.69, and finally decreases to 1.56.(see Figure 6.9). This is relatively close to the behaviour of a constant function.

<u>Conclusion:</u>

The absorb *Algorithm behaves very well, so the extra step taken to absorb (extra step to the *algorithm) is useful in practice, as well as in theory (it guarantees a logarithmic number of levels of abstractions and at the same time strives to maximize c). Also, the double *algorithm and the *algorithm both behave well.

Recall that our aim is to have the sum of costs to abstract and solve less than solving directly in the original space. In all we are solving 4N problems. If we take the case of "Star & Abs": time to abstract is about 90 sec, time to solve N problems is about 3600/4 = 900 sec. The sum is then 990 sec. Solving the same N problems for "Original" takes about 9600/4 = 2400 sec. For "Star" the sum is about 1400 sec, and for "Double Star" it is 1600 sec.

Compared to random algorithms, the *algorithms are better in all ways. The only exception is in abstraction time, in which they are all quite similar. This confirms that maximizing c will improve the time complexity as well as making solution lengths shorter. Compared to "Standard", the *algorithms are not very "far" (especially "Star & Abs") in terms of number of arcs traversed, but for solution lengths "Standard" looks almost optimal, whereas the *algorithms are away from optimal.

For other puzzles, results are similar. The exception is, the best algorithm for one puzzle is not necessarily the best for other puzzles. For example, in Towers of Hanoi "Star & Abs" was considered the best, whereas "Star" was the best for the 5-puzzle. Also

the improvement is not the same for all puzzles. For example, the number of arcs traversed in the best algorithm in the 5-puzzle is about four times less than those in "Original". However in the missionaries and cannibals problem, the number of arcs traversed in the best algorithm is just half of those in "Original". But solutions produced by all algorithms were all almost optimal. See Appendix B.

### 6.3.2 The general case

In this case, the state-spaces used in the experiments are those for the following puzzles: the Towers of Hanoi (the non-primitive inverse version), the Navigation Problem (also, the non-primitive inverse version) and the Water Jug problem. Also, as in the preceding section, for the sake of clarity, we will, only discuss, in this section, the results produced by the Towers of Hanoi (the non-primitive inverse version). All the other results, for the other puzzles, are shown in Appendix B.

In this experiment, we, also, used all the algorithms in the strong star family, as described in Chapter 5. Those algorithms are:

**1)The strong *algorithm:**

Named in the graphs as (Strong *).

**2) The strong in-out *algorithm:**

Named in the graphs as (Strong I/O *).

**3) The double strong *algorithm:**

Named in the graphs as (Double strong *).

**4) The double strong in-out *algorithm:**

Named in the graphs as (Double I/O strong *).

The Original state-space (named **Original**) and the standard Towers of Hanoi abstraction (**Standard**) are also shown in the graphs.

In the general case, it is harder to find good abstractions than in the primitive inverse case. For the number of arcs traversed, the results are not that impressive although they are still better than solving in the original space. Recall that we use a conservative method of abstracting that guarantees that the worst abstraction we can have is the original space itself. See Appendix C.

Explanation of the graphs

In this case, there are only three different graphs. The first is for the number of arcs traversed, the second is for search time and the third measures abstraction time. Each graph includes the algorithms "Double I/O Strong *", "Strong *", "Standard", "Original", "I/O Strong *" and "Double Strong *".

In Figure 6.10 the y-axis has units of one million (arcs traversed). The curves for all abstraction algorithms are below the one plotted by original. The only exception is in the "Strong *", since according to the "shape" (see Appendix C), the algorithm left the original space unchanged. Those results mean that also in the general case, problem-solving using abstraction always involves traversing a number of arcs less or equal to those traversed when solving in the original space. Unlike the primitive inverse case, the strong star algorithms are not "close" to each other, but, generally are better than "Original". Furthermore, the "gap" between them and "Original" gets wider as the number of disks increases. This, as in the primitive inverse case, means that ratios are not

constant, and thus they don't have the same complexity. In terms of arcs traversed, the best among them is "I/O Strong *".

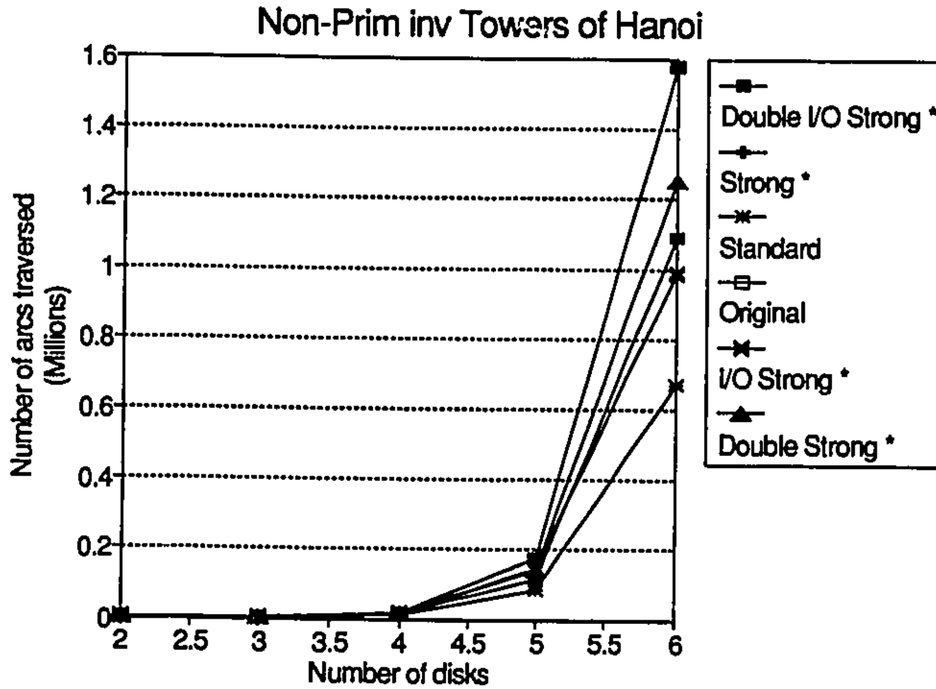



**Figure 6.10** Number of arcs traversed vs Disk number

In the graph showing the time to search (Figure 6.11), the scale of the y-axis is 1 thousand seconds. Normally, the graphs should be similar to the ones showing the number of arcs traversed. In this case, there are more differences in order than in the primitive inverse case. For example, "Standard" which is the best in terms of arcs traversed is worse than "Double Strong *" in search time. Also "Double Strong *" is the second worse in terms of arcs traversed. According to its "shape", "Double Strong *" has fewer levels of abstractions than "Standard" and classes have much larger size (see Appendix B). Refinement is more frequent in the case where classes are small (for "standard", c = 3) and thus requires more overheads. Notice that in this case the overhead of translating a solution from one level to another becomes more significant than in the primitive inverse case. This is because the "w term" is more significant in the non-primitive inverse case.

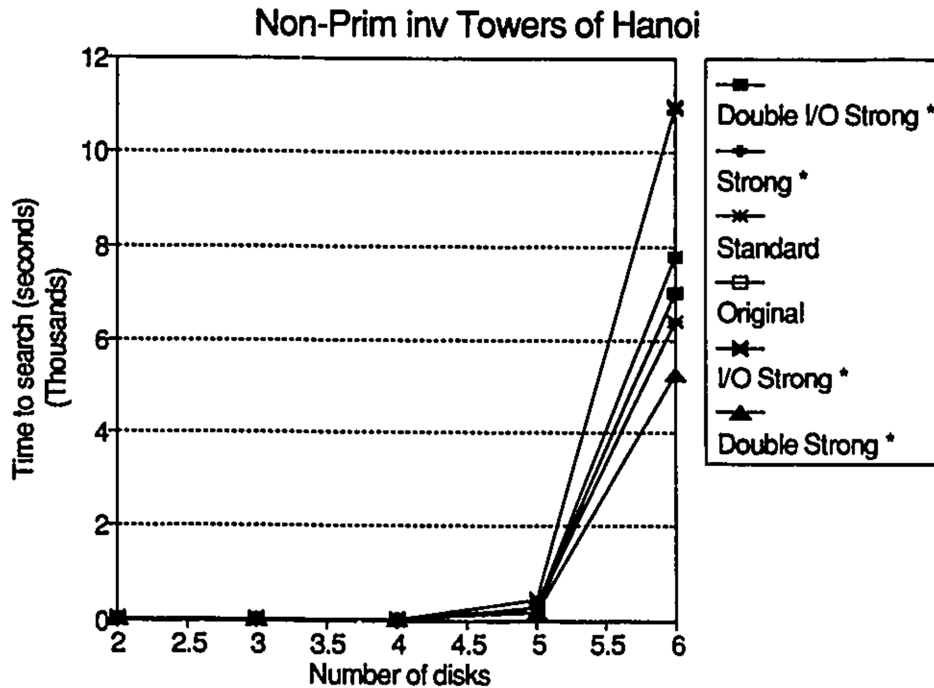



**Figure 6.11** Search time vs Disk number

The abstraction time increases in this case. This is due to the more complex heuristics we use and also to the use at each level of abstraction of the strongly connected algorithm. See Figure 6.12. "Double I/O Strong *" (as was the case for "Double Star" in the primitive inverse case) takes an unacceptably long time to abstract, but this is due to programming inefficiencies, not its inherent complexity. This time ought to be similar to the time of "I/O Strong *".

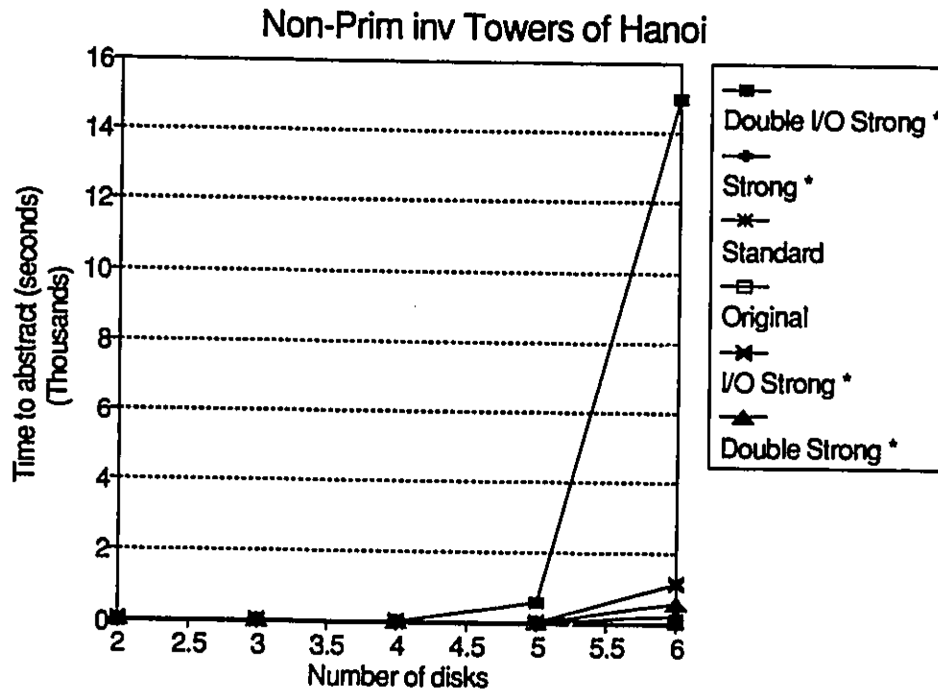



**Figure 6.12** Abstraction time vs Disk number

Finally, for the solution length, they were all optimal. This is very particular to this puzzle and cannot be taken as a general result.

Conclusion:

It is hard, in this case, to "judge" these algorithms. Each one could be the best for one measurement and the worst for another measurement. Even "Standard", although it is the best for arcs traversed is only the second best for search time.

In terms of search time all the algorithms, except "Double Strong *", are slower than "Original". This indicates that abstraction has failed in this case.

In contrast to the primitive inverse case, the strong *algorithms, compared to "Standard", are not very different in terms of search time, are the same for solution lengths, but differ a lot in terms of number of arcs traversed.

For other puzzles, results are similar. But most of the algorithms produce the original space. In the case they don't produce the original space, solutions are generally longer and problem-solving using abstraction always involves traversing fewer arcs than problem-solving in the original space. See Appendix B.

## 6.4 Conclusion

The main conclusion from this experiment is that abstraction "wins" in the primitive inverse case whereas it "fails" in the general case. This is because the cost to abstract plus the cost to solve was less than the cost to solve in the original space for the primitive inverse case. However, it was the opposite for the general case. This failure is not due to a fundamental limitation of abstraction, but just a weakness in our current approach.

# Chapter 7

# Literature review

This chapter describes the related work on abstraction in problem solving. It has two main sections: The first shows general approaches that deal with abstraction. The second section presents closely related work.

## 7.1 General Approaches dealing with abstraction

This section is divided into two sub-sections: How to use abstractions for problem solving and how to generate abstractions for problem solving.

### 7.1.1 The use of abstractions for problem solving

The idea of decomposing a problem into a number of simpler subproblems was initially introduced by [Polya, 1945] and others. Afterwards several researchers have shown that significant reductions can be obtained by dividing a problem into subproblems [Newell et al, 1962], [Minsky, 1963]. The analyses and proofs of this strategy assume a problem is divided into small, equal-sized and independent subproblems that can be solved without backtracking. Recall that we never do backtracking in our system.

The use of abstractions for problem solving can be done in different ways. An

**abstraction space** can be either a simplification or a reformulation of the original space. Notice that in our approach we have both: simplification is obtained through our way of abstraction and reformulation is obtained through graph relabelling. A well-known way of representing a problem was given by the STRIPS system [Fikes and Nilsson, 1971], [Fikes et al, 1972] which is based on the First Order Logic. ABSTRIPS [Sacerdoti, 1974] employs abstract problem spaces for hierarchical problem solving. The abstraction spaces for a problem domain are represented by assigning criticalities: numbers indicating the relative difficulty to the preconditions of each operator and ABSTRIPS works on the "difficult" goals first. When a solution to a given problem cannot be found, the system backtracks to find a different abstract solution. The same basic approach is also used in hierarchical PRODIGY [Knoblock, 1991]. It is based on removing properties (literals) instead of simply dropping preconditions (as ABSTRIPS does). This allows the simplification of the goals of a problem in an abstract space. It has also the ability to maintain the structure of the problem solving trace so that it can easily backtrack across abstraction levels. There is no attempt to avoid backtracking.

Another approach to using abstractions for problem solving employs **abstract operators**. In this case, there is no notion of solving a problem at one level of abstraction before refining the plan to the next level. Recall that in hierarchical problem solving, an abstract plan is built and then refined by selectively expanding its individual operators into successively more detailed ones. In the case of abstract operators, refinement is done using a set of action reductions [Yang, 1989], which specify the relationship between an abstract operator and the refinements of that operator. 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. Although there are attempts (e.g. goal protection by using a kernel in NOAH [Sacerdoti, 1977]) to solve this problem, the problem is still difficult to handle efficiently. Problem solvers that employ abstraction problem spaces avoid this problem by expanding completely the plan at each level before moving to next, and that's what we do.

Another way of using abstractions allows operators to be combined into macro operators to form a **macro problem space** [Korf, 1987]. The macros are defined by operators in the original problem, which implies that translating the solution in the macro space into a solution in the original space is easy: just replace each macro by its definition. The difficulty with this strategy is in finding a good set of macro operators.

Abstractions can also be used to create a set of **admissible heuristics**. An example of such heuristics is the solution length [Pearl, 1985]. An optimal solution path for any resulting abstracted problem gives a lower bound on true distance to goal. This bound can be used as an admissible evaluation function for guiding the base-level search [Mostow and Prieditis, 1989].

### 7.1.2 The generation of abstractions for problem solving

Similarly, there are different approaches to generating abstractions for problem solving. ABSTRIPS uses a short-plan heuristic which separates the details from the important information. It places the static literals, literals whose truth value cannot be changed by any operator, in the highest abstraction space. It places literals that cannot be achieved with a "short" plan in the next highest abstraction space. Finally, it places remaining literals at lower levels corresponding to their place in the user-defined partial order. It determines whether a short plan exists by assuming that the preconditions higher

in the partial order hold and attempts to show the remaining conditions can then be solved in a few steps. This is actually an automatic production of a three-level abstraction hierarchy, with the static literals at the top of the hierarchy, the "important" literals next, and the details at the bottom [Knoblock, 1991]. Any further refinement of levels comes from the user-defined abstraction hierarchy. ALPINE [Knoblock, 1991] produces more effective abstractions with less knowledge than ABSTRIPS, but it is still based on ABSTRIPS itself. Abstraction hierarchies generated by ALPINE have "the ordered monotonicity property", a property which guarantees that every possible refinement of an abstract plan will leave the conditions established in the abstract plan unchanged. In other words, it means that once a goal is satisfied at one level, it will not be violated while refining a plan at a lower level, but it may be necessary to backtrack to the more abstract level if it cannot be refined. One drawback is that "ALPINE is very sensitive to the representation of operators and this could limit the granularity of the abstractions" [Knoblock, 1991]. Also the algorithm used may generate constraints that are unnecessary to ensure the ordered monotonicity property. ABSOLVER [Mostow and Prieditis, 1989] uses a set of abstraction transformations to create abstractions of a problem. Those transformations include dropping preconditions, dropping goals and dropping predicates from the problem space. Abstractions are selected using a generate-and-test procedure.

There are systems that learn sequences of subproblems through experience and then apply them to future episodes. This approach is called **problem reduction** [Amarel, 1968] where a problem is replaced by a number of subproblems where search is easier to accomplish. For example in game playing, the system learns "more" whenever it gets defeated. The disadvantage of this approach is that each problem may require a different set of problem reductions. Notice that in this case search for an abstract solution does not take place.

When sequences of operators are combined to form macro operators or when

objects (e.g states) are also combined into macro objects, an abstraction is obtained by **aggregation**. Our system, which combines states into a single state (e.g. the star algorithm), is an aggregation algorithm. Most of the systems that use macro operators simply add those operators to the original space. Although this can reduce the solution depth, it increases the branching factor and this could reduce the overall performance [Minton, 1985].

The last approach to be briefly described in this section is the **goal ordering** approach. A famous example that employs this strategy is GPS [Newell and Simon, 1972]. It is a means-ends analysis problem solver which uses a table of differences to select operators to focus the search. GPS requires an ordering of the differences between the initial state and the goal. Niizuma and Kitahashi follow a relatively similar approach with the addition of using equivalence relations between states [Niizuma and Kitahashi, 1985]. STATIC [Etzioni, 1990] is a system which statically analyses the problem space definition to generate a set of control rules and then guide the search in the original space. Those control rules are generated for PRODIGY to solve the problem.

## 7.2 Closely related work

This section describes methods of abstraction which are similar to our approach (especially in the use of state-spaces in the concretization approach). We will be describing three approaches: **concretization** [Prieditis and Janakiraman, 1992], **localization** [Lansky, 1992] and briefly **symmetry** [Lowry, 1989].

In hierarchical problem solving, a search space can be reduced by generating abstractions with transformations known as concretizations. These transformations add constraints to the given problem. This speeds-up search because adding constraints reduces the branching factor. The generation of solutions becomes more efficient but

longer solutions will generally be obtained. This is very similar to our approach. To use concretizations, we need "primitive inverse" state spaces just like the requirement of the star algorithm. The idea in searching used in this strategy is to build a concrete space which is a subspace of the original, then solve from the initial state i to any state in the concretized space, say i', then solve backwards from the goal state g to some state in the concrete space, say g', then solve within the concrete space the problem to get from i' to g' and finally "glue" the three-part solution to get the final result: i...i'...g'...g. This process is done recursively in a hierarchy of concrete spaces. It is worth mentioning, that any solution in the concrete space is guaranteed to be a solution in the original space because the concretized space is more restricted [Prieditis and Janakiraman, 1992]. "However a solvable problem in the original space may have no solution in the concrete space" [Prieditis and Janakiraman, 1992]. This is not the case in our approach. In our approach, there is a solution in the abstract space if and only if there is a solution in the original space. There are also other problems using the concretization strategy. "Constructing concretization hierarchies is generally a difficult problem" [Prieditis and Janakiraman, 1992], and their generation is not automatic (as for our case), it is rather hand-crafted.

Localization is a technique developed in [Lansky, 1992] to improve planning. The main idea with this strategy is to "move away" from the classical STRIPS-like operator descriptions. The encoding of domain information is done in terms of "actions", "definitions", and "constraints". For example, in a STRIPS-like operator description, a precondition or a set of preconditions is included within the description of the operator. In the case of localization, it is explicitly "put apart" and defined as a constraint or a set of constraints. With this representation, the planning space gets broken into a set of smaller reasoning spaces where search is done, at first glance, independently. Those are called domain regions and localization allows them to overlap and interact. According to Lansky, abstraction itself could be seen as particular case of localization [Lansky, 1992].

The similarity with our approach is the idea of focusing search on local regions of the search space in order to make search more efficient.

The approach taken by Lowry is to abstract using symmetries. To abstract a set of states, he searches for a transformation group whose orbits stay within the boundaries of the set. Then a homomorphism is defined in which the orbits are mapped to individuals in the abstract model [Lowry, 1989]. This looks very promising once it is done. Usually if a space gets described through an algebraic structure or reformulated so that it satisfies particular algebraic properties, then we could expect impressive results. The problem here is that symmetries often do not exist and, when they do, they are hard to find. The similarity to our approach is mainly the interest in homomorphisms.

## 7.3 Conclusion

It is somewhat hard to compare our approach to most of the others described in this chapter, because we require an explicitly represented graph but others use an implicitly represented graph. If our approach gets improved so that it becomes applicable to some implicit representation of a graph, then heuristics could be compared in detail. Nevertheless, it is worth mentioning some general disadvantages that almost all the described systems share. Korf was unable to find a single good heuristic evaluation function for Rubik's cube. He concluded that "if there does exists a heuristic, its form is probably quite complex" [Korf, 1985]. Knoblock said: "The more stringent requirements on the abstractions formed by ALPINE, however, prevent it from finding abstractions in problem spaces in which SOAR can produce abstractions (e.g. the eight-puzzle)" [Knoblock, 1991]. Many of researchers conclude their papers by: "... the system has shown great results on certain AI problems and we are in the phase of "expanding" the system so that it efficiently handles more AI problems...". This all goes back to the sensitivity issue. On top of this, most of these systems use the STRIPS-like representation.

Although it looks appealing, there are many difficulties resulting from this, and each new system tries to overcome those difficulties. They usually succeed to do this for specific problems, but the solutions introduce new difficulties. For example, in planning, to prevent goals that have already been achieved from being undone there is strategy called goal protection that tries to handle this problem. But "goal protection can both benefit and hinder search performance, depending in which goals are solved and which of the previously goals are protected" [Knoblock et al, 1992].

It is very important to strive for a representation which avoids those problems. Definitely, it is not an easy task to do, but it is of a great importance and thus more effort in research should go towards this.

# Chapter 8

# Conclusion

This chapter is divided into three different sections. The first section presents a summary of the thesis. The second section shows the limitations and weaknesses of our approach to speed up search. Finally the third section describes some directions for future work.

## 8.1 Summary of the thesis

Most existing abstraction algorithms are sensitive to the initial problem formulation. Given two different descriptions of the same space, they will produce different abstractions, of which one might be efficient for problem-solving while the other might be inefficient.

This thesis presented a completely automated approach to generating and using abstractions for problem solving in state-spaces. The strategy to overcome the problem of sensitivity is called the graph relabelling strategy. The abstraction algorithms used are all based on that strategy and on a theoretical study of the complexity to abstract and to search using an abstraction. This study presents theorems and compares analytical results to some known graph algorithms.

The goal of this thesis is to devise a strategy for abstraction and prove its usefulness for automatically speeding up the search in state-spaces. The abstraction of a state-space is done recursively to create a hierarchy. This process is done only once and is "problem independent". This means that after creating an abstraction, we can use the abstraction to solve any given problem. The use of the abstraction hierarchy for problem-solving is also recursive. It was proved that, given a state-space with N states:

- The construction of the abstraction hierarchy is efficient: $O(N^2)$.
- Solving the problem instance using the hierarchy is faster on average than solving the problem in the original space.
- The solutions found using the hierarchy are generally longer than the solutions found in the original space, but they are not excessively large (this was not theoretically proved).
- The memory required to store the hierarchy is of the same order of storing the original state-space.

These results were empirically verified in the primitive inverse case by an experimental work based on the system we built. However, in the general case, our current algorithms are too costly.

## 8.2 Limitations and weaknesses

The main problem in our approach is the use of explicit graphs. Puzzles are normally represented implicitly as it is the case, for instance, with the STRIPS-like representations. To apply our methods, it is necessary to construct the whole graph representing the state-space. This limits our approach to relatively small graphs.

Another problem we face, is the generation of abstractions for the general case.

Although, the approach is safe and the worst abstraction we can obtain is the original space itself, we frequently do get the original space, that is our methods fail to abstract. Unlike the first weakness, this is not a fundamental limitation of our approach, but just a weakness of our current heuristics.

## 8.3 Future work

There are many possible directions in which to continue this work. These include work to be done in the near future and work to be thought of for the distant future.

### 8.3.1 Work for the near future

• Recall from Chapter 5, that refining using the *algorithm can be done without search. This will allow c, the class size, to be larger because search within the same class will only be a "table lookup" regardless of the class size. Recall that searching within the same class for the problem <S,G> has a solution S"hub"G. This approach could be further studied and empirically tested.

• Recall from Chapter 4, that there is a term called TW which is the total work to refine a solution from the most abstract level to the original level. When the classes size: c is constant, the term TW is a function of c and d. $TW = w/d\ N^{(\ln d)/(\ln c)}$. This function can be differentiated and then optimal values of c and d will be obtained for which TW is minimal.

### 8.3.2 Work for the distant future

• In this thesis, there was no attempt to minimize the solution length. This represents a very important point. If we suppose that we get optimal solutions under the same approach and with the same complexities then this will be considered as a challenging result. Probably, this needs more heuristics to control the way we abstract.

• Implicit representation of a "star": If we can, implicitly, represent or describe a node in the graph and its neighbours then this could permit us to work with large graphs and apply our algorithms on them.

• Abstracting explicitly "large" primitive inverse state-spaces: Given a problem <S,G>, we build T1: a tree with root S and depth dp, then T2: a tree with root G and depth dp. At this level the abstract space has three classes: T1, T2 and the class that contains all the other states, say C1. We abstract T1 and then T2 as separate spaces, and then we discard them from the whole graph (for future use). We do this recursively to C1 and we stop when "leaves" of T1 intersect with those of T2.

This is definitely not as simple as it looks, it could be considered as a starting point to tackle the problem of large graphs.

# Appendix A

# Puzzles

## A.1 Towers of Hanoi

It was already shown in previous chapters, and this puzzle has two versions: one for the primitive inverse case and one for the non-primitive case (see pages 11 and 75).

e.g. TOH3 is the 3-disk Towers of Hanoi.

## A.2 The Navigation Problem

Similarly as for the Towers of Hanoi, we've used two versions for both cases. The first one presents a non-primitive inverse state-space and was explained in Chapter 1 (page 3). The second presents a primitive inverse state-space: the robot can rotate in both directions and can move forward and backward.

e.g. NAV(3,3) The board is $3 \times 3$.

## A.3 The 5-Puzzle

It is a smaller version of the well-known 8-puzzle. There are five tiles in a two by three frame. The sixth position is referred to as the blank tile. A tile adjacent to the blank can be moved horizontally or vertically into that position. The goal is to move the tiles from one configuration to another. Usually a classical goal is one which corresponds to a configuration where the tiles are ordered. Here is the formulation of the problem we use in this experiment:

- A state is a list of six integers from 0 to 5.
- The number of all states is then 6! = 720.
- 0 represents the blank case, and the order is: start from the left upper corner and move clockwise until reaching the left lower corner.

e.g: The state [0,2,3,4,5,1] corresponds to:

```
- 2 3
1 5 4
```

The state [1,2,3,4,5,0] corresponds to:

```
1 2 3
- 5 4
```

- We have 4 possible operators:

  1: move blank to the right (if possible)

  2: - - - left -

  3: - - - up -

  4: - - - down -

# A.4 The Arrow Puzzle

There is a set of n arrows, each of which can point up or down. There are a set of n-1 operators, each of which apply to the two adjacent arrows. The effect of an operator is to invert both of the arrows it applies to. The problem is to find a sequence of operators that converts one configuration of arrows into another. Here is the formulation of the problem used in this experiment:

- A state is a list of size n that only contains zeros and ones. Zero corresponds to an arrow pointing down and one corresponds to an arrow pointing up.
- A state of size n is any combination of ones and zeros in a list of size n, that is the number of states of size n is 2 to the power of n.
- An operator is represented by a number between 1 and n-1. The operator i causes the arrows i and i+1 to be flipped.

e.g. Suppose that the number of arrows is 3.
The number of all states is 8. Those states are:
{[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]}
The operators are: 1 and 2.
Applying the operator 1 to the state [0,0,0] will produce [1,1,0].

e.g. ARR3: The number of arrows is 3.

# A.5 The Missionaries and Cannibals

There are n missionaries and n cannibals standing on the left bank of a river on which sits a boat that can ferry 2 of them across. The goal is to end up with all the people and the boat on the right bank. The constraint that must be satisfied at every state

is that no area (left or right banks) contains more cannibals than missionaries in fear that some missionaries may be eaten. The way this problem was formulated forbids any possibility of "losing" any missionaries. This means that the set of all states here are all the possible "safe" states. Here is our formulation:

- A state is a list [MR,CR,ML,CL] which means

| | | | | |
|---|---|---|---|---|
| MR: the number of missionaries on the right bank | | | | |
| CR: | - | cannibals | - | - |
| ML: | - | missionaries | - left | - |
| MR: | - | cannibals | - | - |

- In all there are 10 operators:

| | | | | | |
|---|---|---|---|---|---|
| 1: move 1 missionary from right bank to left bank | | | | | |
| 2: | - cannibal | - | - | - | - |
| 3: | - missionary | - | left | - | right |
| 4: | - cannibal | - | - | - | - |
| 5: | 1 missionary and 1 cannibal from right to left | | | | |
| 6: | 2 missionaries | | | - | - |
| 7: | 2 cannibals | | | - | - |
| 8: | 1 missionary and 1 cannibal from left to right | | | | |
| 9: | 2 missionaries | | | - | - |
| 10: | 2 cannibals | | | - | - |

e.g. MC3: The number of missionaries = the number of cannibals = 3.

## A.6 The Water Jug Problem

We are given two jugs, a big one and a smaller one. Neither has any measuring markers on it. The quantities are considered to be integers and the problem is how to get a certain amount of water in either one of the jugs. Here is our formulation of the problem:

- The variable big represents the capacity of the big jug, and small represents the capacity of the smaller jug.

- All possible states are all the couples (x,y) such that x is the content of the bigger jug and y is of the smaller jug. So x varies between 0 and big, and y varies between 0 and small.

- Here are all possible operators:

1: Fill the big jug
2: - - small -
3: Empty the big jug on the ground
4: - - small - -
5: Pour some water from the small jug into the big until the big jug is full
6: - - - - big - small - small -
7: Pour all the water from the small jug into the big jug
8: - - - - big - small -

e.g: Let big = 3 and small = 2.

All possible states are:

{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2),(3,0),(3,1),(3,2)}

Applying operator 3 to the state (2,1) results in the state (0,1).

e.g. JUG(4,2): big = 4 and small = 2.

# Appendix B

# Tabular results

| | |
|---|---|
| Dectime: | Abstraction time. |
| Arcs_traversed: | Number of arcs traversed. |
| Searchtime: | Searching time (total for all problems). |
| Avsolg: | Average solution length. |
| c: | Average class size. |
| algorithm: | The abstraction algorithm. |

## B.1 The Prim Inv Towers of Hanoi

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| TOH2 | | | | | | |
| 0.1000, | 390, | 0.5000, | 2.50, | 4.50, | 1.43, | Star & Abs |
| 0.1000, | 444, | 0.7000, | 2.31, | 2.12, | 1.32, | Rand Edge |
| 0.1000, | 354, | 0.4000, | 1.83, | 3.00, | 1.05, | Double Star |
| 0.2000, | 708, | 1.1000, | 2.83, | 1.66, | 1.62, | Edge |
| 0.1000, | 250, | 0.3000, | 1.75, | 1.00, | 1.00, | Original |
| 0.1000, | 312, | 0.6000, | 1.83, | 3.00, | 1.05, | Rand Star |
| 0.0000, | 262, | 0.5000, | 1.75, | 3.00, | 1.00, | Standard |
| 0.1000, | 326, | 0.5000, | 1.75, | 3.00, | 1.00, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| TOH3 | | | | | | |
| 0.8000, | 2093, | 5.5000, | 4.55, | 3.68, | 1.14, | Star & Abs |
| 1.0000, | 3519, | 8.6000, | 6.20, | 2.41, | 1.55, | Rand Edge |
| 1.0000, | 3255, | 7.0000, | 6.12, | 3.75, | 1.53, | Double Star |
| 1.1000, | 4583, | 1.4000, | 6.10, | 1.92, | 1.53, | Edge |
| 0.3000, | 3130, | 5.0000, | 4.00, | 1.00, | 1.00, | Original |
| 0.8000, | 1873, | 5.5000, | 4.25, | 3.00, | 1.06, | Rand Star |
| 0.4000, | 1489, | 5.5000, | 4.06, | 3.00, | 1.02, | Standard |
| 0.8000, | 2093, | 5.5000, | 4.55, | 3.68, | 1.14, | Star |
| TOH4 | | | | | | |
| 8.4000, | 17517, | 12.6000, | 14.87, | 3.52, | 1.83, | Star & Abs |
| 1.9000, | 28301, | 17.9000, | 14.60, | 2.10, | 1.80, | Rand Edge |
| 4.5000, | 16783, | 16.9000, | 12.19, | 3.45, | 1.50, | Double Star |
| 3.2000, | 31580, | 17.1000, | 14.12, | 1.86, | 1.74, | Edge |
| 2.2000, | 33355, | 13.9000, | 8.13, | 1.00, | 1.00, | Original |
| 7.4000, | 14443, | 15.3000, | 10.44, | 3.05, | 1.28, | Rand Star |
| 3.4000, | 7885, | 8.4000, | 8.32, | 3.00, | 1.02, | Standard |
| 8.5000, | 17090, | 14.1000, | 12.30, | 3.48, | 1.51, | Star |
| TOH5 | | | | | | |
| 10.3000, | 95604, | 143.5000, | 23.25, | 4.34, | 1.40, | Star & Abs |
| 18.5000, | 171677, | 272.0000, | 30.29, | 2.23, | 1.83, | Rand Edge |
| 31.9000, | 122527, | 287.3000, | 28.03, | 3.18, | 1.69, | Double Star |
| 22.8000, | 228820, | 640.3000, | 34.90, | 1.72, | 2.11, | Edge |
| 7.0000, | 325238, | 348.6000, | 16.57, | 1.00, | 1.00, | Original |
| 15.9000, | 85372, | 166.3000, | 21.27, | 3.00, | 1.28, | Rand Star |
| 9.5000, | 43485, | 149.3000, | 17.05, | 3.00, | 1.03, | Standard |
| 10.7000, | 114489, | 217.8000, | 28.44, | 3.04, | 1.72, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| TOH6 | | | | | | |
| 91.5000, | 624494, | 3635.6000, | 52.02, | 3.95, | 1.55, | Star & Abs |
| 140.0000, | 1048066, | 4801.7000, | 64.83, | 2.20, | 1.93, | Rand Edge |
| 533.2000, | 721575, | 3824.0000, | 52.39, | 4.45, | 1.56, | Double Star |
| 166.0000, | 1547514, | 14171.4000, | 79.18, | 1.72, | 2.36, | Edge |
| 27.7000, | 3047735, | 9643.3000, | 33.52, | 1.00, | 1.00, | Original |
| 104.6000, | 635669, | 8073.2000, | 52.61, | 2.73, | 1.57, | Rand Star |
| 43.2000, | 235379, | 2966.8000, | 34.63, | 3.00, | 1.03, | Standard |
| 108.5000, | 808648, | 5338.5000, | 66.13, | 3.35, | 1.97, | Star |

## B.2 The Arrow puzzle

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| ARR3 | | | | | | |
| 0.1000, | 40, | 0.1000, | 1.00, | 1.00, | 1.00, | Star & Abs |
| 0.1000, | 88, | 0.2000, | 1.00, | 2.00, | 1.00, | Rand Edge |
| 0.1000, | 40, | 0.1000, | 1.00, | 1.00, | 1.00, | Double Star |
| 0.1000, | 88, | 0.1000, | 1.00, | 2.00, | 1.00, | Edge |
| 0.0000, | 40, | 0.1000, | 1.00, | 1.00, | 1.00, | Original |
| 0.1000, | 95, | 0.2000, | 1.00, | 2.00, | 1.00, | Rand Star |
| 0.1000, | 95, | 0.1000, | 1.00, | 2.00, | 1.00, | Star |
| ARR4 | | | | | | |
| 0.3000, | 356, | 0.7000, | 1.81, | 4.00, | 1.21, | Star & Abs |
| 0.5000, | 480, | 1.0000, | 1.50, | 2.00, | 1.00, | Rand Edge |
| 0.4000, | 313, | 0.6000, | 1.50, | 4.00, | 1.00, | Double Star |
| 0.4000, | 480, | 1.0000, | 1.50, | 2.00, | 1.00, | Edge |
| 0.1000, | 180, | 0.4000, | 1.50, | 1.00, | 1.00, | Original |
| 0.5000, | 360, | 0.7000, | 1.69, | 2.33, | 1.13, | Rand Star |
| 0.4000, | 356, | 0.7000, | 1.81, | 4.00, | 1.21, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| ARR5 | | | | | | |
| 1.6000, | 1172, | 2.9000, | 2.31, | 5.33, | 1.16, | Star & Abs |
| 2.2000, | 2336, | 5.4000, | 2.12, | 2.00, | 1.06, | Rand Edge |
| 2.0000, | 1188, | 3.1000, | 2.31, | 8.00, | 1.16, | Double Star |
| 2.3000, | 2336, | 5.4000, | 2.12, | 2.00, | 1.06, | Edge |
| 0.3000, | 936, | 1.9000, | 2.00, | 1.00, | 1.00, | Original |
| 1.9000, | 1037, | 3.6000, | 2.56, | 3.40, | 1.28, | Rand Star |
| 2.0000, | 2124, | 4.5000, | 3.38, | 3.33, | 1.69 | Star |
| ARR6 | | | | | | |
| 8.2000, | 3506, | 4.7000, | 3.38, | 4.00, | 1.35, | Star & Abs |
| 2.6000, | 10568, | 10.1000, | 2.81, | 2.00, | 1.12, | Rand Edge |
| 1.5000, | 3139, | 6.1000, | 3.56, | 5.33, | 1.42, | Double Star |
| 2.4000, | 10568, | 11.0000, | 2.81, | 2.00, | 1.12, | Edge |
| 1.0000, | 4940, | 4.0000, | 2.50, | 1.00, | 1.00, | Original |
| 9.8000, | 8518, | 9.6000, | 5.34, | 2.85, | 2.14, | Rand Star |
| 1.5000, | 6844, | 10.9000, | 3.53, | 4.00, | 1.41, | Star |
| ARR7 | | | | | | |
| 8.1000, | 19104, | 11.7000, | 5.02, | 5.79, | 1.67, | Star & Abs |
| 12.0000, | 45696, | 19.8000, | 3.53, | 2.00, | 1.18, | Rand Edge |
| 13.4000, | 10252, | 9.6000, | 3.94, | 8.00, | 3.94, | Double Star |
| 12.1000, | 45696, | 15.9000, | 3.53, | 2.00, | 1.18, | Edge |
| 4.7000, | 25360, | 11.9000, | 3.00, | 1.00, | 1.00, | Original |
| 8.1000, | 34262, | 28.0000, | 6.78, | 2.67, | 2.26, | Rand Star |
| 12.3000, | 18217, | 11.0000, | 5.95, | 6.13, | 1.98, | Star |
| ARR8 | | | | | | |
| 48.2000, | 70813, | 59.7000, | 7.80, | 8.25, | 2.23, | Star & Abs |
| 39.6000, | 191992, | 88.9000, | 4.27, | 2.00, | 1.22, | Rand Edge |
| 50.8000, | 83782, | 60.5000, | 6.12, | 7.16, | 1.75, | Double Star |
| 49.5000, | 191992, | 86.7000, | 4.27, | 2.00, | 1.22, | Edge |
| 5.7000, | 125860, | 91.8000, | 3.50, | 1.00, | 1.00, | Original |
| 58.2000, | 150516, | 123.2000, | 7.48, | 2.18, | 2.14, | Rand Star |
| 52.2000, | 63426, | 77.6000, | 6.63, | 4.54, | 1.89, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| ARR9 | | | | | | |
| 352.5000, | 366436, | 334.1000, | 9.70, | 9.43, | 2.43, | Star & Abs |
| 271.9000, | 791920, | 490.7000, | 5.01, | 2.00, | 1.25, | Rand Edge |
| 353.7000, | 440030, | 533.4000, | 10.63, | 8.55, | 2.66, | Double Star |
| 267.9000, | 791920, | 487.7000, | 5.01, | 2.00, | 1.25, | Edge |
| 15.6000, | 606152, | 730.5000, | 4.00, | 1.00, | 1.00, | Original |
| 285.8000, | 308766, | 360.5000, | 8.16, | 5.16, | 2.04, | Rand Star |
| 349.3000, | 359157, | 475.9000, | 10.37, | 4.69, | 1.17, | Star |

## B.3 The 5-Puzzle

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| 550.9000, | 158462, | 2275.6000, | 29.35, | 4.72, | 2.36, | Star & Abs, |
| 593.5000, | 212746, | 3061.7000, | 22.04, | 2.41, | 1.77, | Rand Edge, |
| 1766.1000, | 146331, | 2638.5000, | 23.42, | 3.50, | 1.88, | Double Star, |
| 683.6000, | 385214, | 4373.8000, | 20.39, | 1.79, | 1.64, | Edge, |
| 30.0000, | 520204, | 4151.7000, | 12.44, | 1.00, | 1.00, | Original, |
| 595.2000, | 192302, | 3117.1000, | 22.49, | 2.31, | 1.81, | Rand Star, |
| 498.6000, | 131401, | 2357.2000, | 23.90, | 3.56, | 1.92, | Star, |

## B.4 The Prim Inv Navigation Problem

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| NAV(3,3) | | | | | | |
| 5.1000, | 3750, | 7.1000, | 4.72, | 4.00, | 1.53, | Star & Abs |
| 7.9000, | 7655, | 13.7000, | 5.25, | 2.06, | 1.70, | Rand Edge |
| 6.8000, | 2911, | 5.5000, | 4.07, | 4.00, | 1.32, | Double Star |
| 8.5000, | 8527, | 17.3000, | 4.75, | 1.79, | 1.54, | Edge |
| 1.1000, | 5402, | 4.0000, | 3.08, | 1.00, | 1.00, | Original |
| 7.1000, | 4668, | 11.5000, | 4.38, | 2.73, | 1.42, | Rand Star |
| 6.7000, | 5518, | 6.0000, | 5.57, | 3.50, | 1.81, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| NAV(4,4) | | | | | | |
| 7.0000, | 11188, | 23.8000, | 5.79, | 4.67, | 1.49, | Star & Abs |
| 7.6000, | 20667, | 32.2000, | 5.81, | 2.00, | 1.50, | Rand Edge |
| 10.7000, | 11142, | 25.7000, | 5.31, | 5.79, | 1.37, | Double Star |
| 7.6000, | 20667, | 31.2000, | 5.81, | 2.00, | 1.50, | Edge |
| 2.9000, | 19509, | 22.5000, | 3.88, | 1.00, | 1.00, | Original |
| 7.2000, | 26806, | 39.4000, | 6.35, | 2.50, | 1.42, | Rand Star |
| 7.7000, | 11453, | 16.3000, | 5.80, | 4.03, | 1.50, | Star |
| NAV(5,5) | | | | | | |
| 12.0000, | 25837, | 47.2000, | 8.29, | 5.08, | 1.69, | Star & Abs |
| 12.0000, | 42127, | 76.4000, | 7.64, | 2.02, | 1.56, | Rand Edge |
| 12.6000, | 28961, | 57.3000, | 10.53, | 5.08, | 2.15, | Double Star |
| 15.1000, | 53259, | 100.6000, | 7.58, | 1.92, | 1.55, | Edge |
| 6.4000, | 53724, | 78.0000, | 4.90, | 1.00, | 1.00, | Original |
| 10.8000, | 40467, | 79.7000, | 7.89, | 2.68, | 1.61, | Rand Star |
| 14.3000, | 19860, | 42.2000, | 7.78, | 4.12, | 1.59, | Star |
| NAV(6,6) | | | | | | |
| 13.1000, | 42810, | 103.1000, | 10.03, | 5.45, | 1.77, | Star & Abs |
| 16.5000, | 100283, | 244.4000, | 10.79, | 2.04, | 1.90, | Rand Edge |
| 30.3000, | 57349, | 114.7000, | 9.20, | 4.24, | 1.62, | Double Star |
| 23.5000, | 92951, | 257.8000, | 8.33, | 1.85, | 1.47, | Edge |
| 3.0000, | 118204, | 237.0000, | 5.67, | 1.00, | 1.00, | Original |
| 28.8000, | 92370, | 268.6000, | 11.85, | 2.93, | 2.09, | Rand Star |
| 22.6000, | 46642, | 139.1000, | 9.22, | 3.64, | 1.63, | Star |
| NAV(7,7) | | | | | | |
| 25.8000, | 72635, | 193.8000, | 12.34, | 5.37, | 1.88, | Star & Abs |
| 29.1000, | 142951, | 406.0000, | 10.96, | 2.01, | 1.67, | Rand Edge |
| 54.6000, | 77423, | 469.6000, | 12.76, | 4.76, | 1.94, | Double Star |
| 41.6000, | 166119, | 537.4000, | 10.33, | 1.93, | 1.57, | Edge |
| 3.7000, | 230120, | 590.9000, | 6.57, | 1.00, | 1.00, | Original |
| 44.0000, | 132034, | 645.8000, | 14.79, | 2.53, | 2.25, | Rand Star |
| 37.9000, | 83985, | 243.1000, | 10.60, | 4.68, | 1.61, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| NAV(8,8) | | | | | | |
| 52.5000, | 95617, | 311.1000, | 11.63, | 6.32, | 1.58, | Star & Abs |
| 54.4000, | 227143, | 920.0000, | 10.47, | 2.00, | 1.42, | Rand Edge |
| 97.4000, | 119999, | 407.1000, | 12.53, | 5.88, | 1.70, | Double Star |
| 53.8000, | 227143, | 924.0000, | 10.47, | 2.00, | 1.42, | Edge |
| 8.9000, | 405154, | 1324.3000, | 7.38, | 1.00, | 1.00, | Original |
| 96.3000, | 304267, | 1241.7000, | 20.97, | 2.70, | 2.84, | Rand Star |
| 82.3000, | 164498, | 701.3000, | 16.67, | 4.02, | 2.26, | Star |
| NAV(9,9) | | | | | | |
| 126.8000, | 232566, | 1518.3000, | 14.15, | 5.54, | 1.68, | Star & Abs |
| 104.7000, | 373737, | 1840.5000, | 14.50, | 2.08, | 1.72, | Rand Edge |
| 204.1000, | 203591, | 1351.5000, | 14.60, | 4.39, | 1.73, | Double Star |
| 115.6000, | 438224, | 2382.4000, | 14.31, | 1.90, | 1.70, | Edge |
| 11.4000, | 673620, | 2753.2000, | 8.42, | 1.00, | 1.00, | Original |
| 153.8000, | 332816, | 1542.0000, | 16.23, | 2.84, | 1.93, | Rand Star |
| 129.7000, | 192579, | 1192.4000, | 17.52, | 4.87, | 2.08, | Star |
| NAV(10,10) | | | | | | |
| 193.9000, | 292484, | 2238.5000, | 24.61, | 5.97, | 2.70, | Star & Abs |
| 145.4000, | 464270, | 2376.1000, | 14.07, | 2.01, | 1.55, | Rand Edge |
| 327.2000, | 264313, | 1714.7000, | 15.45, | 4.33, | 1.70, | Double Star |
| 174.6000, | 565943, | 3603.6000, | 15.13, | 1.94, | 1.66, | Edge |
| 13.7000, | 1039722, | 5195.9000, | 9.10, | 1.00, | 1.00, | Original |
| 336.6000, | 625816, | 18475.7000, | 21.39, | 2.03, | 2.35, | Rand Star |
| 241.6000, | 295014, | 1756.0000, | 17.20, | 4.69, | 1.89, | Star |

# B.5 The Missionaries and Cannibals

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| MC5 | | | | | | |
| 1.3000, | 784, | 5.9000, | 3.08, | 8.00, | 1.00, | Star & Abs |
| 1.8000, | 1420, | 2.3000, | 4.09, | 2.00, | 1.33, | Rand Edge |
| 1.7000, | 1158, | 1.3000, | 2.55, | 3.00, | 0.83, | Double Star |
| 1.8000, | 1420, | 2.2000, | 4.09, | 2.00, | 1.33, | Edge |
| 0.8000, | 1183, | 5.9000, | 3.08, | 1.00, | 1.00, | Original |
| 1.6000, | 1253, | 2.2000, | 3.42, | 2.12, | 1.11, | Rand Star |
| 1.6000, | 1309, | 1.2000, | 2.59, | 2.33, | 0.84, | Star |
| MC10 | | | | | | |
| 4.3000, | 3112, | 7.5000, | 6.55, | 4.08, | 1.04, | Star & Abs |
| 6.1000, | 4168, | 8.8000, | 7.46, | 2.18, | 1.18, | Rand Edge |
| 6.1000, | 3214, | 7.6000, | 6.45, | 3.22, | 1.02, | Double Star |
| 6.9000, | 5590, | 8.8000, | 7.43, | 1.74, | 1.18, | Edge |
| 2.0000, | 5235, | 7.9000, | 6.31, | 1.00, | 1.00, | Original |
| 5.1000, | 3743, | 9.0000, | 6.79, | 2.53, | 1.08, | Rand Star |
| 4.6000, | 2961, | 7.9000, | 5.68, | 3.22, | 0,90, | Star |
| MC20 | | | | | | |
| 8.0000, | 10868, | 22.7000, | 13.70, | 3.16, | 1.07, | Star & Abs |
| 3.9000, | 11816, | 32.5000, | 15.00, | 2.38, | 1.18, | Rand Edge |
| 8.7000, | 12616, | 21.9000, | 13.95, | 3.19, | 1.09, | Double Star |
| 5.5000, | 13476, | 35.5000, | 15.64, | 1.98, | 1.22, | Edge |
| 6.3000, | 21825, | 27.9000, | 12.77, | 1.00, | 1.00, | Original |
| 8.5000, | 12034, | 27.5000, | 13.46, | 2.77, | 1.05, | Rand Star |
| 2.0000, | 11927, | 32.7000, | 12.82, | 2.40, | 1.00, | Star |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| MC50 | | | | | | |
| 13.4000, | 58766, | 194.4000, | 33.70, | 3.73, | 1.05, | Star & Abs |
| 19.5000, | 60494, | 277.2000, | 38.96, | 2.20, | 1.21, | Rand Edge |
| 32.8000, | 71089, | 263.8000, | 33.69, | 2.99, | 1.05, | Double Star |
| 24.0000, | 74750, | 423.8000, | 39.84, | 1.73, | 1.24, | Edge |
| 6.0000, | 139488, | 332.2000, | 32.16, | 1.00, | 1.00, | Original |
| 19.6000, | 65426, | 298.8000, | 33.31, | 2.42, | 1.04, | Rand Star |
| 15.5000, | 65146, | 235.0000, | 33.21, | 2.69, | 1.03, | Star |
| MC100 | | | | | | |
| 50.7000, | 240404, | 1427.2000, | 68.21, | 3.19, | 1.06, | Star & Abs |
| 56.0000, | 205427, | 2385.8000, | 78.83, | 2.05, | 1.22, | Rand Edge |
| 169.0000, | 266741, | 1419.7000, | 67.64, | 3.21, | 1.05, | Double Star |
| 65.6000, | 217055, | 3070.6000, | 80.66, | 1.76, | 1.25, | Edge |
| 18.2000, | 561913, | 2533.0000, | 64.50, | 1.00, | 1.00, | Original |
| 55.7000, | 252725, | 1869.9000, | 69.33, | 2.33, | 1.07, | Rand Star |
| 52.6000, | 244960, | 1776.8000, | 67.98, | 2.55, | 1.05, | Star |

## B.6 The Non-Prim Inv Navigation Problem

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| NAV(3,3) | | | | | | |
| 3.8000, | 3102, | 3.9000, | 6.31, | 6.10, | 1.32, | Dble I/O Strong* |
| 0.1000, | 3227, | 8.8000, | 4.78, | 1.00, | 1.00, | Original |
| 1.3000, | 3227, | 9.4000, | 4.78, | 36.00, | 1.00, | Strong * |
| 2.2000, | 3227, | 9.6000, | 4.78, | 36.00, | 1.00, | I/O Strong * |
| NAV(4,4) | | | | | | |
| 4.2000, | 9173, | 13.3000, | 8.45, | 4.09, | 1.45, | Dble I/O Strong* |
| 0.3000, | 10925, | 8.2000, | 5.83, | 1.00, | 1.00, | Original |
| 4.4000, | 10925, | 9.4000, | 5.83, | 64.00, | 1.00, | Strong * |
| 8.2000, | 10925, | 9.5000, | 5.83, | 64.00, | 1.00, | I/O Strong * |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| NAV(5,5) | | | | | | |
| 12.6000, | 24389, | 35.8000, | 12.29, | 5.49, | 1.76, | Dble I/O Strong* |
| 0.8000, | 28595, | 22.6000, | 7.00, | 1.00, | 1.00, | Original |
| 1.8000, | 28595, | 18.2000, | 7.00, | 100.0, | 1.00, | Strong * |
| 7.2000, | 28595, | 21.0000, | 7.00, | 100.0, | 1.00, | I/O Strong * |
| NAV(6,6) | | | | | | |
| 13.7000, | 38292, | 90.7000, | 13.03, | 5.26, | 1.70, | Dble I/O Strong* |
| 1.7000, | 61869, | 52.3000, | 7.66, | 1.00, | 1.00, | Original |
| 6.2000, | 61869, | 48.7000, | 7.66, | 144.0, | 1.00, | Strong * |
| 11.0000, | 61869, | 51.1000, | 7.66, | 144.0, | 1.00, | I/O Strong * |
| NAV(7,7) | | | | | | |
| 22.4000, | 77327, | 156.3000, | 14.36, | 10.20, | 1.64, | Dble I/O Strong* |
| 3.0000, | 118879, | 124.0000, | 8.73, | 1.00, | 1.00, | Original |
| 13.5000, | 118879, | 126.1000, | 8.73, | 196.0, | 1.00, | Strong * |
| 23.3000, | 118879, | 124.7000, | 8.73, | 196.0, | 1.00, | I/O Strong * |
| NAV(8,8) | | | | | | |
| 41.4000, | 144064, | 346.7000, | 22.61, | 9.05, | 2.36, | Dble I/O Strong* |
| 5.3000, | 208507, | 268.4000, | 9.57, | 1.00, | 1.00, | Original |
| 15.7000, | 208507, | 265.7000, | 9.57, | 256.0, | 1.00, | Strong * |
| 37.7000, | 208507, | 274.9000, | 9.57, | 256.0, | 1.00, | I/O Strong * |
| NAV(9,9) | | | | | | |
| 84.8000, | 140454, | 449.6000, | 19.63, | 9.66, | 1.82 | Dble I/O Strong* |
| 8.2000, | 342989, | 552.2000, | 10.80, | 1.00, | 1.00, | Original |
| 26.5000, | 342989, | 551.3000, | 10.80, | 324.0, | 1.00, | Strong * |
| 72.0000, | 342989, | 545.4000, | 10.80, | 324.0, | 1.00, | I/O Strong * |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| NAV(10,10) | | | | | | |
| 143.1000, | 259897, | 856.8000, | 22.40, | 15.44, | 2.00, | Dble I/O Strong* |
| 2.8000, | 530177, | 1014.6000, | 11.21, | 1.00, | 1.00, | Original |
| 39.5000, | 530177, | 1020.4000, | 11.21, | 400.0, | 1.00, | Strong * |
| 126.0000, | 530177, | 1019.5000, | 11.21, | 400.0, | 1.00, | I/O Strong * |

## B.7 The Non-Prim Inv Towers of Hanoi

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| TOH2 | | | | | | |
| 0.4000, | 188, | 0.4000, | 2.83, | 9.00, | 1.00, | Dble I/O Strong* |
| 0.0000, | 188, | 0.3000, | 2.83, | 1.00, | 1.00, | Original |
| 0.1000, | 237, | 0.5000, | 2.83, | 3.00, | 1.00, | Standard |
| 0.1000, | 188, | 0.4000, | 2.83, | 9.00, | 1.00, | Strong * |
| 0.2000, | 237, | 0.5000, | 2.83, | 3.00, | 1.00, | I/O Strong * |
| 0.3000, | 188, | 0.4000, | 2.83, | 9.00, | 1.00, | Dble Strong * |
| TOH3 | | | | | | |
| 7.4000, | 1753, | 6.3000, | 9.22, | 5.43, | 1.00, | Dble I/O Strong* |
| 0.2000, | 1921, | 4.3000, | 9.22, | 1.00, | 1.00, | Original |
| 0.2000, | 1660, | 6.6000, | 9.22, | 3.00, | 1.00, | Standard |
| 0.8000, | 1921, | 4.8000, | 9.22, | 27.00, | 1.00, | Strong * |
| 2.4000, | 2189, | 8.5000, | 9.22, | 3.23, | 1.00, | I/O Strong * |
| 1.8000, | 1539, | 5.9000, | 9.22, | 6.00, | 1.00, | Dble Strong * |
| TOH4 | | | | | | |
| 28.4000, | 15464, | 13.2000, | 27.11, | 4.39, | 1.00, | Dble I/O Strong* |
| 1.0000, | 18629, | 15.5000, | 27.11, | 1.00, | 1.00, | Original |
| 2.9000, | 11634, | 19.2000, | 27.11, | 3.00, | 1.00, | Standard |
| 7.1000, | 18629, | 15.0000, | 27.11, | 81.00, | 1.00, | Strong * |
| 10.1000, | 18008, | 20.4000, | 27.11, | 6.56, | 1.00, | I/O Strong * |
| 9.7000, | 15687, | 12.6000, | 27.11, | 15.90, | 1.00, | Dble Strong * |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| TOH5 | | | | | | |
| 560.0000, | 117939, | 301.4000, | 76.56, | 4.00, | 1.00, | Dble I/O Strong* |
| 3.0000, | 173150, | 258.5000, | 76.56, | 1.00, | 1.00, | Original |
| 4.8000, | 86095, | 258.8000, | 76.56, | 3.00, | 1.00, | Standard |
| 13.2000, | 173150, | 266.3000, | 76.56, | 243.0, | 1.00, | Strong * |
| 54.9000, | 144720, | 434.8000, | 76.56, | 2.53, | 1.00, | I/O Strong * |
| 36.0000, | 141385, | 191.1000, | 76.56, | 46.84, | 1.00, | Dble Strong * |
| TOH6 | | | | | | |
| 14972.0000, | 1087538, | 7767.7000, | 210.95, | 4.02, | 1.00, | Dble I/O Strong* |
| 18.4000, | 1579631, | 7028.0000, | 210.95, | 1.00, | 1.00, | Original |
| 35.2000, | 671167, | 6402.7000, | 210.95, | 3.00, | 1.00, | Standard |
| 242.8000, | 1579631, | 7042.8000, | 210.95, | 729.0, | 1.00, | Strong * |
| 1164.9000, | 989665, | 10983.7000, | 210.95, | 3.12, | 1.00, | I/O Strong * |
| 559.5000, | 1251730, | 5271.7000, | 210.95, | 48.52, | 1.00, | Dble Strong * |

## B.8 The Water Jug Problem

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| JUG(4,2) | | | | | | |
| 0.7000, | 124, | 0.3000, | 1.21, | 1.85, | 1.00, | Dble I/O Strong* |
| 0.1000, | 124, | 0.2000, | 1.21, | 1.00, | 1.00, | Original |
| 0.6000, | 213, | 0.5000, | 1.33, | 1.77, | 1.10, | Strong * |
| 0.6000, | 241, | 0.4000, | 1.21, | 1.77, | 1.00, | I/O Strong * |
| 0.7000, | 124, | 0.3000, | 1.21, | 1.85, | 1.00, | Dble Strong * |
| JUG(9,4) | | | | | | |
| 4.9000, | 5374, | 9.5000, | 4.83, | 2.00, | 1.00, | Dble I/O Strong* |
| 0.8000, | 5374, | 8.3000, | 4.83, | 1.00, | 1.00, | Original |
| 7.3000, | 5374, | 1.2000, | 4.83, | 2.00, | 1.00, | Strong * |
| 6.8000, | 5374, | 1.0000, | 4.83, | 2.00, | 1.00, | I/O Strong * |
| 5.9000, | 5374, | 9.5000, | 4.83, | 2.00, | 1.00, | Dble Strong * |

| dectime | arcs_traversed | searchtime | avsolg | c | ratio | algorithm |
|---|---|---|---|---|---|---|
| JUG(15,7) | | | | | | |
| 4.8000, | 17227, | 11.8000, | 9.31, | 1.51, | 1.00, | Dble I/O Strong* |
| 4.7000, | 17227, | 10.2000, | 9.31, | 1.00, | 1.00, | Original |
| 5.1000, | 17227, | 12.2000, | 9.31, | 1.51, | 1.00, | Strong * |
| 5.2000, | 17227, | 13.2000, | 9.31, | 1.51, | 1.00, | I/O Strong * |
| 12.2000, | 17227, | 12.5000, | 9.31, | 1.51, | 1.00, | Dble Strong * |
| JUG(20,9) | | | | | | |
| 12.4000, | 29715, | 19.1000, | 9.07, | 1.37, | 1.00, | Dble I/O Strong* |
| 4.1000, | 29715, | 19.0000, | 9.07, | 1.00, | 1.00, | Original |
| 14.6000, | 29715, | 19.5000, | 9.07, | 1.37, | 1.00, | Strong * |
| 21.1000, | 29715, | 11.6000, | 9.07, | 1.37, | 1.00, | I/O Strong * |
| 22.2000, | 29715, | 12.4000, | 9.07, | 1.37, | 1.00, | Dble Strong * |
| JUG(50,3) | | | | | | |
| 14.4000, | 105551, | 41.9000, | 17.24, | 2.06, | 1.00, | Dble I/O Strong* |
| 3.6000, | 105551, | 41.9000, | 17.24, | 1.00, | 1.00, | Original |
| 24.7000, | 87957, | 47.8000, | 24.46, | 1.52, | 1.00, | Strong * |
| 17.7000, | 105551, | 45.7000, | 17.24, | 2.06, | 1.00, | I/O Strong * |
| 26.2000, | 93677, | 63.0000, | 25.24, | 1.53, | 1.00, | Dble Strong * |

# Appendix C

# Raw Program Output

| | |
|---|---|
| ClSizes: | Sizes of all classes |
| ClperLev: | Number of classes per level |
| Arcs_traversed: | Number of arcs traversed |
| avSolLength: | Average solution length |
| c: | Average c. |
| dectime: | Abstraction time. |
| levels: | Number of abstraction levels. |
| ratioTsum: | Arcs_traversed/Sumlength. |
| Searchtime: | Searching time (total for all problems). |
| Sumlength: | The sum of lengths of all solutions. |

## C.1 The Prim Inv Towers of Hanoi

TOH6

```
val it =
  {STAR_ABS={ClSizesA1=[[346,289,94],[95,128,69,119,53,50,45,44,43,27,30,26],
                [22,18,19,18,20,24,18,20,22,19,19,24,20,19,18,25,19,14,14,
                 15,17,19,18,15,14,15,14,17,21,26,16,12,13,12,15,11,13,12,
                 8,8,9,8,8,9,6,6],
                [5,4,4,4,4,4,4,4,4,5,4,5,4,4,4,4,4,4,4,4,4,4,4,5,4,5,5,4,
                 4,4,4,4,4,4,4,4,4,4,4,5,4,4,4,5,4,4,4,4,4,4,4,4,4,4,4,4,4,
                 4,4,5,4,5,4,4,4,5,5,4,5,4,5,4,5,4,4,4,4,4,5,4,4,4,5,4,4,4,
                 4,4,4,4,4,4,4,4,5,4,5,4,5,...]],
      ClperLevA1=[3,12,46,188,729],Arcs_traversedA1=624494,
      avSolLengthA1=52.0205761316872,cA1=3.94948735738514,
```

```
      dectimeA1=TIME {sec=90,usec=150000},levelsA1=5,
      ratioTsumA1=4.11685520660285,searchtimeA1=TIME {sec=3633,usec=260000},
      sumlengthA1=151692},
   RAND_EDGE={ClSizesA2=[[299,228,202],[86,110,102,101,100,118,112],
              [32,56,56,56,56,54,54,62,46,45,44,60,52,56],
              [16,20,20,20,20,28,28,20,28,27,26,18,18,18,16,16,20,16,16,
               16,20,20,20,16,16,16,16,26,26,26,18,18,18,26,18,22],
              [8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
               8,8,8,8,8,8,8,8,8,8,8,8,8,12,12,12,12,12,12,12,12,12,12,
               12,11,10,10,10,10,10,10,10,10,10,10,10,10,14,8,8,8,8,8,8,
               8,8,8,8,8,8,8,8],
              [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
               4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
               4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
               4,4,4,4,4,4,4,6,6,6,6,6,6,...],
              [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
               2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
               2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
               2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
      ClperLevA2=[3,7,14,36,81,175,364,729],Arcs_traversedA2=1048066,
      avSolLengthA2=64.8292181069959,cA2=2.19971471209566,
      dectimeA2=TIME {sec=131,usec=900000},levelsA2=8,
      ratioTsumA2=5.54409073116027,searchtimeA2=TIME {sec=4796,usec=570000},
      sumlengthA2=189042},
   DOUBLE_STAR={ClSizesMA2=[[584,145],[145,83,111,83,87,63,81,18,26,18,9,5],
                 [30,22,21,26,31,22,22,30,27,27,26,20,24,19,27,23,24,23,
                  25,13,35,18,22,17,18,21,15,10,9,10,12,6,9,5,5,5,6,5,6,
                  4,3,3,2,1],
                 [6,6,6,6,5,5,6,5,6,5,6,6,5,5,6,5,5,6,5,5,5,5,5,5,5,5,6,
                  5,6,5,5,6,5,5,5,5,5,6,5,5,6,5,5,5,5,6,5,6,5,5,5,5,6,5,
                  6,5,6,5,5,5,5,6,5,6,5,5,5,5,5,6,5,6,6,5,6,5,5,5,5,5,5,
                  5,5,5,5,6,5,5,6,6,5,5,5,6,5,6,4,5,4,4,...]],
        ClperLevMA2=[2,12,44,175,729],Arcs_traversedMA2=721575,
        avSolLengthMA2=52.3923182441701,cMA2=4.45241341991342,
        dectimeMA2=TIME {sec=532,usec=120000},levelsMA2=5,
        ratioTsumMA2=4.72309132324449,
        searchtimeMA2=TIME {sec=3820,usec=400000},sumlengthMA2=152776},
   EDGE={ClSizesCO=[[696,33],[260,436,33],[236,308,128,33,24],
               [198,146,162,116,38,33,24,12],
               [98,108,100,100,108,62,38,38,33,24,12,8],
               [60,56,48,48,56,38,38,38,38,33,52,52,52,52,24,24,12,8],
               [32,32,24,24,24,22,22,22,22,22,20,20,20,20,24,32,24,24,
                32,32,32,32,28,24,24,16,16,16,16,11,12,8],
               [16,16,16,16,16,16,16,16,16,16,14,14,14,14,16,16,16,16,
                16,16,16,16,16,16,16,16,16,11,16,16,16,16,12,16,16,16,
                16,12,16,8,8,8,8,8,8,6,6,6,6,6,6,6,6,8,8,8],
               [8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,6,6,
                6,6,6,6,6,6,6,6,6,6,6,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                8,8,8,8,8,8,8,8,8,3,4,4],
               [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...],
               [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
     ClperLevCO=[2,3,5,8,12,18,32,57,96,189,365,729],
```

```
        Arcs_traversedCO=1547514,avSolLengthCO=79.1803840877915,
        cCO=1.71883019781362,dectimeCO=TIME {sec=165,usec=100000},
        levelsCO=12,ratioTsumCO=6.7023864177747,
        searchtimeCO=TIME {sec=14164,usec=740000},sumlengthCO=230890},
   ORGINAL={ClSizesO=[[1]],ClperLevO=[729],Arcs_traversedO=3047735,
        avSolLengthO=33.5157750342936,cO=1.0,
        dectimeO=TIME {sec=27,usec=70000},levelsO=1,
        ratioTsumO=31.1846171161953,
        searchtimeO=TIME {sec=9634,usec=930000},sumlengthO=97732},
   RAND_STAR={ClSizesRS=[[660,69],[219,392,69,49],
                    [73,104,121,69,120,151,49,42],
                    [29,39,40,41,41,43,29,24,13,37,29,40,35,11,32,35,28,
                     25,36,3,35,28,17,9,13,17],
                    [11,9,10,14,11,13,9,11,9,12,16,12,12,18,12,11,13,16,
                     15,10,12,10,11,9,11,10,11,13,13,10,19,13,8,13,11,17,
                     10,9,6,5,12,14,9,10,11,8,14,9,6,7,10,12,9,11,5,8,10,
                     9,9,1,10,4,6,3,4,16,1,3,6,4,4,4,3,8,4],
                    [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,4,3,4,4,
                     3,4,4,4,4,4,4,3,4,4,4,4,4,4,3,4,4,3,4,4,4,4,4,4,4,4,
                     4,4,4,4,3,4,4,4,4,4,4,4,4,3,4,2,4,3,4,3,4,3,4,4,4,4,
                     3,3,4,2,3,2,4,3,2,4,3,3,3,3,4,4,3,2,3,4,4,3,...]],
          ClperLevRS=[2,4,8,26,75,237,729],Arcs_traversedRS=635669,
          avSolLengthRS=52.6059670781893,cRS=2.72842745861733,
          dectimeRS=TIME {sec=104,usec=60000},levelsRS=7,
          ratioTsumRS=4.1438927242029,
          searchtimeRS=TIME {sec=8065,usec=820000},sumlengthRS=153399},
  STANDARD={ClSizesSd=[[243,243,243],[81,81,81,81,81,81,81,81,81],
                  [27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,
                   27,27,27,27,27,27,27,27,27],
                  [9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
                   9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
                   9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
                   9],
                  [3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
                   3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
                   3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
                   3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,...]],
        ClperLevSd=[3,9,27,81,243,729],Arcs_traversedSd=235379,
        avSolLengthSd=34.6286008230453,cSd=3.0,
        dectimeSd=TIME {sec=37,usec=620000},levelsSd=6,
        ratioTsumSd=2.33101597393466,
        searchtimeSd=TIME {sec=2962,usec=480000},sumlengthSd=100977},
 STAR={ClSizesS=[[349,380],[194,182,145,98,10,15,43,26,16],
             [50,46,58,63,65,53,65,45,44,57,36,43,26,10,16,8,16,15,13],
             [15,17,16,16,17,19,17,15,17,20,16,16,15,15,18,17,17,18,13,
              19,15,18,15,16,17,14,11,16,13,12,13,13,10,13,13,13,12,14,
              11,11,11,10,10,8,10,9,7,8,8,8,5,5,8,3,4,2,4,4,2],
             [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
              4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
              4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
              4,4,4,4,4,4,4,4,4,4,4,4,4,4,...]],
     ClperLevS=[2,9,19,59,222,729],Arcs_traversedS=808648,
     avSolLengthS=66.1292866941015,cS=3.35257398343928,
     dectimeS=TIME {sec=100,usec=850000},levelsS=6,
     ratioTsumS=4.19351459553085,searchtimeS=TIME {sec=5331,usec=750000},
     sumlengthS=192833}}
```

# C.2 The Arrow Puzzle

ARR9

```
val it =
  {STAR_ABS={ClSizesA1=[[141,143,86,86,29,27],
               [10,10,10,11,9,9,10,9,9,9,10,11,10,10,9,9,9,9,9,9,10,9,9,
                10,10,10,9,10,10,9,10,10,10,8,8,10,7,7,7,7,7,8,7,7,8,7,6,
                5,5,6,5,6,6,5,5,5,4,5,6,4,4,4,4,3,3,2,2,2]],
      ClperLevA1=[6,68,512],Arcs_traversedA1=366436,
      avSolLengthA1=9.6953125,cA1=9.43137254901961,
      dectimeA1=TIME {sec=352,usec=50000},levelsA1=3,
      ratioTsumA1=36.9093473005641,searchtimeA1=TIME {sec=328,usec=610000},
      sumlengthA1=9928},
   RAND_EDGE={ClSizesA2=[[128,128,128,128],[64,64,64,64,64,64,64,64],
               [32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32],
               [16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,
                16,16,16,16,16,16,16,16,16,16,16,16,16,16],
               [8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                8,8,8,8,8,8],
               [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...],
               [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
      ClperLevA2=[4,8,16,32,64,128,256,512],Arcs_traversedA2=791920,
      avSolLengthA2=5.0078125,cA2=2.0,dectimeA2=TIME {sec=262,usec=990000},
      levelsA2=8,ratioTsumA2=154.430577223089,
      searchtimeA2=TIME {sec=483,usec=770000},sumlengthA2=5128},
   DOUBLE_STAR={ClSizesMA2=[[186,202,70,30,20,2,2],
                 [16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,
                  12,12,12,12,12,12,12,12,8,8,8,8,8,6,6,6,6,4,4,4,4,4,4,
                  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]],
        ClperLevMA2=[7,61,512],Arcs_traversedMA2=440030,
        avSolLengthMA2=10.62890625,cMA2=8.55386416861827,
        dectimeMA2=TIME {sec=350,usec=370000},levelsMA2=3,
        ratioTsumMA2=40.4290701947813,
        searchtimeMA2=TIME {sec=530,usec=340000},sumlengthMA2=10884},
   EDGE={ClSizesCO=[[128,128,128,128],[64,64,64,64,64,64,64,64],
                [32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32],
                [16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,
                 16,16,16,16,16,16,16,16,16,16,16,16,16,16],
                [8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                 8,8,8,8,8,8,8,8],
                [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...],
                [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
```

```
        ClperLevCO=[4,8,16,32,64,128,256,512],Arcs_traversedCO=791920,
        avSolLengthCO=5.0078125,cCO=2.0,
        dectimeCO=TIME {sec=265,usec=290000},levelsCO=8,
        ratioTsumCO=154.430577223089,
        searchtimeCO=TIME {sec=483,usec=470000},sumlengthCO=5128},
   ORGINAL={ClSizesO=[[1]],ClperLevO=[512],Arcs_traversedO=606152,
         avSolLengthO=4.0,cO=1.0,dectimeO=TIME {sec=9,usec=660000},
         levelsO=1,ratioTsumO=147.986328125,
         searchtimeO=TIME {sec=728,usec=250000},sumlengthO=4096},
   RAND_STAR={ClSizesRS=[[108,147,256,1],
                    [92,48,79,78,21,25,15,24,13,7,11,24,23,12,13,2,1,1,6,
                     4,5,1,1,2,1,3],
                    [9,9,9,9,9,9,9,9,3,9,9,9,7,7,9,7,7,7,5,7,7,9,7,7,6,7,
                     7,7,5,7,4,6,4,7,4,5,3,5,4,5,2,7,7,3,5,3,4,6,2,9,2,5,
                     4,4,6,4,4,5,4,7,1,3,5,4,4,4,5,5,6,3,4,1,5,2,1,2,3,3,
                     2,4,2,1,2,1,1,3,2,3,1,1,3,1,3,1,1,4,1,2,3,2,...]],
          ClperLevRS=[4,26,134,512],Arcs_traversedRS=308766,
          avSolLengthRS=8.162109375,cRS=5.1582472254114,
          dectimeRS=TIME {sec=285,usec=80000},levelsRS=4,
          ratioTsumRS=36.9425699928212,
          searchtimeRS=TIME {sec=358,usec=250000},sumlengthRS=8358},
   STAR={ClSizesS=[[255,254,1,1,1],
              [79,90,81,97,61,37,27,8,10,10,1,1,1,2,1,1,1,1,1,1,1],
              [9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
               9,9,9,9,7,7,7,7,7,7,7,7,7,7,7,7,7,7,5,5,5,5,5,5,5,5,5,5,4,4,
               4,4,4,4,3,3,3,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
               1,1,1,1,1,1,1,1,1,1,1,1,1,...]],ClperLevS=[5,21,103,512],
      Arcs_traversedS=359157,avSolLengthS=10.369140625,
      cS=4.69187856372322,dectimeS=TIME {sec=340,usec=930000},levelsS=4,
      ratioTsumS=33.8252966660388,searchtimeS=TIME {sec=466,usec=990000},
      sumlengthS=10618}}
```

## C.3 The 5-Puzzle

```
val it =
  {STAR_ABS={ClSizesA1=[[188,165,111,93,61,40,62],
                [23,24,19,22,29,19,20,24,25,27,23,31,35,29,27,20,19,19,24,
                 23,23,18,29,19,17,19,14,15,14,12,12,10,11,6,6,6,7],
                [5,4,5,5,6,6,5,6,5,5,6,5,5,6,6,4,5,6,5,5,5,5,4,5,4,4,6,4,5,
                 5,5,6,5,4,5,6,4,4,5,6,5,5,4,6,4,4,5,5,6,5,5,5,5,5,6,4,5,6,
                 5,4,5,4,5,4,5,4,5,5,5,5,5,4,5,6,5,4,6,4,5,4,4,5,4,6,4,4,6,
                 5,6,5,4,5,5,4,5,4,4,4,5,5,...]],ClperLevA1=[7,37,184,720],
      Arcs_traversedA1=158462,avSolLengthA1=29.3458333333333,
      cA1=4.72391024564938,dectimeA1=TIME {sec=550,usec=90000},levelsA1=4,
      ratioTsumA1=3.74986984712954,searchtimeA1=TIME {sec=2272,usec=360000},
      sumlengthA1=42258},
   RAND_EDGE={ClSizesA2=[[204,204,156,156],[48,48,48,48,54,54,54,54,48,48,108,108],
                [24,24,24,24,30,30,30,30,24,24,42,42,36,36,30,30,24,24,24,
                 24,24,24,24,24,24,24],
                [12,12,12,12,12,12,18,18,12,12,18,18,12,12,12,12,18,18,18,
                 18,18,18,18,18,12,12,12,12,12,12,12,12,12,12,12,12,12,
                 12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12],
                [6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                 6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                 6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                 6,6,6,6,6,6,6,6,6,6,6,6,6,...],
```

[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
ClperLevA2=[4,12,26,54,120,360,720],Arcs_traversedA2=212746,
avSolLengthA2=22.0375,cA2=2.41096866096866,
dectimeA2=TIME {sec=590,usec=350000},levelsA2=7,
ratioTsumA2=6.70403983109599,searchtimeA2=TIME {sec=3058,usec=370000},
sumlengthA2=31734},
DOUBLE_STAR={ClSizesMA2=[[219,234,85,64,38,34,30,6,2,1,1,1,1,1,1,1,1],
[28,24,24,23,23,26,27,28,29,22,26,23,26,27,22,25,22,22,
19,20,18,17,17,16,15,13,12,13,12,11,8,8,7,7,7,6,6,7,7,
2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,4,4,...]],
ClperLevMA2=[17,62,231,720],Arcs_traversedMA2=146331,
avSolLengthMA2=23.4166666666667,cMA2=3.49658279734181,
dectimeMA2=TIME {sec=1764,usec=210000},levelsMA2=4,
ratioTsumMA2=4.33959074733096,
searchtimeMA2=TIME {sec=2637,usec=150000},sumlengthMA2=33720},
EDGE={ClSizesCO=[[198,198,162,162],[162,162,156,156,36,36,6,6],
[84,84,78,78,78,78,78,78,36,36,6,6],
[48,48,36,36,48,48,42,42,36,36,30,30,36,36,42,42,36,36,6,
6],
[24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,
18,18,18,18,18,18,24,24,18,18,12,12,12,12,12,12,6,6,6,
6],
[12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,6,
6,6,6,6,6,6,6,6,6,6,6],
[6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,...],
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,...],
[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
ClperLevCO=[4,8,12,20,38,66,120,240,360,720],
Arcs_traversedCO=385214,avSolLengthCO=20.3916666666667,
cCO=1.79129895445685,dectimeCO=TIME {sec=677,usec=660000},
levelsCO=10,ratioTsumCO=13.1185805748536,
searchtimeCO=TIME {sec=4373,usec=80000},sumlengthCO=29364},
ORGINAL={ClSizesO=[[1]],ClperLevO=[720],Arcs_traversedO=520204,
avSolLengthO=12.4444444444444,cO=1.0,
dectimeO=TIME {sec=28,usec=200000},levelsO=1,
ratioTsumO=29.0292410714286,
searchtimeO=TIME {sec=4146,usec=570000},sumlengthO=17920},
RAND_STAR={ClSizesRS=[[312,8,2,2,232,23,112,9,2,10,6,2],
[119,96,227,92,80,32,10,2,6,2,9,2,5,13,10,8,2,5],
[34,40,40,31,33,28,16,40,24,40,32,22,26,27,26,2,40,
18,6,12,5,28,8,2,2,10,2,10,9,5,1,17,14,12,8,2,9,12,

```
                5,1,10,6,5],
               [8,8,11,10,10,9,15,9,8,6,11,11,9,8,6,14,11,9,5,8,8,
                12,6,10,4,7,9,10,9,7,5,12,7,8,6,5,7,16,4,11,9,12,5,
                11,1,4,5,6,11,8,9,6,12,10,2,5,5,10,4,6,9,5,6,4,8,6,
                6,11,6,6,7,10,4,5,3,4,2,9,5,2,5,1,2,3,5,7,4,4,5,3,6,
                1,5,2,2,2,4,2,2,4,...],
               [3,3,4,4,3,4,3,4,3,2,3,3,3,4,3,3,3,4,3,3,3,3,3,4,2,4,
                3,3,4,3,3,3,3,3,3,3,4,3,4,4,3,3,3,3,3,3,4,4,2,3,3,2,
                3,3,3,2,3,3,4,2,4,3,4,3,1,3,3,2,4,4,3,2,4,3,2,3,3,2,
                3,3,3,3,3,3,4,3,3,2,3,3,3,3,3,4,4,3,3,2,3,2,...]],
         ClperLevRS=[12,18,43,115,298,720],Arcs_traversedRS=192302,
         avSolLengthRS=22.4888888888889,cRS=2.31414384478329,
         dectimeRS=TIME {sec=593,usec=220000},levelsRS=6,
         ratioTsumRS=5.93817934782609,
         searchtimeRS=TIME {sec=3114,usec=310000},sumlengthRS=32384},
  STAR={ClSizesS=[[357,358,2,1,2],
            [67,85,73,60,70,56,47,50,35,36,27,25,25,18,15,13,5,2,1,1,6,
             1,2],
            [16,15,15,9,10,14,11,12,12,15,14,12,13,15,11,11,12,10,16,9,
             12,13,13,11,15,11,12,13,15,9,8,10,12,7,11,8,12,8,8,8,11,8,
             13,8,11,9,6,7,8,10,9,7,9,6,7,7,7,9,12,6,5,5,5,5,5,5,5,2,1,
             2,4,2,4,1,2,4,4,1,2,2,1,1,1,2,2,1,2,1,2,2,1],
            [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,4,4,4,4,...]],
      ClperLevS=[5,23,91,296,720],Arcs_traversedS=131401,
      avSolLengthS=23.9041666666667,cS=3.56042535607753,
      dectimeS=TIME {sec=496,usec=260000},levelsS=5,
      ratioTsumS=3.81735517982685,searchtimeS=TIME {sec=2357,usec=20000},
      sumlengthS=34422}}
```

# C.4 The Non-Prim Inv Navigation Problem

NAV10

```
val it =
 {STAR_ABS={ClSizesA1=[[316,84],[61,41,45,42,35,35,36,28,18,19,13,14,13],
            [5,5,5,5,6,6,5,5,5,6,5,5,5,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
             5,5,5,5,5,5,5,5,5,5,7,5,5,5,6,5,6,6,5,5,5,5,5,4,5,4,4,4,4,
             4,4,4,4,4,5,4,4,4,3,3,4,3,3,3,3,3,3,3,3,4,3,3,3,3,3,2,2,2,
             2,2,2,2,2]],ClperLevA1=[2,13,92,400],
     Arcs_traversedA1=292484,avSolLengthA1=24.61125,cA1=5.97491638795987,
     dectimeA1=TIME {sec=186,usec=790000},levelsA1=4,
     ratioTsumA1=7.42759916704759,searchtimeA1=TIME {sec=2230,usec=850000},
     sumlengthA1=39378},
  RAND_EDGE={ClSizesA2=[[128,128,144],[64,64,64,64,64,80],
            [32,32,32,32,32,32,32,32,32,32,32,48],
            [16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,
             16,16,16,16,16,16],
            [8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
             8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8],
            [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,4,4,4,4],
```

[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
ClperLevA2=[3,6,12,25,50,100,200,400],Arcs_traversedA2=464270,
avSolLengthA2=14.07,cA2=2.01190476190476,
dectimeA2=TIME {sec=138,usec=740000},levelsA2=8,
ratioTsumA2=20.623223169865,searchtimeA2=TIME {sec=2376,usec=10000},
sumlengthA2=22512},
DOUBLE_STAR={ClSizesMA2=[[395,2,1,1,1],
[94,65,61,45,40,25,17,21,15,4,2,2,2,2,1,1,1,1,1],
[8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,7,7,7,7,7,7,7,7,7,
7,7,7,7,7,7,7,7,6,6,6,6,6,6,5,5,5,4,4,4,4,4,3,3,3,3,3,
3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1]],ClperLevMA2=[5,19,88,400],
Arcs_traversedMA2=264313,avSolLengthMA2=15.45375,
cMA2=4.32567783094099,dectimeMA2=TIME {sec=318,usec=920000},
levelsMA2=4,ratioTsumMA2=10.6896788805306,
searchtimeMA2=TIME {sec=1709,usec=570000},sumlengthMA2=24726},
EDGE={ClSizesCO=[[256,144],[128,128,128,16],[64,64,64,64,64,64,16],
[32,32,32,32,32,32,32,32,32,32,32,32,16],
[16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,
16,16,16,16,16,16,16],
[8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8],
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4],
[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
ClperLevCO=[2,4,7,13,25,50,100,200,400],Arcs_traversedCO=565943,
avSolLengthCO=15.135,cCO=1.94127747252747,
dectimeCO=TIME {sec=170,usec=460000},levelsCO=9,
ratioTsumCO=23.3706227287744,
searchtimeCO=TIME {sec=3598,usec=560000},sumlengthCO=24216},
ORGINAL={ClSizesO=[[1]],ClperLevO=[400],Arcs_traversedO=1039722,
avSolLengthO=9.1,cO=1.0,dectimeO=TIME {sec=8,usec=570000},
levelsO=1,ratioTsumO=71.409478021978,
searchtimeO=TIME {sec=5193,usec=290000},sumlengthO=14560},
RAND_STAR={ClSizesRS=[[398,1,1],[397,1,1,1],[396,1,1,1,1],[395,1,1,1,1,1],
[330,65,1,1,1,1,1],
[80,61,45,73,1,1,79,16,37,1,1,1,4],
[11,24,21,12,25,30,17,9,14,12,19,10,8,1,11,16,16,9,7,
13,13,1,4,12,1,6,12,7,9,4,4,5,7,3,6,1,1,2,5,2,1,6,2,
1],
[5,5,5,4,3,5,4,5,5,5,5,5,3,1,5,5,5,5,4,5,5,4,5,5,3,5,
4,5,4,4,5,4,3,4,4,4,2,3,2,3,5,2,4,4,2,4,3,3,4,2,5,5,
4,3,5,4,3,5,5,4,3,3,3,3,2,3,2,3,4,4,3,2,3,4,2,5,4,3,
2,2,4,2,2,2,1,3,1,2,3,2,3,2,1,2,2,2,1,3,4,1,...]],
ClperLevRS=[3,4,5,6,7,13,44,141,400],Arcs_traversedRS=625816,
avSolLengthRS=21.395,cRS=2.02914789111598,
dectimeRS=TIME {sec=335,usec=160000},levelsRS=9,
ratioTsumRS=18.2816078523019,
searchtimeRS=TIME {sec=18472,usec=370000},sumlengthRS=34232},
STAR={ClSizesS=[[274,59,63,4],
[40,40,38,36,31,27,19,28,20,23,20,18,9,14,10,4,5,4,5,5,4],

```
            [5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
             5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,
             4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,2,
             2,2,2,2,2,1,1,1,1,1,1,1,1,...]],ClperLevS=[4,21,105,400],
    Arcs_traversedS=295014,avSolLengthS=17.2025,cS=4.68650793650794,
    dectimeS=TIME {sec=238,usec=360000},levelsS=4,
    ratioTsumS=10.7184275541346,searchtimeS=TIME {sec=1753,usec=300000},
    sumlengthS=27[illegible]24}}
```

# C.5 The Missionaries and Cannibals

MC100

```
val it =
 {STAR_ABS={ClSizesA1=[[134,79,88],[28,57,29,38,30,67,31,21],
              [21,10,9,13,14,9,15,17,11,9,9,11,11,10,12,8,13,16,14,10,12,
               14,12,6,8,7],
              [7,7,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,6,5,
               4,4,4,4,4,3,3,3,4,3,3,3,3,3,3,3,3,3,3,3,3,4,4,3,3,4,3,3,
               3,3,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]],
     ClperLevA1=[3,8,26,81,301],Arcs_traversedA1=240404,
     avSolLengthA1=68.2101328903654,cA1=3.18702516619183,
     dectimeA1=TIME {sec=41,usec=970000},levelsA1=5,
     ratioTsumA1=2.92729375951294,searchtimeA1=TIME {sec=1427,usec=20000},
     sumlengthA1=82125},
 RAND_EDGE={ClSizesA2=[[143,158],[79,66,92,64],[47,32,32,34,32,60,32,32],
             [19,16,16,16,16,16,18,16,16,16,16,28,16,16,16,16,28],
             [11,8,8,8,8,8,8,8,8,8,8,8,8,10,8,8,8,8,8,8,8,8,8,8,12,8,8,
              8,8,8,8,8,8,8,8,12],
             [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
              4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,6,4,4,4,4,4,4,4,4,4,
              4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,7],
             [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
              2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
              2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
              2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
    ClperLevA2=[2,4,8,17,36,74,150,301],Arcs_traversedA2=205427,
    avSolLengthA2=78.8264119601329,cA2=2.04741375829611,
    dectimeA2=TIME {sec=56,usec=0},levelsA2=8,
    ratioTsumA2=2.16450841349953,searchtimeA2=TIME {sec=2383,usec=280000},
    sumlengthA2=94907},
 DOUBLE_STAR={ClSizesMA2=[[136,113,52],[66,66,41,36,34,22,18,12,6],
               [19,24,22,22,16,24,20,16,12,11,10,14,11,12,11,11,8,6,6,
                6,6,6,4,4],
               [6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,
                3,3,3,3,3,2,2,2,2,2,2,1,1,1,1,1,1,1]],
     ClperLevMA2=[3,9,24,72,301],Arcs_traversedMA2=266741,
     avSolLengthMA2=67.6395348837209,cMA2=3.21180555555556,
     dectimeMA2=TIME {sec=161,usec=800000},levelsMA2=5,
     ratioTsumMA2=3.2753874112822,
     searchtimeMA2=TIME {sec=1417,usec=270000},sumlengthMA2=81438},
EDGE={ClSizesCO=[[237,64],[173,64,64],[79,94,64,64],[63,62,64,64,32,16],
            [31,32,32,30,32,32,32,32,32,16],
            [15,16,16,16,16,16,14,16,16,16,16,16,16,16,16,16,16,16,
             16],
```

```
                 [7,8,8,8,8,8,8,8,8,8,8,8,8,6,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                  8,8,8,8,8,8,8,8,8,8],
                 [4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                  4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                  4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,2],
                 [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
                  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,...]],
      ClperLevCO=[2,3,4,6,10,19,38,76,151,301],Arcs_traversedCO=217055,
      avSolLengthCO=80.6569767744186,cCO=1.76446884318965,
      dectimeCO=TIME {sec=63,usec=260000},levelsCO=10,
      ratioTsumCO=2.23512269464839,
      searchtimeCO=TIME {sec=3070,usec=60000},sumlengthCO=97111},
  ORGINAL={ClSizesO=[[1]],ClperLevO=[301],Arcs_traversedO=561913,
       avSolLengthO=64.4950166112957,cO=1.0,
       dectimeO=TIME {sec=15,usec=320000},levelsO=1,
       ratioTsumO=7.2362978416525,searchtimeO=TIME {sec=2533,usec=0},
       sumlengthO=77652},
  RAND_STAR={ClSizesRS=[[243,58],[79,95,58,69],[42,45,31,38,27,38,41,27,12],
                  [18,13,27,30,12,18,21,15,13,20,19,19,6,12,14,11,23,
                   10],
                  [9,12,8,7,5,12,11,12,6,10,7,5,7,10,8,6,12,7,11,7,10,
                   6,3,9,5,14,4,3,7,12,5,11,10,3,5,7,2,2,5,3,3],
                  [5,3,3,5,5,5,3,5,5,5,5,4,3,5,3,5,5,5,5,5,5,3,4,5,5,3,
                   3,5,3,3,1,3,3,3,4,5,2,3,5,4,5,3,5,2,3,3,3,2,5,2,3,2,
                   4,2,4,1,4,2,4,5,3,2,2,2,2,3,5,2,2,2,3,2,2,2,2,1,1,2,
                   2,1,4,1,2,2,3,1,3,3,1,2,2,1,1,2,1,1,2,1]],
         ClperLevRS=[2,4,9,18,41,98,301],Arcs_traversedRS=252725,
         avSolLengthRS=69.3322259136213,cRS=2.3315750419409,
         dectimeRS=TIME {sec=46,usec=970000},levelsRS=7,
         ratioTsumRS=3.02751689108247,
         searchtimeRS=TIME {sec=1860,usec=990000},sumlengthRS=83476},
  STAR={ClSizesS=[[156,125,20],[68,93,42,57,21,20],
            [15,37,21,27,16,24,24,18,29,22,20,22,12,8,6],
            [13,13,12,16,10,7,14,13,6,9,8,7,12,5,12,9,10,12,12,8,7,9,10,
             6,9,6,7,6,7,8,3,3,1,5,4,2],
            [6,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
             4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
             3,3,3,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,
             1,1,1,1,1]],ClperLevS=[3,6,15,36,92,301],
     Arcs_traversedS=244960,avSolLengthS=67.9767441860465,
     cS=2.54545893719807,dectimeS=TIME {sec=46,usec=660000},levelsS=6,
     ratioTsumS=2.99301109427692,searchtimeS=TIME {sec=1775,usec=180000},
     sumlengthS=81844}}
```

## C.6 The Navigation Problem

NAV(10,10)

```
val it =
  {DOUBLE_IO_STRONG_STAR={ClSizesDIO=[[400],
                             [1,1,182,1,1,1,1,84,1,1,1,1,1,1,1,84,1,1,
                              1,1,1,8,4,4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                              1,1],
                             [1,1,1,1,20,1,1,1,1,1,1,1,1,20,1,1,1,1,1,
```

```
                                1,1,24,1,1,1,1,1,24,1,1,1,1,1,20,1,1,1,
                                1,1,1,16,1,1,1,1,1,16,1,1,1,1,1,16,1,1,
                                1,1,16,1,1,1,1,12,1,1,1,1,1,1,1,8,1,1,1,
                                1,1,1,1,1,8,1,1,1,1,18,1,1,12,1,1,1,1,1,
                                8,1,1,14,1,1,8,...]],
                        ClperLevDIO=[1,40,132,400],
                        Arcs_traversedDIO=259897,avSolLengthDIO=22.40125,
                        cDIO=15.4434343434343,
                        dectimeDIO=TIME {sec=143,usec=10000},levelsDIO=4,
                        ratioTsumDIO=7.25118575972323,
                        searchtimeDIO=TIME {sec=850,usec=680000},
                        sumlengthDIO=35842},
 ORGINAL={ClSizesO=[[1]],ClperLevO=[400],Arcs_traversedO=530177,
     avSolLengthO=11.2075,cO=1.0,dectimeO=TIME {sec=1,usec=180000},
     levelsO=1,ratioTsumO=29.5659714476913,
     searchtimeO=TIME {sec=1010,usec=460000},sumlengthO=17932},
 STRONG_STAR={ClSizesA1=[[400]],ClperLevA1=[1,400],Arcs_traversedA1=530177,
      avSolLengthA1=11.2075,cA1=400.0,
      dectimeA1=TIME {sec=38,usec=150000},levelsA1=2,
      ratioTsumA1=29.5659714476913,
      searchtimeA1=TIME {sec=1020,usec=40000},sumlengthA1=17932},
 IO_STRONG_STAR={ClSizesRS=[[400]],ClperLevRS=[1,400],
          Arcs_traversedRS=530177,avSolLengthRS=11.2075,
          cRS=400.0,dectimeRS=TIME {sec=121,usec=500000},
          levelsRS=2,ratioTsumRS=29.5659714476913,
          searchtimeRS=TIME {sec=1013,usec=650000},
          sumlengthRS=17932},
 DOUBLE_STRONG_STAR={ClSizesA2=[[400]],ClperLevA2=[1,400],
          Arcs_traversedA2=530177,avSolLengthA2=11.2075,
          cA2=400.0,dectimeA2=TIME {sec=63,usec=570000},
          levelsA2=2,ratioTsumA2=29.5659714476913,
          searchtimeA2=TIME {sec=1014,usec=540000},
          sumlengthA2=17932}}
```

# C.7 The Non-Prim Inv Towers of Hanoi

TOH6

```
val it =
 {DOUBLE_IO_STRONG_STAR={ClSizesDIO=[[729],[243,81,27,81,27,243,27],
                          [1,81,27,27,27,27,9,3,8,81,27,81,27,27,
                           27,80,9,1,72,27,9,3,3,9,9,27],
                          [9,1,9,3,8,9,9,9,9,9,9,9,9,9,9,9,3,8,9,3,
                           9,9,9,8,9,3,9,9,9,9,8,9,26,3,9,3,27,8,9,
                           3,9,3,9,27,1,3,9,9,9,9,9,9,8,27,1,9,1,9,
                           9,9,8,9,9,9,9,1,9,9,9,9,3,3,9,3,3,3,3,3,
                           3,3,9,9,3,3,1,9,1,3,3,8,9,3,3,3,3,9,1,8,
                           1,1,...],
                          [9,1,1,9,3,9,1,3,3,3,3,1,3,3,8,3,3,3,1,9,
                           3,3,1,9,1,3,1,1,9,1,3,3,9,3,1,1,9,3,9,1,
                           3,9,3,8,3,1,9,3,9,3,1,9,1,9,3,3,3,1,3,3,
                           1,9,3,3,1,1,9,1,1,9,1,9,1,3,1,1,1,9,3,3,
                           9,3,1,9,3,3,1,3,8,1,3,9,1,3,3,3,1,3,1,1,
                           ...]],ClperLevDIO=[1,7,26,101,215,729],
                 Arcs_traversedDIO=1087538,
                 avSolLengthDIO=210.950617283951,
```

```
                cDIO=4.02366232892137,
                dectimeDIO=TIME {sec=14966,usec=600000},levelsDIO=6,
                ratioTsumDIO=1.76797500373903,
                searchtimeDIO=TIME {sec=7758,usec=970000},
                sumlengthDIO=615132},
ORGINAL={ClSizesO=[[1]],ClperLevO=[729],Arcs_traversedO=1579631,
     avSolLengthO=210.950617283951,cO=1.0,
     dectimeO=TIME {sec=13,usec=540000},levelsO=1,
     ratioTsumO=2.5679545203306,searchtimeO=TIME {sec=7021,usec=700000},
     sumlengthO=615132},
STANDARD={ClSizesSd=[[243,243,243],[81,81,81,81,81,81,81,81,81],
               [27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,
                27,27,27,27,27,27,27,27,27],
               [9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
                9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
                9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
                9],
               [3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
                3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
                3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
                3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,...]],
      ClperLevSd=[3,9,27,81,243,729],Arcs_traversedSd=671167,
      avSolLengthSd=210.950617283951,cSd=3.0,
      dectimeSd=TIME {sec=34,usec=120000},levelsSd=6,
      ratioTsumSd=1.09109426919751,
      searchtimeSd=TIME {sec=6400,usec=270000},sumlengthSd=615132},
STRONG_STAR={ClSizesA1=[[729]],ClperLevA1=[1,729],Arcs_traversedA1=1579631,
       avSolLengthA1=210.950617283951,cA1=729.0,
       dectimeA1=TIME {sec=234,usec=880000},levelsA1=2,
       ratioTsumA1=2.5679545203306,
       searchtimeA1=TIME {sec=7041,usec=180000},sumlengthA1=615132},
IO_STRONG_STAR={ClSizesRS=[[729],[243,81,81,243,81],
                    [81,81,81,81,27,27,27,27,27,27,81,27,27,27,
                     81],
                    [27,9,27,9,27,9,81,3,27,27,9,9,27,9,27,9,9,9,
                     3,3,9,9,81,27,27,27,9,27,27,27,9,9,3,3,9,9,
                     9,3,9,9,9,9,9],
                    [3,3,9,9,3,9,9,9,9,9,9,1,9,3,9,9,9,3,3,3,3,3,
                     3,1,3,3,3,9,9,9,3,3,3,9,9,9,3,9,9,3,3,3,9,3,
                     9,3,3,1,3,3,9,3,3,3,9,9,9,9,3,9,9,3,3,3,9,9,
                     3,3,1,1,3,3,9,9,9,9,9,9,3,3,9,3,9,9,3,9,9,3,
                     9,3,1,3,3,1,9,9,9,3,3,9,...],
                    [3,3,3,3,3,3,3,3,3,1,3,3,3,3,3,1,3,3,3,3,1,3,
                     1,3,3,3,3,3,3,3,3,3,1,1,1,3,3,3,3,3,3,3,3,3,
                     1,3,3,1,1,3,3,1,3,3,3,3,3,3,1,3,3,3,3,3,3,3,
                     1,3,3,3,1,1,3,3,3,3,3,3,9,1,3,3,3,3,3,3,1,1,
                     3,3,3,3,3,1,3,3,3,1,3,3,...]],
              ClperLevRS=[1,5,15,43,141,297,729],
              Arcs_traversedRS=989665,
              avSolLengthRS=210.950617283951,cRS=3.11777747789623,
              dectimeRS=TIME {sec=1160,usec=490000},levelsRS=7,
              ratioTsumRS=1.60886606451949,
              searchtimeRS=TIME {sec=10980,usec=370000},
              sumlengthRS=615132},
DOUBLE_STRONG_STAR={ClSizesA2=[[729],
                    [3,1,9,1,1,3,3,9,3,27,27,9,9,9,1,1,3,3,3,9,9,
                     27,9,3,9,9,1,9,1,1,3,1,27,3,3,9,1,27,9,3,3,
                     9,9,9,9,1,1,3,1,9,3,9,1,9,1,1,3,3,1,9,1,1,3,
                     3,9,9,1,1,3,1,1,9,1,9,3,3,3,1,1,9,3,3,1,1,9,
```

```
                    3,3,1,1,9,1,1,27,9,9,1,1,3,9,1,...],
                    [3,1,9,1,1,3,3,3,9,1,9,9,1,3,3,3,3,9,1,1,3,1,
                     1,3,1,1,3,1,9,3,3,1,9,1,1,3,1,1,3,3,9,3,9,9,
                     9,3,3,1,1,3,3,1,9,1,9,1,1,3,3,3,3,1,9,9,1,3,
                     3,1,9,3,3,1,1,3,3,1,1,3,3,1,1,3,3,3,9,3,3,3,
                     1,3,3,1,1,1,1,3,3,1,1,3,...]],
             ClperLevA2=[1,141,295,729],Arcs_traversedA2=1251730,
             avSolLengthA2=210.950617283951,cA2=48.5211283407461,
             dectimeA2=TIME {sec=551,usec=850000},levelsA2=4,
             ratioTsumA2=2.03489657504406,
             searchtimeA2=TIME {sec=5263,usec=870000},
             sumlengthA2=615132}}
```

# C.8 The Water Jug Problem

```
JUG(50,3)

val it =
  {DOUBLE_IO_STRONG_STAR={ClSizesDIO=[[106,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                    1]],ClperLevDIO=[99,204],
                  Arcs_traversedDIO=105551,
                  avSolLengthDIO=17.2358490566038,
                  cDIO=2.06060606060606,
                  dectimeDIO=TIME {sec=11,usec=340000},levelsDIO=2,
                  ratioTsumDIO=14.4432129173508,
                  searchtimeDIO=TIME {sec=41,usec=90000},
                  sumlengthDIO=7308},
   ORGINAL={ClSizesO=[[1]],ClperLevO=[204],Arcs_traversedO=105551,
          avSolLengthO=17.2358490566038,cO=1.0,
          dectimeO=TIME {sec=1,usec=260000},levelsO=1,
          ratioTsumO=14.4432129173508,searchtimeO=TIME {sec=40,usec=190000},
          sumlengthO=7308},
   STRONG_STAR={ClSizesA1=[[106,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
                 [20,47,34,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  ...]],ClperLevA1=[99,102,204],
          Arcs_traversedA1=87957,avSolLengthA1=24.4551886792453,
          cA1=1.51515151515152,dectimeA1=TIME {sec=19,usec=570000},
          levelsA1=3,ratioTsumA1=8.48268878387501,
          searchtimeA1=TIME {sec=43,usec=480000},sumlengthA1=10369},
  IO_STRONG_STAR={ClSizesRS=[[106,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                          1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                          1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                          1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                          1,1,1,1,1,1,1,1,1,1,1,1,1,1]],
                ClperLevRS=[99,204],Arcs_traversedRS=105551,
                avSolLengthRS=17.2358490566038,cRS=2.06060606060606,
```

```
                decdtimeRS=TIME {sec=14,usec=370000},levelsRS=2,
                ratioTsumRS=14.4432129173508,
                searchtimeRS=TIME {sec=40,usec=570000},
                sumlengthRS=7308},
DOUBLE_STRONG_STAR={ClSizesA2=[[106,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1],
                       [68,38,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                        1,1,1,1,1,1,1,1,1,1,1,1,1]],
                ClperLevA2=[99,100,204],Arcs_traversedA2=93677,
                avSolLengthA2=25.2429245283019,cA2=1.52505050505051,
                dectimeA2=TIME {sec=24,usec=220000},levelsA2=3,
                ratioTsumA2=8.75240586751378,
                searchtimeA2=TIME {sec=54,usec=900000},
                sumlengthA2=10703}}
```

# Bibliography

[Aho at al, 1983] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. Data Structures and Algorithms. Addison-Wesley, 1983.

[Amarel, 1968] Saul Amarel. On representations of problems of reasoning about actions. In Donald Michie, editor, Machine Intelligence 3, pages 131-171. Edinburgh University Press, Edinburgh, Scotland, 1968.

[Cormen et al, 1990] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. Introduction to Algorithms. Joined production-distribution MIT Press, McGraw-Hill, 1990.

[Etzioni, 1990] Oren Etzioni. A Structural Theory of Explanation-Based Learning. Ph.D. Thesis, School of Computer Science, Canergie Mellon University, 1990. Available as Technical Report CMU-CS-90-185.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:189-208, 1971.

[Fikes et al, 1972] Richard E. Fikes, Peter E. Hart and Nils J. Nilsson. Learning and executing generalized robot plans. Artificial Intelligence, 3:251-288, 1972.

[Giunchinglia and Walsh, 1992] Fausto Giunchinglia and Toby Walsh. A theory of abstraction. Artificial intelligence, 57 (1992): 323-389.

[Holte and Zimmer, 1989] Robert C. Holte and Robert M. Zimmer. A mathematical Framework for studying representation. Proceeding of the sixth international workshop on Machine Learning, Cornell University Ithaca, New York, June 26-27, 1989.

[Holte et al, 1992] Robert Holte, Robert Zimmer and Alan MacDonald. When does changing representation improve problem-solving performance ? NASA Ames Research Center. Tech Report. FIA-92-06, 1992.

[Holte, 1988] Robert C. Holte. An Analytical Framework for Learning Systems. Ph.D. dissertation, Brunel University. also TR-AI88-72, Computer Sciences Dept., Univ. of Texas at Austin.

[Knoblock, 1991] Craig A. Knoblock. Automatically Generating Abstractions for Problem Solving. Technical report CMU-CS-91-120, Computer Science Department, Canergie-Mellon University.

[Knoblock et al, 1992] Craig A. Knoblock, Josh D. Tenenberg and Qiang Yang. The Relationship Between Abstraction and Goal Protection in Planning. Not published.

[Korf, 1985] Richard E. Korf. Learning to solve problems by searching for Macro-Operators. Pitman, Marshfield, MA, 1987.

[Korf, 1987] Richard E. Korf. Macro-operators: A weak method for learning. Artificial Intelligence, 26(1):35-77, 1985.

[Lansky, 1992] Amy L. Lansky. Localization vs. Abstraction: A comparison of Two Search Reduction Techniques. NASA Ames Research Center. Tech Report. F1A-92-06, 1992.

[Lowry, 1989] Michael R. Lowry. Algorithm Synthesis through Problem Reformulation. PhD thesis, Stanford University, 1989.

[Minsky, 1963] Marvin Minsky. Steps towards artificial intelligence. In Edward A. Feigenbaum, editor, Computers and Thought, pages 406-450. McGraw-Hill, New York, NY, 1963.

[Minton, 1985] Steven Minton. Selectively generalizing plans for problem solving. In Proceeding of the Ninth International Joint Conference on Artificial Intelligence, pages 596-599, Los Angeles, CA, 1985.

[Moore, 1959] Edward F.Moore. The shortest path through a maze. In Proceedings of the International Symposium on the theory

of Switching, pages 285-292. Harvard University Press, 1959.

[Mostow and Prieditis, 1989] Jack Mostow and Armand E. Prieditis. Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pages 701-707, Detroit, MI, 1989.

[Newell and Simon, 1972] Allen Newell and Herbert A. Simon. Human Problem Solving. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Newell et al, 1962] Allen Newell, J. C. Shaw, and Herbert A. Simon. The process of creative thinking. In Contemporary Approaches to creative thinking, pages 63-119. Atherton Press, New York, 1962.

[Niizuma and Kitahashi, 1985] Sezaburo Niizuma and Tadahiro Kitahashi. A Problem-Decomposition Method Using Differences or Equivalence Relations Between States. Artificial Intelligence 25 (1985) 117-151.

[Pearl, 1985] Judea Pearl. Intelligent Search Strategies for Computer Problem Solving. Addison-Wesly, 1985.

[Polya, 1945] George Polya. How to Solve It. Princeton University Press,

Princeton, NJ, 1945.

[Prieditis and Janakiraman, 1992] Armand Prieditis and Bhaskar Janakiraman. Becoming Reactive By Concretization. NASA Ames Research Center. Tech Report. F1A-92-06, 1992.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. Artificial Intelligence, 5(2): 115-135, 1974.

[Sacerdoti, 1977] Earl D. Sacerdoti. A Structure for Plans and Behaviour. American Elsevier, New York, NY, 1977.

[Stefik and Conway, 1982] M. Stefik and L. Conway. Towards the principled Engineering of Knowledge, the AI Magazine, vol. 3, no. 3, pages 4-16, 1982.

[Tanimoto, 1990] Steven L. Tanimoto. The Elements of Artificial Intelligence Using Common Lisp. W.H. Freeman and Company, 1990.

[Yang, 1989] Qiang Yang. Improving the Efficiency of Planning. PhD. Thesis, Department of Computer Science, University of Maryland, 1989.