

Abstract

Mehta, Reshma Girish. Visualization Search Strategies. (Under the direction of Dr. Christopher G. Healey)

Innovations in high performance computing and high bandwidth networks have led to the onset of data explosion. Along with large size, the datasets are typically multivariate. The need for effective exploration of this data has led to the area of multidimensional visualization. Research in low level human visual system has resulted in the construction of perceptual guidelines that can produce effective visualizations. However, application of these guidelines to a dataset requires users to be experts in the visualization domain.

ViA is a semi-automated visualization assistant that uses perceptual guidelines along with a heuristic search algorithm to generate perceptually salient visualizations.

This thesis aims to study the behavior of the current hint-based search strategy and determine its efficiency. We compare hint-based search with two generic heuristic search algorithms, simulated annealing and reactive tabu search, by adapting them to ViA's search domain. We use time efficiency, space efficiency, ability to find multiple optimal solutions and optimality as performance metrics. Further, in order to “see” the areas of the search space explored by each search algorithm, we have developed a focus + context visualization system using hyperbolic geometry.

VISUALIZATION SEARCH STRATEGIES

by

Reshma G. Mehta

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER SCIENCE

Raleigh, North Carolina

2004

APPROVED BY:

Chair of Advisory Committee

Biography

Reshma Girish Mehta was born to Girish Mehta and Nila Mehta on October 22nd 1979 in India, Mumbai. She did her schooling in Mumbai. She received a Bachelor of Engineering degree in Computer Engineering from Sardar Patel College of Engineering, Mumbai, India. She then worked for a year at Tata Institute of Fundamental Research, Mumbai, India. Reshma is currently enrolled in the masters program in Computer Science at North Carolina State University. She plans to join industry after her graduation.

Acknowledgments

This thesis would not have been completed without the guidance and support of my advisor Dr. Christopher Healey. He has worked more than me on this thesis. I thank him for his precious time. I thank him for the times when he helped me believe in my abilities. I owe a lot to you Sir. Thank You.

I am greatly thankful to my parents who always supported me and helped me become what I am today. They have always been with me throughout all the bad times and helped me overcome my fears. Thank you mom and dad for everything that you have given me.

I am more than thankful to my friend Vivek who inspite of his own busy schedule, took time to read my thesis and listen to my talk every night. Vivek, it would have been impossible to finish this thesis without you. Thanks a lot.

I am grateful to all my lab mates for creating such a lovely friendly enviroment and helping me finish my thesis. Thank you Laura for proofreading my thesis inspite of your busy schedule. Sarat thanks for your help with ViA and always being ready to answer my doubts. Amit thanks for reading my thesis. Brent, thanks for your company.

Contents

List of Figures	vi
------------------------	-----------

List of Tables	vii
-----------------------	------------

1 Introduction	1
1.1 Visualization	2
1.2 Scientific Visualization	3
1.3 Visualization and Perception	6
1.4 Visualization Assistant (ViA)	7
1.4.1 Search Process in ViA	8
1.5 Research Goals	9
1.6 Organization	10
2 Search	12
2.1 Problem Solving as Search	13
2.2 Components of a Search System	14
2.3 Formal Description	16
2.4 Search Techniques	18
2.4.1 Uninformed Search	19
2.4.2 Informed Search	21
3 ViA	24
3.1 Architecture	25
3.2 Evaluation Engines	28
3.3 Search Engine	31
3.3.1 State	31
3.3.2 Operators	32
3.3.3 Search Space	33
3.3.4 Initial State	34
3.3.5 Goal State	34
3.3.6 Heuristic Function	34
3.3.7 Control Strategies	35

4	Search Strategies	36
4.1	Hint-Based Search	37
4.1.1	Generation of Hints	38
4.1.2	Search Algorithm	41
4.2	Simulated Annealing	44
4.2.1	The Framework	45
4.2.2	The Search Algorithm	46
4.3	Reactive Tabu Search	49
4.3.1	The Framework	53
4.3.2	The Search Algorithm	54
5	Hyperbolic Visualization	60
5.1	Motivation	60
5.2	Hyperbolic Geometry	62
5.3	Implementation	65
6	Analysis	69
6.1	Search Space Visualization	70
6.1.1	Search Space Explored by Hint-Based Search	73
6.1.2	Search Space Explored by SA	73
6.1.3	Search Space Explored by RTS	74
6.2	Performance	75
6.2.1	Time Efficiency	75
6.2.2	Number of Optimal Mappings Found	84
6.2.3	Space Efficiency	85
6.2.4	Optimality	86
7	Conclusions and Future Work	87
	Bibliography	90

List of Figures

1.1	Minard’s display of Napoleon army.	3
1.2	Edward Tufte’s plots.	5
2.1	Representation of Search Space	15
2.2	Search Space Size	20
3.1	ViA’s Architecture	26
4.1	Heuristic Process in ViA	42
5.1	Difference between Euclidean and hyperbolic geometry.	63
5.2	Poincare Disk Model.	64
5.3	Distortion in Poincare Model.	65
5.4	Hyperbolic Visualization.	67
6.1	Search space visualization.	72
6.2	Search Space explored by hint-based search.	73
6.3	Search Space explored by SA.	74
6.4	Search Space explored by RTS.	74
6.5	Performance graph with initial inputs.	78
6.6	Hyperbolic Visualization with focus on point A	79
6.7	Hyperbolic Visualization with focus on point C	81
6.8	RTS spending more time in local maxima.	82

List of Tables

1.1	Tufte's work	4
2.1	Exponential growth of the search space	20
4.1	Hint Classes and Properties	39
6.1	Attribute Characteristics of the dataset	70
6.2	Initial state for run (a).	77
6.3	Initial state for run (b).	77
6.4	Initial state for run (c).	77
6.5	Initial state for run (d).	77
6.6	Initial state for Figure 6.8(a).	82
6.7	Initial state for Figure 6.8(b).	82
6.8	Hint-based search Vs RTS: Number of evaluations to find the first optimal mapping	83
6.9	Hint-based search Vs SA: Number of evaluations to find the first optimal mapping	84
6.10	Number of optimal mappings found by each algorithm	85

Chapter 1

Introduction

The exponential growth in the various sectors of computer science, such as gigabits of computational power, terabits of storage capacities and fast growing broadband networks, have resulted in tremendous data explosion. Recent results from UC Berkeley's School of Information Management and Systems state: in 2000 the estimated volume of data on the public web was 20-50 terabytes; in 2003, the volume reached 167 terabytes, three times the amount of data in 2000. Estimates suggest around 1 to 2 exabytes (10^{18} bytes) of unique data being produced each year [LPV03]. These huge amounts of data do not have any significance if we cannot extract meaning out of it. There is a need to harness this data for decision-making processes. One promising approach towards understanding and analyzing data is to visualize it. Visual tools aid users to understand the data better.

1.1 Visualization

“Now we have already discussed imagination in the treatise *On the Soul*, and we concluded there that thought is impossible without an image.” - *Aristotle*

“Logicians may reason about abstractions. But the great mass of men must have images.” - *Thomas Babington Macaulay*

To *visualize* means to form an image or a model of an abstract thought or an object. Visualization is sometimes considered as the simple process of making a graph or an image. But in fact, it is a complex cognitive process that provides a frame of reference which helps one think and understand. The purpose of visualization is to provide “insight” and thus facilitate understanding. By insight, we mean discovery, decision making or explanation. Visualization can be defined as “The use of computer-supported, interactive visual representations of data to amplify cognition” [CMS99].

The importance of visualization is exemplified in an early image by Charles Joseph Mi-nard which illustrates the famous march of Napoleon’s army in Russia in 1812 (Fig 1.1). It displays five different variables over the duration of the march: the size of the army, its location (longitude,latitude), the direction of the army’s movement, and the temperature. The size of the army is represented by the width of the band. The upper brown band shows the army’s advance from the Polish-Russian border to Moscow with an initial strength of 422,000 men. The lower black band shows its retreat back from Moscow. A mere 10,000 men returned. At the bottom of the visualization, the temperature at various points in the

army's march is shown. This clearly indicates that winter was a leading cause of the army's failure. Minard's visualization illustrates how complex multivariate data can be integrated into a graphical representation so that viewers can quickly gain access to the information without necessarily being aware that they are viewing a world of five separate dimensions.

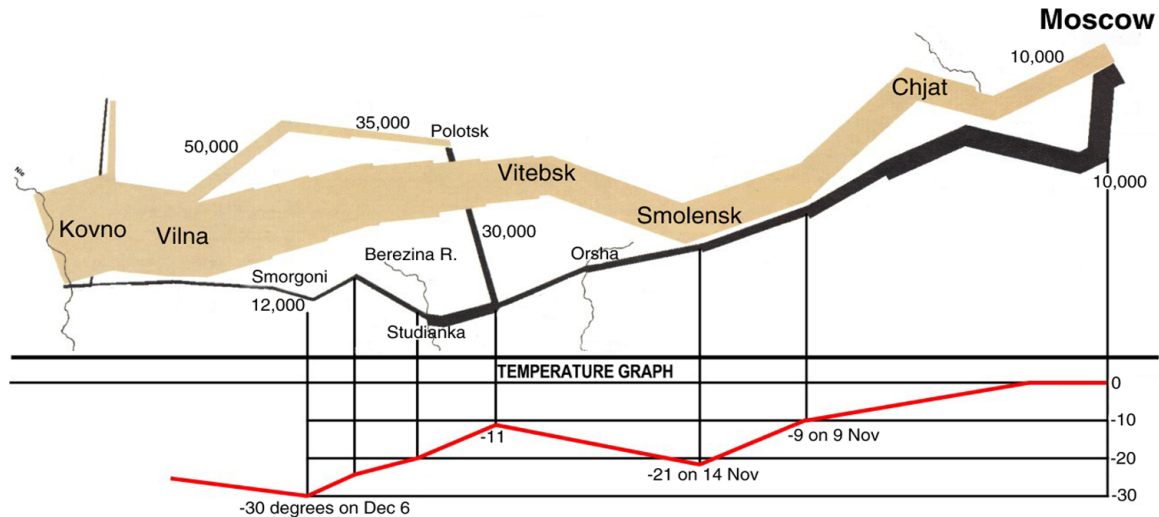


Figure 1.1: The Napoleon's March to Moscow. The war of 1812 [Tuf86].

1.2 Scientific Visualization

Visualization has become a critical component in the field of scientific computing. Scientific visualization is the conversion of collections of strings and numbers (often called datasets) into images that are used for visual exploration and analysis [HA99]. It strives to represent complex concepts and data so as to maximize the information gain. Scientific visualization allows scientists to observe, study, and explore large amounts of data in an effective manner. It provides a binding (or mapping) of data to a representation that can

be perceived [FOL94]. A simple example, from Edward Tufte’s “*The Visual Display of Quantitative Information*” [Tuf86], shows the power of graphical representation.

I		II		III		IV	
X	Y	X	Y	X	Y	X	Y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	9.75	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	8.0	9.25
12.0	9.84	12.0	9.13	12.0	8.15	18.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.73	5.0	5.73	8.0	6.89

Table 1.1: Tufte’s Work: Four similar tables

Four different datasets having the same number of elements, mean and correlation coefficients are shown in Table 1.1. In this format, they represent a collection of numbers which do not provide any further information. When the same data is represented as X-Y plots (Figure 1.2), it yields immediate insight into the relationships between the data points in each dataset. One can quickly identify outliers, trends, and correlations.

An interesting field in scientific visualization is the visualization of multidimensional data. By multidimensional, we mean a dataset containing two or more attributes. One method of visualizing multidimensional data is to use a simple object whose visual appearance like color, orientation, size, position and so on is used to represent attribute values from each data element. Consider a multidimensional dataset D containing n data elements

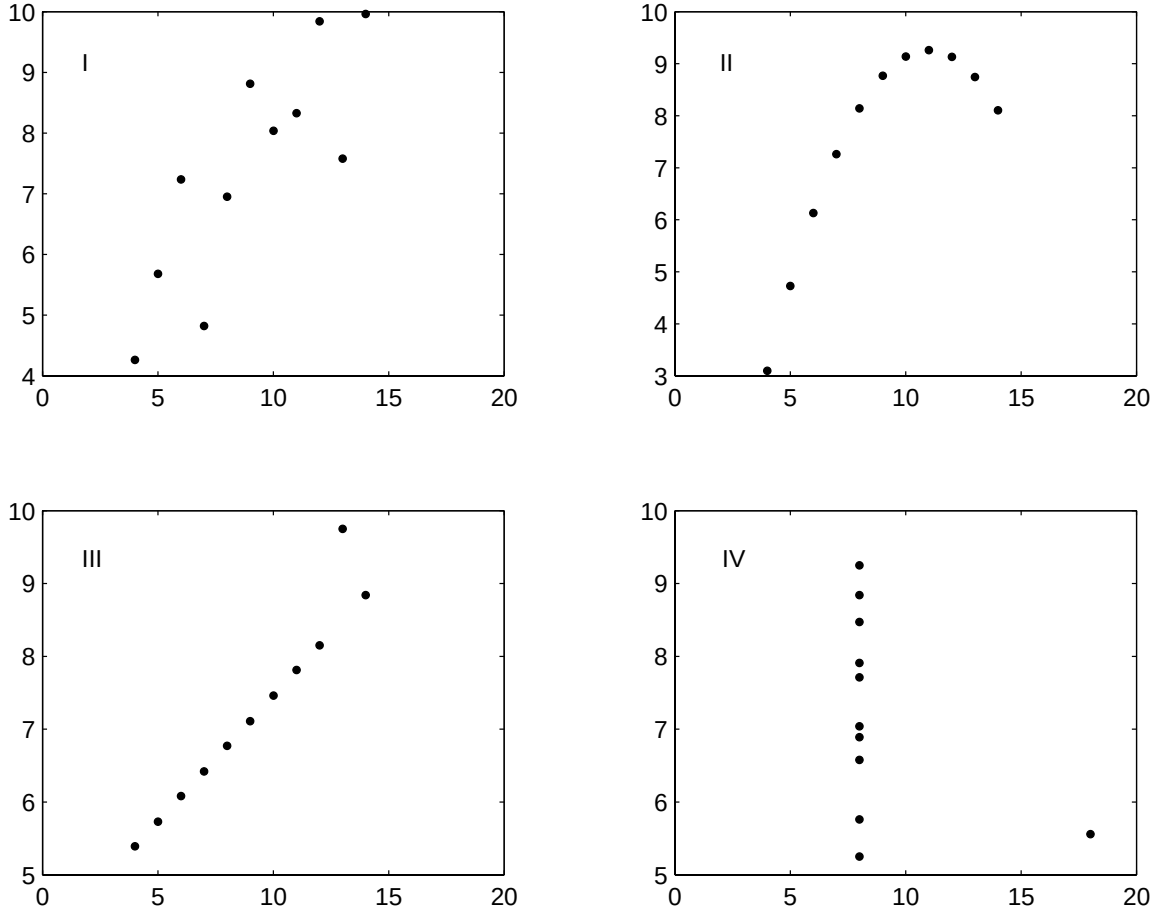


Figure 1.2: The four datasets represented as graph [Tuf86].

$e_i, D = \{e_1, \dots, e_n\}$. Each data element contains values for two or more data attributes $A = \{A_1, \dots, A_m\}, m > 1$ stored in D , that is, $e_i = a_{i,1}, \dots, a_{i,n}, a_{i,j} \in A_j$. A mapping M is defined as $M(V, \phi)$ where $V = \{V_1, \dots, V_m\}$ identifies a visual feature V_j to use to represent each attribute A_j . $\phi_j : A_j \rightarrow V_j$ is a function that maps the domain of A_j onto the range of displayable values in V_j . The generation of multidimensional visualization can be characterized as:

1. The design of a data-feature mapping M , a function that defines visual features (such as color, texture, or motion) to represent the data.

2. An analysis of a viewers interpretation of the images M produces.

An effective M generates visualizations that let viewers rapidly, accurately, and effortlessly explore their data [HE02].

1.3 Visualization and Perception

The effectiveness of a visualization depends on many different conditions like conventions, subjective user preferences, cultural settings and the structural correspondence with the data. For example, the effectiveness of a temperature map in which temperature values are visualized by color hue values may increase if conventional color hue values are used for high and low temperatures, i.e. red for high temperatures and blue for low temperatures. The process of visualization can be considered as the inverse of the interpretation process, i.e. the visualization process generates for an input data a visual representation, the interpretation of which results in the understanding of the original data. The effectiveness of a visualization can be measured in terms of the ease and directness of acquiring its intended interpretation. Since the interpretation depends on human visual perception, an effective visualization could strongly benefit from the capabilities of the human visual system [Das01].

Experiments in low level human vision suggest that certain basic visual features can be identified without the need for focused attention. They are detected rapidly and independent of the total amount of information being displayed [Hea92]. These features include

common properties like hue, luminance, orientation, size and direction. Unfortunately, an ad hoc mapping will not automatically take advantage of these perceptual strengths. In fact, certain combinations of visual features actively interfere with our ability to comprehend and may even mislead us. Studies on the effective use of these features and their combinations have led to the development of perceptual guidelines. Use of these guidelines can increase the amount of information that we can visualize and can produce visualizations that cater to the human visual system [HE02]. Unfortunately, the application of the guidelines to create effective mappings and the evaluation of the effectiveness of a mapping requires users to be both domain as well as visualization experts. In order to ease this burden, researchers in our lab have developed a semi-automated visualization assistant, ViA.

1.4 Visualization Assistant (ViA)

The function of ViA can be divided into two main parts: evaluation engines that use the perceptual guidelines to determine the applicability of a mapping to a particular dataset and a collection of analysis needs and a search engine to intelligently search the space of all possible mappings for those specific mappings best suited to the current dataset and tasks. The evaluation engines, based on the perceptual guidelines, encode rules for each visual feature that ViA understands. The search engine utilizes a general purpose, domain independent search that uses guidelines from the evaluation engines to find optimal mappings.

The user provides information regarding the relative importance and spatial frequency of the attributes in a dataset, together with any analysis tasks to be performed on each.

The search process begins with an initial mapping and then moves to new mappings based on the feedback from the evaluation engines. This process is repeated until one or more perceptually salient mappings are obtained.

1.4.1 Search Process in ViA

ViA's search engine tries to find optimal mappings between the attributes and visual features within the constraints set by the perceptual guidelines. An "optimal mapping" is one which has the highest evaluation weight as provided by the evaluation engine. Thus the search problem in ViA can be viewed as a global optimization problem. One way to solve the global optimization problem is to search all possible solutions and choose the best result. Unfortunately the space of possible solutions in the case of ViA is very large, making brute force search infeasible. One technique that can be used to solve the global optimization problem is to use heuristic search. Heuristics are techniques that improve search efficiency by trading completeness for speed. ViA uses a heuristic search algorithm (called hint-based search in the rest of the thesis). The heuristic search is based on the effectiveness of a mapping as measured by the evaluation engines with a numeric value called evaluation weight. Along with the evaluation weight, evaluation engines also generate suggestions that are used by ViA's search algorithm to try to improve a mapping. In this way ViA moves through the space of possible mappings by constraining its path to mappings with a potential for improvement.

1.5 Research Goals

The problem of finding the best mappings between the visual features and the attributes can be solved using any of a number of heuristic search techniques. This thesis studies the efficiency of the current heuristic search algorithm in ViA and compares it with two other heuristic techniques simulated annealing and reactive tabu search. We aim to achieve the following goals:

1. Study the interaction of the search engine with the dataset and the perceptual guidelines.
2. Incorporate simulated annealing and reactive tabu search to try to generate optimal mappings.
3. Compare the performance of hint-based search, simulated annealing and reactive tabu search.
4. Analyze the behavior of the three search algorithms by developing a system to visualize the huge search space and trace the areas explored by each of them.

The search engine interacts with the evaluation engines to obtain an evaluation of a mapping. The evaluation engines provide an effectiveness metric, independent of ViA's search technique. Thus changes to the search technique do not affect the other components in ViA. Simulated annealing and reactive tabu search are heuristic strategies which use different methods to escape out of local maxima. This thesis evaluates how each of the search

algorithms works within ViA and compares them with the current hint-based search algorithm. Time, search space explored, optimality are used to compare the search strategies.

One way to study the behavior of an algorithm is to visualize it. This requires a technique that allows us to visualize the entire search space, as well as to concentrate on interesting subareas in this space. Visualizing the search space in normal Euclidean space is difficult, particularly since the search space can grow to contain thousands of states. Instead, we chose to visualize in hyperbolic space, where the area increases exponentially and can hence accommodate a search space with many nodes. The infinite hyperbolic space can be mapped to a finite area of Euclidean space using models which will be described later. We exploit the advantages of hyperbolic space to explore the search space.

1.6 Organization

In this thesis, we aim to visualize and compare the performance and behavior of hint-based search with two other generic search strategies: simulated annealing and reactive tabu search. The thesis is organized as follows. Chapter 2 formally defines the search problem and describes the components of a search system. It also outlines the approaches to solve the problem. Chapter 3 describes the architecture of ViA and how evaluation engines integrate the perceptual guidelines into the search process. Chapter 4 describes the hyperbolic visualization system, explains the different search strategies, namely, hint-based search, simulated annealing and reactive tabu search in detail. Chapter 5 describes hyperbolic geometry and its models and how it is used to study the behavior of a search

strategy. Chapter 6 compares the three search strategies with respect to optimality, time and space efficiency. Chapter 7 provides some conclusions drawn from the comparison.

Chapter 2

Search

The Webster's dictionary defines *search* as the examination of a set of items to identify one or more items having a given property. Many problems in computer science can be described as the *search* for an item, from among a set of items, that satisfy a certain criteria. Search refers to a general class of algorithms that explore a problem space consisting of the set of all possible solutions, in order to find a solution. According to the researchers at CIRL (The Computational Intelligence Research Laboratory), virtually every problem in computer science can be solved using a search algorithm. A simple approach to search can be a brute force algorithm that examines each and every solution within the problem space to find the best. However, such an exhaustive search may not always be feasible, especially when the number of solutions is very large. In such scenarios, a heuristic-based approach for search can be used to improve the efficiency of the search process.

In this chapter, we will look at how a problem solving task can be reduced to a search problem. We will also study the basic terminology and the main components of a search problem. We will then investigate how heuristics can be used to improve the efficiency of the search system.

2.1 Problem Solving as Search

Search plays a very important role in computer science. It forms the conceptual backbone of every approach to the systematic exploration of alternatives. Any problem solving task can be viewed as a search for the “correct desired solution.” In artificial intelligence, search forms the core concept in ideas that deal with deduction, inference, planning, commonsense reasoning, theorem proving and so on. These ideas form the basis of information retrieval, automated agents, robotics and natural language understanding [Gar80]. Thus, search can be viewed as a tool for problem solving in a wide variety of problem domains.

A common AI-based approach to problem solving involves finding a sequence of actions that can achieve the goal configuration, starting from the initial state. One way to do this, is to explore the set of possible actions that are applicable to an initial state. This approach is then repeated for each of the states encountered until the goal state is reached. Each configuration can be considered as a node and the actions as edges, thus leading to a graph representation. Thus, problem solving can be considered as a search through a graph of nodes to find the goal node. The next section describes how a problem can be formally represented as a search problem.

2.2 Components of a Search System

Reducing a given problem to a search problem requires the definition of the following components:

1. *State*: A distinguishable stage which describes a possible task-domain configuration in the problem solving process.
2. *Operators*: Actions required to move from one state to another. We assume the operators are *deterministic* and *discrete* i.e., we know exactly which state will be reached after the operator is applied and we do not represent what happens while the action is being applied.
3. *Search Space*: The set of all possible states of a problem is called the search space. It defines the environment in which the search takes place. A search space consists of a set of states, where each state represents a configuration reachable from the start state and a set of operators that help us traverse the search space. As shown in Figure 2.1, a search space can be represented as a inverted tree where the root represents the initial node and each level represents states generated by applying operators to the states in the previous level. Its structure corresponds to the structure of problem solving in two important ways:
 - It provides a formal definition of a problem as the need to convert an initial state to a desired goal state using a set of permissible operations.
 - It defines the process of problem solving as a combination of knowledge (repre-

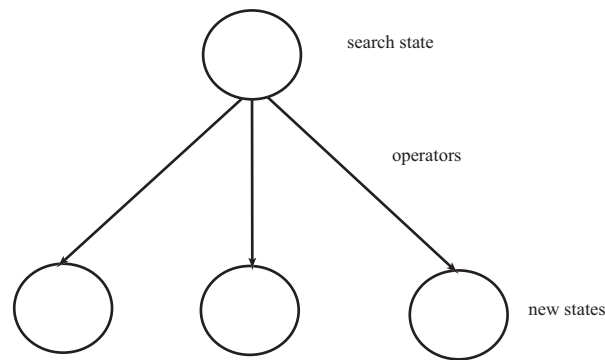


Figure 2.1: Search Space as a set of states and operators.

sented as rules defining the permissible moves in the space) and search [RK90].

4. *Initial State*: The initial state represents the starting configuration of the search process. To achieve the goal state, some search algorithms require that the initial state satisfy certain criteria. Other search algorithms autonomously find their way to the goal state regardless of the initial state. Hence, the initial state might or might not be crucial in achieving the goal state.
5. *Goal State or Search Criteria*: The goal state can be a single state or a set of states. They vary widely in different problem domains in terms of quantity and quality. For example, it may be a single desired state or an exhaustive enumeration of all the states in the space; or, it may be a desired exact match such as a global optimum or an approximate search for something like a local optimum.
6. *Control Strategy*: The control strategy determines which operators should be applied next. The choice of the control strategy has a crucial impact on how quickly, and even whether the goal state is achieved. The properties of a good control strategy

are:

- A good control strategy must cause motion: This ensures that the search does not get locked and never lead to a solution
- A good control strategy must be systematic: A non-systematic strategy may explore a useless sequence of operators repeatedly before the solution is achieved. The requirement that the control strategy be systematic corresponds to the need for global motion (over a course of several steps) as well as for local motion (over the course of single step).

Different control strategies, when applied to an identical search space with the same initial and goal state, might generate different paths from the initial state to the goal state. In fact, the ability to reach the goal state may be determined by the control strategy. For example, greedy or hill-climbing techniques can cycle over states that are already visited. This is because each step is taken with the sole criterion of immediate gain regardless of past history or future consequences. Tabu search, on the other hand, will not exhibit this behavior because it remembers and avoids visiting nodes encountered in recent history.

2.3 Formal Description

Putting these definitions together, the process of providing a formal description of a problem consists of the following steps:

1. Define a search space containing all possible configurations of the relevant variables in the problem domain. The definitions can be provided without explicitly enumerating all of the states contained within the search space.
2. Specify a set of rules that describe the actions (operators) available.
3. Specify the initial and goal states.
4. Design a good control strategy appropriate for the problem domain.

The problem can then be solved using the rules, along with the control strategy, to move through the problem space until a path from an initial state to a goal state is found. With this formal definition of a problem, we can now define a search algorithm as a framework for specifying a class of possible traversals of the search space. Figure 2.3.1 provides a framework for a generic search algorithm.

Algorithm 2.3.1: GENERAL SEARCH ALGORITHM(*problem, strategy*)

comment: Initialization

$state \leftarrow \text{Initial state}$

$g \leftarrow \text{formulate_goal}()$

$problem \leftarrow \text{formulate_problem}(problem, state, g)$

comment: The search process

while $state \notin \text{goal state}$ **and** *not all states explored*

do $\begin{cases} action = \text{choose_next_move}(state, problem, strategy) \\ state = \text{perform_action}(action, state, problem) \end{cases}$

if $state \in \text{goal state}$

then return (*success*)

else return (*failure*)

With this search framework, we can customize the search algorithm for different problem domains using different control strategies. In the following sections, we will explore some of these strategies.

2.4 Search Techniques

Search techniques can be broadly divided into two main categories: *blind* or *uninformed* search and *informed* or *heuristic* search.

2.4.1 Uninformed Search

Uninformed search strategies do not exploit any apriori knowledge available from a state. These strategies do not have any preferences in choosing the next move. Uninformed/blind search can be considered as the systematic generation and testing of search space elements. Although they can provide complete and optimal solutions, they are highly inefficient. All uninformed search strategies have worst case exponential time complexity. Examples of these include breadth first and depth first search.

2.4.1.1 Combinatorial Explosion

A simple motion causing and systematic control strategy would be to explore all possible states in a search space until the goal state is achieved. This approach is effective when the size of the search space is small. The time needed by this algorithm is directly proportional to the number of states in the search space.

In order to estimate the size of a search space, consider a problem where only two operators can be applied to each state. Figure 2.2 shows the first four levels of a search space for this problem.

The number of states at any level, n , is 2^n and the total number of states in a search space with n levels is $2^{(n+1)} - 1$. Thus, the number of states is exponential in terms of the number of levels. Table 2.1 provides an estimate of the exponential growth of the search space.

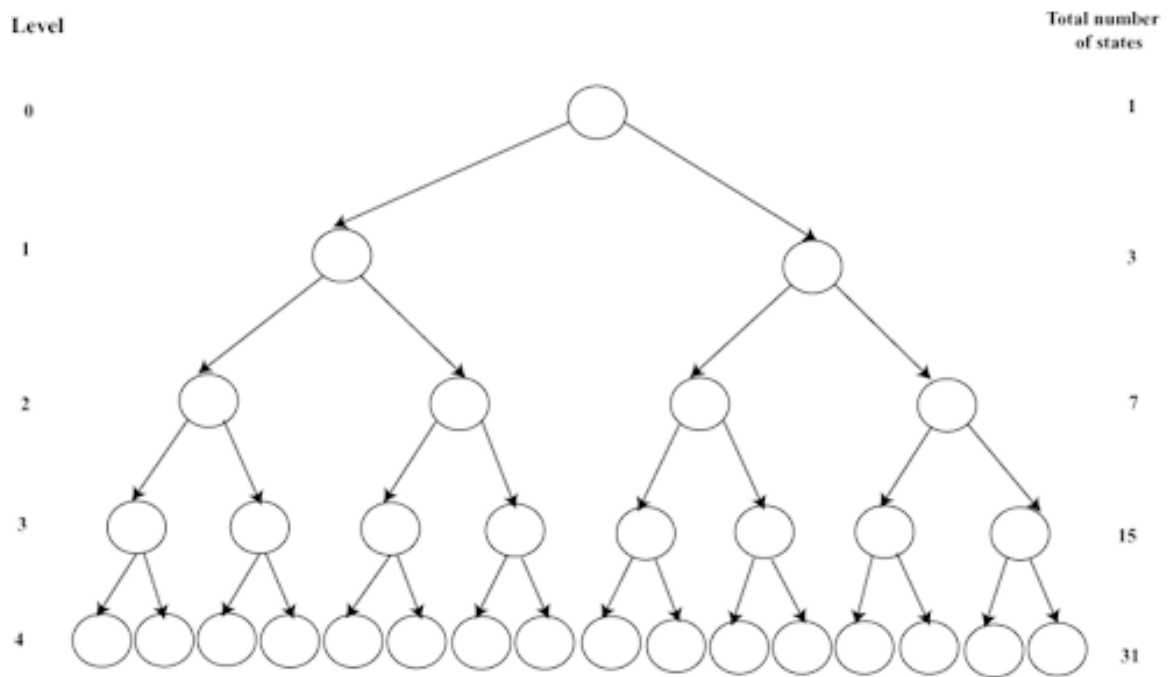


Figure 2.2: Growth in the search space

n	2^n	$2^{n+1} - 1$
2	4	7
4	16	31
6	64	128
10	1024	2047
15	32768	65536
20	1048576	2097151
30	1073741824	2147483647

Table 2.1: Search Space: Exponential growth

Since in general, the number of operators is greater than two, the resulting search space is extremely large. For instance, in chess, a player has an average of 30 moves that he/she can make. If we look ahead by 10 moves, there would be a total of 30^{10} states that would be generated. Note, however, that the above calculations do not take into consideration that when operators are applied some nodes are identical to ones already existing. In some situations, this might have a great effect on the size of the search space.

2.4.2 Informed Search

As mentioned earlier, an increase in the number of operators leads to a combinatorial explosion in the size of the search space. The exponential time necessary for exhaustive search makes it infeasible for most nontrivial problems. In order to solve such intractable problems, it is often necessary to compromise a systematic approach for efficiency and to design a control strategy which is no longer guaranteed to find the best answer but that will almost always find a good solution.

Informed or heuristic search strategies aim to improve the efficiency of search by making use of domain knowledge to make intelligent decisions. These strategies use domain information intelligently within the formalism of blind search to restrict allowable search directions [Nil71].

The word “heuristic” is derived from the Greek verb *heuriskien* which means *to find* or *to discover*. George Polya in his book “*How to Solve it*” describes heuristics as the study

of methods for discovering and inventing problem solving techniques. A heuristic is not guaranteed to always solve the problem, but often solves it well enough most of the time and more quickly than a complete solution would.

The basic idea of heuristic search is to focus the search towards more profitable paths, rather than examining all possible paths. In order to recognize the profitable paths, heuristic search techniques make use of domain specific knowledge which can provide information about the merits of a particular state. There are two ways in which domain knowledge can be incorporated within the rule-based search procedures:

- Within the rules/operators. For instance, instead of specifying only the set of next moves, also provide information about the set of next profitable moves.
- As a heuristic function that can evaluate the merits of a given state or move.

A heuristic function or an evaluation function provides a mapping from the problem state descriptions to measures of desirability. It gives a score to a state, which defines its proximity to the goal state. Heuristic functions provide an evaluation criterion to be used when choosing the next move from the set of available moves. They aim to guide the search process in the most profitable direction. Some heuristic functions might look at immediate gains and always choose the best next move whereas others might look at long term gains and choose a move which later becomes more profitable. Heuristic functions can be of several different types and new functions can also be invented. There are a number of generic heuristics, for example, simulated annealing, tabu search and genetic algorithms.

By generic, we mean they provide a general framework to build an algorithm. These algorithms can be tailored to the specific requirements of the problem by defining the problem specific operators and the evaluation functions.

In later chapters, we will describe how the generic algorithms simulated annealing and reactive tabu search were tailored for optimization in ViA, and compare them to a separate heuristic developed specifically for ViA to assess their performance.

Chapter 3

ViA

In the previous chapter, we saw how a problem solving task can be converted into a search problem. In this chapter, we will study how the task of finding optimal data attribute to visual feature mappings in visualization can be reduced to a search problem.

Multidimensional datasets consists of a set of m sample points or data elements where each element encodes a set of n attributes values, one for each attribute $A = (A_1 \dots A_n)$ represented within the dataset. Given a set of visual features $V = (V_1 \dots V_n)$, which can be used to visualize these attributes, a mapping is an ordered pair $M = (A_i, V_i)$. There are $n!$ mappings possible between attributes and visual features. However, not all of these mappings lead to equally effective visualizations. An effective visualization is one that produces images that a viewer can use to rapidly, accurately, and effortlessly perform their analysis tasks.

Research on the low level human visual system has led to a set of perceptual guidelines that allow objective evaluation of the effectiveness of a mapping. Also, new visualization techniques such as non-photorealistic visualization and perceptual flow patterns have produced new ways to visualize multidimensional data. Progress in these two areas combined with the increasing need to extract information from large multidimensional datasets suggests the desire for an automated visualization system that can take data and user preferences and generate effective data-feature mappings.

In our laboratory, we have developed such a system, called the visualization assistant, or ViA. ViA combines perceptual guidelines and an intelligent search engine to identify the data-feature mappings that are most appropriate for a particular dataset and the associated analysis tasks.

3.1 Architecture

The input to ViA consists of a multidimensional dataset, user preferences and the analysis tasks that the user wants to perform. ViA's output is a set of data-feature mappings that according to ViA will generate effective visualizations. ViA aims to achieve the following goals.

- Effective visualizations: Generate mappings that allow users to rapidly, accurately and effectively conduct their analysis
- Domain independence: ViA should not be constrained to one specific application or

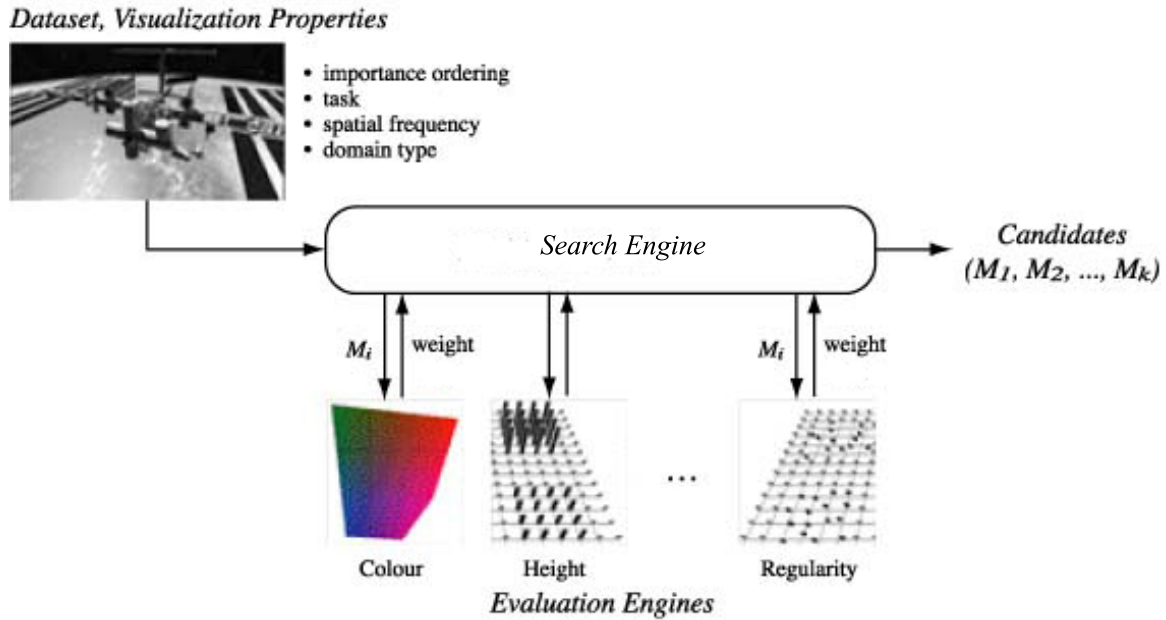


Figure 3.1: Architecture of ViA

environment

- Context dependent visualizations: ViA should generate visualizations that take user preferences and analysis tasks into consideration
- Efficiency: ViA should be intelligent when searching for effective data- feature mappings

ViA's architecture is based on conventional principles of building knowledge based systems. Figure 3.1 shows the architecture. ViA is composed of two main components: the search engine and the evaluation engines. The evaluation engines encode visualization rules, which are based on the abilities of low level human visual system. They provide the heuristic function used by the search algorithm to evaluate the desirability of a given mapping. The search engine consists of a heuristic algorithm that utilizes the information

from the evaluation engines to determine the direction of search.

ViA's operation can be explained in three main steps:

1. Determine the properties of the dataset and the user's preferences.

ViA begins by asking the user domain independent questions about the dataset and the analysis tasks that need to be performed. The information provided by the user includes:

- Importance ordering: This indicates the relative importance of an attribute for the particular analysis task the user wants to perform. It is specified as a value between 0 and 1. For instance, the user might be more interested in temperature differences in different geographic locations than pressure differences when analyzing a weather data set. This can be indicated by the user by giving a higher numeric importance value to temperature.
- Domain type: Users specify whether each attribute is continuous or discrete.
- Analysis task: Users can also optionally specify a task to be performed on each attribute (search, boundary detection etc.). For example, the user might want to search for all geographic locations with low precipitation in a weather data set.
- Spatial frequency: ViA determines the spatial frequency of each attribute. The user can override ViA's decision.

Once the properties of the dataset are determined, ViA begins the search process.

2. Determine the best mapping using a heuristic search process.

The search process uses a heuristic function which is defined by the set of evaluation engines. These engines encode the perceptual guidelines and assign a numeric value to a mapping indicating its desirability. Section 3.2 describes the evaluation engines in greater detail. The start state used by the search process is a mapping based on user preferences and the input to ViA. The goal state is the highest value mapping as determined by the heuristic function. During its operation, the search process maintains a set of the best k mappings and returns this set on termination. Section 3.3 describes the search process in greater detail.

3. Display the best mapping to the user.

Different visualization techniques can be used to visualize the data based on one of the best mappings as selected by the user.

Based on the results of their analysis, users can tune various dataset properties such as the importance of attributes to direct the search towards new and possibly more desirable directions.

3.2 Evaluation Engines

Evaluation engines judge the desirability of a data-feature mapping. A data-feature mapping is a relation between attributes of the dataset and visual features. There is one evaluation engine for each visual features supported by ViA. Currently we have evaluation engines for color, luminance, density, height, orientation and regularity. Evaluation en-

gines provide an estimate on whether the use of a visual feature is appropriate for the data attribute it is currently mapped to.

The evaluation engines use rules derived from studies of human vision. One interesting discovery in this area are pre-attentive features. Pre-attentive features allow rapid and accurate detection of various image properties, independent of the amount of information being displayed. Pre-attentive features include the color and texture feature, ViA currently employs.

Research on the use of color suggests that, when selecting a color to represent an attribute, three criteria need to be considered: color distance in a perceptually balanced color model, the ability to linearly separate targets from non-targets in the color model used and the named regions each color occupied. Experiments show that with the correct use of these criteria, up to seven different colors can be distinguished rapidly and accurately when displayed simultaneously [Hea96].

Research on the use of the three texture dimensions, height, regularity and density, show that texture can also be used to rapidly and accurately visualize multidimensional data [HE98]. Experimental results showed that taller, shorter, denser and sparser targets can be easily identified, but that certain combinations of texture values can produce visual interference. Also, experiments on combination of color and texture, showed that random variation in color can mask underlying texture patterns, but that variations in texture has no effect on a viewer's ability to identify individual colors [HE99].

These results are integrated into the evaluation engines in the form of tests which are applied to a data-feature mapping. Given a data-feature mapping M , each of the evaluation engines perform the following tests to determine how well its visual feature (V_i) can represent the attribute (A_i) in M :

1. Interference: Does any other attribute-feature mapping $A_j \rightarrow V_j$ in the mapping M visually interfere with $A_i \rightarrow V_i$? Interference occurs when some less important attribute A_j uses a more perceptually salient visual feature V_j .
2. Domain type: Does V_i support the domain type of A_i ? Certain features can only support discrete attributes whereas others can support both discrete and continuous attributes.
3. Spatial frequency: Can V_i represent the spatial frequency of A_i ? For example, color is best suited to low frequency data, while luminance is best suited to high frequency data.
4. Task applicability: Can V_i support the analysis tasks the user wants to perform on A_i ?

Based on these questions, each engine gives its part of the mapping a value between 0 and 1. A value of 1 indicates that none of the questions produced a negative answer. The evaluation values returned by each of the engines is averaged to obtain an evaluation weight for the entire mapping. The evaluation function can thus be represented as,

$$E = \frac{\sum_{i=1}^n Evaluation\ Weight(i)}{n} \quad (3.1)$$

where n is the number of visual features used to represent the data attributes.

3.3 Search Engine

Section 2.2 described the various components of a search system. The search engine views the problem of determining an effective data-feature mapping within this framework. Thus the search for an effective mapping can be formalized by assigning the characteristics of ViA's problem domain to the components of the search system.

3.3.1 State

We define a state in ViA as consisting of a data-feature mapping together with the properties of each attribute. Thus the state consists of ordered pairs (A_i, V_i) and the following properties of each attribute:

1. Importance weight
2. Number of discrete values
3. Domain type
4. Task

We chose to include the properties of the attributes as part of the state as it allows ViA to modify them in order to relax the constraints on the search. Without these properties as part of the state, the only possible action would be to swap features to represent different attributes. This can then cause ViA to generate only very low weight mappings. Inclusion

of properties in the state allows ViA to change them in order to suggest mappings with higher evaluation weights whenever such a relaxation is allowed.

3.3.2 Operators

Operators provide the means to make a move in the search space. Four kinds of operators are defined. They modify one of the five attributes of the state to obtain a new state. The operators are feature swap, discretize, importance weight modification and task removal.

3.3.2.1 Feature Swap

Given a state s containing mapping M with data feature pairs (A_1, V_1) and (A_2, V_2) application of $feature_swap(V_1, V_2)$ generates a new state s' with mapping M' such that (A_1, V_2) and (A_2, V_1) . If the new mapping better respects the importance ordering of the attributes and the visual salience of the features, then the resultant state will have a better evaluation weight.

3.3.2.2 Discretize

This operator changes the number of discrete values associated with an attribute. Research suggests that each visual feature can effectively support only a limited number of discrete values. For instance, color can support seven discrete values without overwhelming the user. This domain knowledge is utilized to decide the number of ranges of values that an attribute is discretized to. For example, consider a state s that contains an attribute $A_i \rightarrow color$ with 10 discrete values. When the $discretize(7)$ operator is applied to s , it

generates a new state s' , with $A_i \rightarrow color$ and having seven discrete values.

3.3.2.3 Importance Weight Modify

The user initially specifies the importance ordering of the attributes. Given a state s , application of the *importance_weight_modify*(A_i, A_j) leads to a new state s' with the importance weight of the two attributes reordered. This operator helps to resolve any visual interference resulting from a difference in the importance ordering of the attributes and the perceptual salience of the features.

3.3.2.4 Task Remove

ViA currently supports four exploratory tasks: search, estimate, boundary detection and tracking. Each visual feature can support some of these tasks better than others. The task remove operator helps disassociate a task from an attribute. Thus given a state s , *task_remove*(A_i) creates a new state s' with the attribute A_i no longer having the given task.

3.3.3 Search Space

As described earlier, a state consists of a data-feature mapping and the properties associated with each data attribute. Modification to the mapping or any of the properties leads to a new state. Hence, any data-feature mapping applied to the initial attribute properties is part of the search space. Given a set of m visual features and n attributes, $m \geq n$, ${}^mP_n = \frac{m!}{(m-n)!}$ data feature mappings can be generated. Moreover, any modification of the initial attribute

properties leads to a new set of $\frac{m!}{(m-n)!}$ potential mappings that can themselves be ordered and modified. This can lead to an extremely large search space making exhaustive search infeasible. Hence, ViA adopts heuristic search techniques to try to find the best mappings.

3.3.4 Initial State

The initial mapping is created by assigning visual features ordered by perceptual salience to attributes, ordered by relative importance as specified by the user. The properties of the attributes are the initial properties as specified by the user.

3.3.5 Goal State

A mapping that satisfies the user's constraints on importance ordering, task, frequency and domain and that has the highest evaluation weight is defined as the goal state. ViA tries to reach the goal state using heuristic search (or a good approximation of the goal state).

3.3.6 Heuristic Function

The heuristic function provides an evaluation weight that represents a measure of desirability of a mapping. The evaluation function described in the previous section is used as the heuristic function in ViA.

3.3.7 Control Strategies

Three different control strategies were tested within ViA: the existing hint-based search, simulated annealing and reactive tabu search.

Starting with the initial state, ViA uses operators to move within the search space until the goal state is reached. The choice of operator at each step is governed by the control strategy. Different control strategies use different criteria to select the operators to apply at each step. Chapter 4 describes each of these strategies in detail.

Chapter 4

Search Strategies

In the previous chapter, we described the overall architecture of ViA and discussed the evaluation engines and search engine in detail. In this chapter, we will study three control strategies: ViA’s current hint-based search, simulated annealing and reactive tabu search. Each of these methods differ in the way that they choose the next operator to be applied and in the mechanism used to escape local maximum. Hint-based search uses perceptual guidelines to choose the operators most likely to reach better nodes, and applies combination of these operators to avoid local entrapment. Simulated annealing chooses a random operator at each step and adopts a probabilistic model to avoid being confined in a local maximum. Reactive tabu search selects the best available operator at each step and avoids revisiting recently encountered nodes. From this point on, we use the term “mapping” to refer to a state in a search space. The state also contains attribute properties in addition to a data-feature mapping. We explicitly refer to properties when needed.

4.1 Hint-Based Search

Hint-based search is a heuristic control strategy which combines domain knowledge and guidelines on the use of human perception to direct the search in ViA. The previous chapter described the various operators available in ViA. Hint-based search chooses the operator or a set of operators to be applied so as to focus the search towards perceptually better mappings.

Evaluation engines in ViA provide the evaluation weight of a mapping. We know that if a mapping evaluates to a lower weight, it violates perceptual guidelines in at least one of the evaluation engines. In order to improve the mapping, we need to fix these violations. Since the evaluation engines know which perceptual rules are violated, they were designed to serve two related purposes:

1. Measure the effectiveness of a data attribute-visual feature pair within a data-feature mapping, and
2. Provide suggestions to try to improve the data attribute-visual feature pair, and therefore the mapping as a whole.

These suggestions are called hints. Hints are defined as promising operators that could be used to move the search closer to a goal state. Multiple hints can be generated by each evaluation engine.

4.1.1 Generation of Hints

Four kinds of operators are available to make the next move: discretize, feature swap, importance weight modification and task removal. No visual feature can support every type of an attribute. Table 4.1 shows how well the different attribute characteristics are supported by each visual feature. When a visual feature cannot support the characteristics of the attribute it is currently mapped to, the evaluation engines provide a lower evaluation weight and also suggest operators (i.e. hints) that might fix the violation. The following sections give a brief explanation of why each hint might be suggested.

4.1.1.1 Feature Swap

If an evaluation engine's visual feature cannot support certain properties of the given attribute, for example spatial frequency, or task or domain type, it can suggest swapping the attribute to a more appropriate feature. As an example, if a visual feature V_i cannot support the number of discrete values of attribute A_i without loss of information, the evaluation engine E_i might suggest swapping to another visual feature V_j which can better support the unique values of A_i . If no attribute is using V_j , the data-feature pair in the mapping is updated from (A_i, V_i) to (A_i, V_j) . If some other attribute A_j is using V_j , it swaps with A_i , that is, the data-feature pairs $(A_i, V_i), (A_j, V_j)$ become $(A_i, V_j), (A_j, V_i)$ within the mapping. Other problems within a mapping can generate a *feature_swap* hint, for example, if A_i is a high importance attribute but V_i is a low salience visual feature, or if V_i cannot support the tasks the user wants to perform on A_i .

	Domain Type	Visual Interference	Spatial Frequency	Task Applicability
Luminance	<i>continuous: ok</i> <i>discrete: up to 3</i>		<i>hi: ok</i> <i>lo: poor</i>	<i>search: discrete</i> <i>estimate: discrete</i> <i>boundary: ok</i> <i>track: ok</i>
Color	<i>continuous: ok</i> <i>discrete: up to 7</i>	w/ luminance	<i>hi: poor</i> <i>lo: ok</i>	<i>search: discrete</i> <i>estimate: discrete</i> <i>boundary: ok</i> <i>track: ok</i>
Height	<i>continuous: ok</i> <i>discrete: up to 5</i>	w/ luminance w/ color	<i>hi: ok</i> <i>lo: ok</i>	<i>search: discrete</i> <i>estimate: discrete</i> <i>boundary: ok</i> <i>track: ok</i>
Density	<i>continuous: no</i> <i>discrete: up to 4</i>	w/ luminance w/ color w/ height	<i>hi: poor</i> <i>lo: ok</i>	<i>search: discrete</i> <i>estimate: discrete</i> <i>boundary: ok</i> <i>track: ok</i>
Regularity	<i>continuous: ok</i> <i>discrete: up to 2</i>	w/ luminance w/ color w/ height w/ density	<i>hi: poor</i> <i>lo: ok</i>	<i>search: no</i> <i>estimate: no</i> <i>boundary: ok</i> <i>track: discrete</i>
Orientation	<i>continuous: ok</i> <i>discrete: up to 12</i>	w/ luminance w/ color	<i>hi: ok</i> <i>lo: ok</i>	<i>search: discrete</i> <i>estimate: discrete</i> <i>boundary: ok</i> <i>track: ok</i>

Table 4.1: Hint tests performed on visual features, showing the attribute characteristics supported by the different features (luminance, color, height, density, regularity, and orientation): “ok” for a good match, “poor” for a weak match, and “no” for an unallowed match. [Koc02]

4.1.1.2 Discretization

As shown in Table 4.1, some visual features do not support continuous values. For instance, density does not represent continuous attributes well. Also, each visual feature supports a different number of easily distinguishable discrete values. Each evaluation engine checks to see if its visual feature can represent the domain of the given attribute. If not, it suggests a discretization to range the attribute into a more appropriate set of values.

4.1.1.3 Importance Weight Modification

When a low-importance attribute is mapped to a highly salient visual feature, visual interference can occur. This occurs when the low importance attribute is displayed in a way that interferes with a viewer's ability to see the values of more important attributes mapped to less salient features. A feature swap can be used to correct this problem. If two attributes have similar importance weights, an importance weight modification can also be used to correct this problem. In this case, the importance weight of the stronger attribute is lowered and the importance weight of the weaker attribute is raised to reverse their relative importance. This removes the visual interference, since the higher importance attribute is now using the more salient visual feature, exactly as it should.

4.1.1.4 Task Removal

The user can specify data-exploration tasks to be performed on each attribute. Tasks include: search elements with a target attribute value, estimate the number of elements with a target value, identify spatial boundaries between groups of elements with similar attribute

values, and track elements as they move in time and space. The ability to support a task for a given attribute A_i depends on the visual feature V_i that it maps to and the domain type of A_i . For instance, visual features colour, luminance, height and density can support all four tasks for discrete attributes. However, when mapped to continuous attributes, they can only support boundary detection and tracking. If the data exploration task to be performed is not supported by V_i and if A_i 's importance to the viewer is low, the evaluation engines may recommend to remove the unsupported task in the form of a task remove hint.

4.1.2 Search Algorithm

Hints are generated by each evaluation engine independently. Hints suggest modifications to an attribute-feature pair that when applied will generate a new mapping. Each hint includes an estimate of how much the mappings evaluation weight will improve if the hint is applied. Application of individual hints will direct the search to potentially better mappings in the immediate neighborhood of the current mapping. A search algorithm can now be designed to repeatedly choose the hint with the best improvement weight and apply it at each step until the termination condition is reached. The main drawback of this approach is that it may take a long time to reach the optimal mapping. Also, there is a possibility of becoming trapped in local maximum, since the search is confined to the local neighborhood.

The approach adopted by hint-based search to overcome this problem, is to combine the hints into *hint chains*. Hint chains allow multi-step moves to be performed by applying

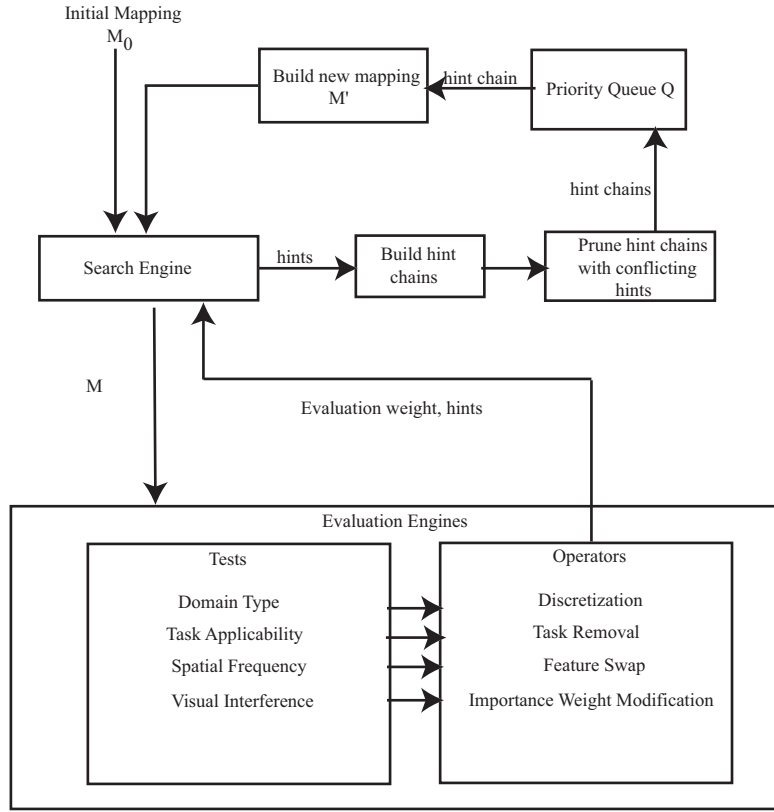


Figure 4.1: The heuristic process used in ViA.

a combination of operators at each step in the search process. This can support rapid convergence to perceptually salient mappings and also reduces the possibility of getting locked in a local maximum.

Hint chains are generated by combining individual hints suggested by the evaluation engines. Since each engine generates its hints independently, certain combination of hints might contain conflicts. That is, two hints might suggest operations on the same visual feature or the same data attribute in a contradictory manner. For example, given data attribute-visual feature pairs (A_1, V_1) and (A_2, V_2) in a mapping M , one hint may suggest swapping of visual features mapped to A_1 and A_2 whereas another hint may suggest discretizing

A_1 . The algorithm to generate hint chains avoids hint conflicts by pruning any chains that contain conflicting hints.

Hint chains are also assigned an estimated improvement weight. This improvement is calculated as an average of the estimated improvement weights of individual hints in the chain. The hint chains are then stored in a priority queue ordered by estimated improvement. The hint chain with the highest estimated improvement is extracted from the priority queue and applied to the current mapping M to generate a new mapping M' . If M' has already been visited, it is ignored. If not, M' is passed to the evaluation engines, new hints are generated, and the process continues. The search terminates when no new mappings can be generated, or when there has been no significant change in the evaluation weight of the best mapping over some fixed number of iterations. Figure 4.1 summarizes the heuristic process used in ViA.

The search process used in ViA can be summarized as follows.

1. An initial mapping M_0 is chosen based on the initial importance ordering of the attributes and the visual salience of the available visual features. Let M^* denote the best data-feature mapping seen so far.
2. If M is already visited, go to step 5. Otherwise, pass M to each of the evaluation engines to determine its evaluation weight, and to generate hints on how to improve M .
3. If the evaluation weight of M is better than M^* , set M^* to M .

4. Generate hint chains from the hints suggested for mapping M , determine their estimated improvement, then place each hint chain together with its parent mapping M into the priority queue Q .
5. If the termination criteria is not satisfied, remove the hint chain with the highest estimated improvement from Q , then apply the hints in the chain to the parent mapping M to generate a new mapping M' . Repeat from step 2.

4.2 Simulated Annealing

Simulated annealing (SA) is a heuristic technique suitable for combinatorial optimization problems. It is a generic probabilistic method for finding an approximation to the global optimum in a large search space. It exploits the analogy between solving combinatorial optimization problems and the process of cooling metals to minimum energy crystalline structures through an *annealing* process. In physical annealing, the metal is set to a high temperature and slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. As cooling proceeds, the system becomes more ordered and approaches a ground state which has little probability for the existence of a high energy state within the material. Hence the process can be thought of as an adiabatic approach to the lowest energy state. If the initial temperature of the system is too low or cooling is done too quickly the system may become quenched, forming defects or freezing out in metastable states (i.e. trapped in a local minimum energy state).

If the energy function of the physical system is replaced by an objective function, then the slow progression towards the ordered ground state is representative of a progression to a global optimum. Any optimization problem can thus be represented as an annealing problem. The entrapment in local minima or maxima is then dependent on the choice of initial and final temperatures and change in temperature as each step of cooling proceeds [GHP⁺97].

4.2.1 The Framework

In order to describe the computational equivalent of this method, we need to define four main components.

1. Possible system configuration i.e., a definition of a solution.
2. A random move generator which picks a random operator at each iteration. The operator is chosen from the set of operators that can be applied to the current state of the search space.
3. An objective function specifying the goal of the procedure.
4. Control parameter T , which is analogous to temperature and a cooling schedule. The cooling schedule determines the number of iterations and the temperature at each iteration before reaching the final temperature [KGV83].

SA is one of the first algorithms that had an explicit strategy to avoid local optimum. The idea is to probabilistically allow moves resulting in solutions that are worse than the current

solution in the hope that one of these moves will help escape the local optimum. Instead of accepting a neighbor s_j of the current solution s_i only when it is better than s_i , SA also accepts s_j when its objective function value is worse than s_i , with probability P_{ij}^A defined as:

$$P_{ij}^A(T) = e^{\left(\frac{-\delta E}{T}\right)} \quad (4.1)$$

Equation 4.1 defines the acceptance probability for solution s_j from current solution s_i as a function of the difference in the objective function values (δE) at s_i and s_j , and the current temperature T . Initially, when the temperature is high, the algorithm accepts almost every new solution, allowing it to explore and decide the direction to an optimum state. As the temperature decreases, fewer changes that lead to inferior solutions are accepted. The hope is that a fairly optimal state is reached at the final temperature. The ability to accept worse solutions and hence the ability to reach the optimal state is governed by the initial temperature, the final temperature based on the termination criteria, and the rate at which cooling takes place, i.e., the cooling schedule. A cooling schedule is defined as a decrement function such as $T_{new} = C * T_{old}$ or $T_{new} = T_{old} - C$ where $C < 1.0$ is a constant.

4.2.2 The Search Algorithm

The SA process as adapted to the problem of searching for the best mapping in ViA is illustrated in algorithm 4.2.1.

Algorithm 4.2.1: SIMULATED ANNEALING(T_0, T_f, M_0)

comment: T_0 is the initial temperature, T_f is the final temperature, M_0

is the initial mapping

$T \leftarrow T_0$

$M \leftarrow M_0$

$M^* \leftarrow M$ Best mapping seen so far

$S \leftarrow$ Set of permissible operators from M

repeat

$\mu \leftarrow$ random $\in S$ (generate a random operator)
 $M' \leftarrow \mu(M)$
comment: $E(n)$ provides the evaluation weight for mapping n
if $E(M') > E(M)$
 then $M \leftarrow M'$
 else $\left\{ \begin{array}{l} M \leftarrow M' \text{ with probability } P = e^{\frac{-(E(M)-E(M'))}{T}} \\ \text{(otherwise } M \text{ remains unchanged)} \end{array} \right.$
if $E(M') > E(M^*)$
 then $M^* \leftarrow M'$
 $T \leftarrow \text{schedule}(T)$

until $T < T_f$

return (M^*)

The algorithm starts by accepting an initial mapping, M_0 , and initializing the initial and final temperatures. It also records the best mapping seen so far. At each step, the algorithm chooses a random mapping, from the *neighborhood* of the current mapping. The *neighborhood* is defined as any mapping M' that can be obtained by applying a permissible operator to the current mapping M . The acceptance of the new mapping is based on the evaluation weight, $E(M)$, of the current mapping, the evaluation weight, $E(M')$, of the new mapping and the current temperature T . If $E(M') > E(M)$, the new mapping is accepted. If $E(M') < E(M)$, it is accepted with a probability $e^{\frac{-(E(M)-E(M'))}{T}}$. The temperature is then changed to a new value based on the cooling schedule and the process is repeated until the final temperature is reached.

The probability of accepting a non-improving mapping decreases as T decreases. Initially when T is high, the probability of accepting downhill moves is high. But as T decreases, the probability decreases and the algorithm degenerates to an iterative improvement algorithm. Hence, SA performs search in two different phases; in the first phase, it performs a random walk permitting an exploration of the search space and in the second phase, it tries to converge to a optimum.

Some of the major disadvantages of SA include:

1. The choice of cooling schedule is critical but difficult. There are no rules to make the selection and it is dependent on the system being optimized.
2. Because of SA's random nature, convergence to the optimal is often very slow.

3. If the system gets locked in a local maximum when the temperature is low, SA will either spend an enormous amount of time before escaping or not find the optimal solution.
4. SA can become trapped in a cycle visiting the same set of nodes (states of the search space) repeatedly.

4.3 Reactive Tabu Search

One of the major drawbacks of SA is, it can get trapped in a cycle, since the choice of an operator at each iteration is independent of the history of the states already visited. One approach to avoid this is to remember the previous states that have been visited. Tabu search (TS) is one such strategy which uses memory to guide the choice of the next move.

TS represents a meta-strategy that complements a local search heuristic with memory-based mechanisms designed to avoid cycles [BT95a]. According to the Webster's Dictionary, "tabu" is a "prohibition imposed by social custom as a protective measure" or is something "banned as constituting a risk". The risk to be avoided in our case is the possibility of following a sequence of nodes that leads to a cycle within a local optimum. At each iteration in the search process, the most recently visited nodes are marked as *tabu*, which means they are prohibited and not considered while making the next move.

The TS scheme can be characterized as a modified greedy search algorithm which incorporates techniques to avoid the occurrence of cycles. It is based on the following two

principles:

1. *Modified Greedy Search.* At each iteration, a node that has the highest objective function value among all the non-tabu nodes reachable from the current node by application of a single operator, is chosen as the next node to visit. It should be noted that this highest value can be lower than the objective function value of the current node. That is, if there are no improving moves from the current node, indicating a possible local optimum, TS chooses a node which least degrades the objective function. This approach differs from the standard greedy search which stops if it cannot find a better node. Acceptance of lower values allows for an attempt to escape from the local maximum.
2. *Cycle Avoidance.* Recently visited nodes are marked as *tabu* nodes in TS. The set of moves which do not lead to tabu nodes are called admissible moves and are considered while choosing the next operator [BT95a].

In order to study the use of tabu nodes to avoid becoming entrapped in a local maximum, consider a combinatorial optimization problem $P(F, E)$ where F denotes the set of possible nodes (i.e. the entire search space) and E is the objective function to be maximized. Let $N(f)$ define the neighborhood of each node f , i.e. $N(f) \subseteq F$. $N(f)$ can also be defined as a set of nodes that can be obtained by applying a set of permissible operators, S , to f . Thus $N(f)$ can be represented as:

$$N(f) = \{g \in F \text{ such that } g = \mu(f) \text{ for } \mu \in S\}$$

At each iteration t , the set of nodes in $N(f^{(t)})$ which are not tabu are called *admissible* nodes and are represented as $A^{(t)}$. The set of tabu nodes at iteration t is represented as $T^{(t)}$. Initially, all nodes are admissible and hence, $A^{(0)} = N(f^{(0)})$, $T^{(0)} = \phi$. At any given iteration t , the next move, $\mu^{(t)}$, is chosen as the best move from the set of admissible moves.

TS aims to reduce the possibility of getting trapped in a cycle with the use of prohibited moves. Consider the modified greedy search algorithm *without* the prohibited moves, and where the current node, $f^{(t)}$, is the local maximum. Every node in $N(f^{(t)})$, has a value lower than $f^{(t)}$. At iteration $t + 1$, the neighbor with the highest value is chosen as the next node, $f^{(t+1)}$. Hence, $E(f^{(t+1)}) = E(\mu^{(t)}(f^{(t)})) < E(f^{(t)})$. Now, while choosing node $f^{(t+2)}$ with the modified greedy search algorithm alone, the best node among the neighbors of $f^{(t+1)}$ is the local maximum and hence,

$$f^{(t+2)} = f^{(t)}$$

At this point, the system will be trapped forever in a cycle of length two [BT95a]. TS avoids this cycle by marking recently visited nodes as *tabu*. These nodes are shifted from $A^{(t)}$ to $T^{(t)}$. The definition of “recent” leads to several different versions of TS.

If recent is measured as ever visited, i.e. a neighbor is prohibited if it has been visited any time in the past, then the algorithm is called **Strict-TS**. Here,

$$A^{(t)} = \{X \in N(f^{(t)}) \text{ such that } X \notin \{f^{(0)}, \dots, f^{(t)}\}\} \quad (4.2)$$

$$\text{and } T^{(t)} = T^{(t-1)} \cup f^{(t)} \quad (4.3)$$

As discussed by Battiti and Tecchiolli [Bat96, BT94], Strict-TS suffers from the problem of slow-convergence and reduced efficiency due to the barriers created by already visited nodes. One solution is to remember only the nodes visited in the last P iterations. P is called the prohibition period. Based on whether P is static or dynamic, two different algorithms have been proposed; Fixed-TS and Reactive-TS.

The variant of TS in which recent is measured as nodes visited in the past P iterations, where P is constant is called **Fixed-TS**. A neighbor is prohibited if it has been visited in the last P iterations. Let $last_used(f)$ define the last time the node f was visited. Hence, equations 4.2 and 4.3 become,

$$A^{(t)} = \{X \in N(f^{(t)}) \text{ such that } last_used(X) = \tau < t - P\} \quad (4.4)$$

$$\text{and } T^{(t)} = T^{(t-1)} \cup f^{(t)} - f^{(t-P)} \quad (4.5)$$

The Fixed-TS algorithm maintains the same prohibition period P throughout the search. As noted by Battiti et al. [BT95b], P should be large enough to avoid cycles and must also be sufficiently small to avoid over-constraining the search. For example, if $P = L$, and the search space size is $L + 1$, after the first P iterations, only one node is present in A . From this point on, the path chosen by TS does not depend on the values of the objective function, and the sequence of the $P + 1$ moves is repeated. The prohibition period has a critical impact on the success of the algorithm. The choice of P without a priori knowledge

of the search space is difficult and also dependent on the specific problem.

One solution is to use a dynamic prohibition period. This variant of TS in which P changes at each iteration t (denoted as $P^{(t)}$), depending on the search status is called **Reactive-TS** (RTS). The RTS scheme overcomes the slow-convergence problem in Strict-TS and suggests a way to change P during the search so that $P^{(t)}$ is appropriate to the local structure of the problem. Hence, equation 4.4 can be restated as,

$$A^{(t)} = \{X \in N(f^{(t)}) \text{ such that } last_used(X) = \tau < t - P^{(t)}\} \quad (4.6)$$

The RTS scheme uses a feedback mechanism to determine the prohibition period $P^{(t)}$.

In the next section, we will study the RTS algorithm in detail.

4.3.1 The Framework

RTS is a heuristic algorithm complemented with a feedback mechanism that uses past history. RTS uses the history for two main purposes:

1. *Parameter Tuning*: RTS automates the tuning of the prohibition period by monitoring the search behavior based on its history of nodes explored.
2. *Intensification and Diversification of Search*: There is a well-known trade-off between the exploration (intensification) of promising regions and exploiting new regions (diversification). Intensification without diversification can trap an algorithm in a local maximum. Diversification without intensification performs an unguided

search, often spending most of its time exploring suboptimal areas. RTS tries to achieve a balance through the feedback mechanism by starting with intensification and switching to diversification when the need arises. The feedback mechanism and the history are used to dictate the need for diversification.

A long term memory remembers all the nodes visited during the search process. When a node is selected, it is checked for repetition. If the node is repeated, then the tabu list size is increased. This is known as the fast “reactive” mechanism of RTS. A slow reduction mechanism is also present in which the list size is reduced. This occurs when the search explores new nodes without any repetitions. When the number of repetitions becomes significant (also called as a “chaotic” state), a diversification mechanism is adopted. The chaotic state serves as evidence that the system is trapped in a local optimum. Diversification is realized by making a random walk through different regions of the search space.

4.3.2 The Search Algorithm

The RTS approach as adapted for ViA is shown in algorithm 4.3.1. The process starts with initialization and then loops until the specified maximum number of iterations is reached. Other termination criteria such as, interval since the last change to the M^* value, or a threshold on the quality of solution can also be used. The system stores the number of times each mapping is visited (repetitions) and the time (iteration number) that it was last encountered. When the number of repetitions exceeds a threshold MAXREP, the mapping is considered an often repeated mapping.

Algorithm 4.3.1: REACTIVE TABU SEARCH($max_iterations, M_0$)**comment:** Initialization

$t \leftarrow 0$ (iteration counter)
 $P^{(0)} \leftarrow 1$ (prohibition period)
 $chaotic \leftarrow 0$ (diversification factor)
 $f^{(0)} \leftarrow M_0$ (Initial mapping)
 $M^* \leftarrow f^{(0)}$ (best mapping so far)
 $E_b \leftarrow E(f^{(0)})$ (best E so far)

while $t < max_iterations$

{ (check if the current configuration is a repetition)
 $escape \leftarrow check_for_repetitions(f^{(t)})$
if $escape = DO_NOT_ESCAPE$
 {
 $\mu \leftarrow choose_best_move(f^{(t)})$
 $f^{(t+1)} \leftarrow \mu(f^{(t)})$
 (Update time and best so far)
 then {
 $t \leftarrow t + 1$
 if $E(f^{(t)}) > E_b$
 then {
 $E_b \leftarrow E(f^{(t)})$
 $M^* \leftarrow f^{(t)}$
 }
 else *diversify* (diversify search)
 }
return (M^*)

The variable *chaotic* counts the number of often repeated mappings. When this value crosses a threshold CHAOS, diversifying movement, *diversify*, is invoked. The iteration counter t provides the current time. $P^{(t)}$, the dynamic prohibition period, is updated based on the frequency of repetitions.

After initialization, the current mapping is searched against the history (with a possible update of $P^{(t)}$), to check if a diversifying move is needed. This is notified by a return value of `ESCAPE` or `DO_NOT_ESCAPE`. If `ESCAPE` is returned, the search enters a diversification phase by performing random movements in the search space. Otherwise, the search evaluates the admissible neighbors of the current mapping, to find the best neighbor. The mapping with the best evaluation weight among the neighbors is selected as the next mapping. If all the neighbors are tabu, the best tabu node is selected. The current time and the best mapping so far are updated. This process is then repeated for *max_iterations* and the best mapping is returned. The feedback procedure to adapt $P^{(t)}$ and the escape mechanism are explained in the following sections.

4.3.2.1 The Feedback Mechanism

The history of the search is used to adapt the prohibition period $P^{(t)}$ and to initiate diversification. The feedback mechanism of the algorithm is controlled by the frequency of repetitions and causes changes to $P^{(t)}$ or activation of the escape mechanism. Algorithm 4.3.2 illustrates the reactive mechanism in detail. The current mapping M is searched in memory.

If M is found, its last visited time ($M.LastVisit$) and repetition counter ($M.repeat$) are updated. If the number of repetitions is greater than the threshold `MAXREP`, M is included in the set of often repeated mappings. The often repeated counter *chaotic* is updated and if *chaotic* exceeds its threshold `CHAOS`, the function returns immediately notifying a need to escape. In order to discourage cycles, $P^{(t)}$ is increased. If M is not found in memory then it is stored along with the time it was visited and its repetition counter is set to 1.

Algorithm 4.3.2: CHECK_FOR_REPETITIONS(M)

comment: The function returns ESCAPE if diversification is necessary,
 else DO_NOT_ESCAPE

comment: Search the history for ϕ

if $History_Search(M) \neq NULL$

then	{	(Update repeat counter, last visit time)															
		$M.LastVisit \leftarrow t$															
		$M.repeat \leftarrow M.repeat + 1$															
		if $M.repeat > MAXREP$															
		<table border="0"> <tr> <td rowspan="3" style="vertical-align: middle; padding-right: 10px;">then</td> <td rowspan="3" style="vertical-align: middle; font-size: 3em; padding-right: 10px;">{</td> <td>(Add to the set of often repeated mappings)</td> </tr> <tr> <td>$chaotic \leftarrow chaotic + 1$</td> </tr> <tr> <td>if $chaotic > CHAOS$</td> </tr> <tr> <td></td> <td></td> <td> <table border="0"> <tr> <td rowspan="2" style="vertical-align: middle; padding-right: 10px;">then</td> <td rowspan="2" style="vertical-align: middle; font-size: 3em; padding-right: 10px;">{</td> <td>(Reset chaotic counter and indicate escape)</td> </tr> <tr> <td>$chaotic \leftarrow 0$</td> </tr> <tr> <td></td> <td></td> <td>return ($ESCAPE$)</td> </tr> </table> </td> </tr> </table>	then	{	(Add to the set of often repeated mappings)	$chaotic \leftarrow chaotic + 1$	if $chaotic > CHAOS$			<table border="0"> <tr> <td rowspan="2" style="vertical-align: middle; padding-right: 10px;">then</td> <td rowspan="2" style="vertical-align: middle; font-size: 3em; padding-right: 10px;">{</td> <td>(Reset chaotic counter and indicate escape)</td> </tr> <tr> <td>$chaotic \leftarrow 0$</td> </tr> <tr> <td></td> <td></td> <td>return ($ESCAPE$)</td> </tr> </table>	then	{	(Reset chaotic counter and indicate escape)	$chaotic \leftarrow 0$			return ($ESCAPE$)
then	{	(Add to the set of often repeated mappings)															
		$chaotic \leftarrow chaotic + 1$															
		if $chaotic > CHAOS$															
		<table border="0"> <tr> <td rowspan="2" style="vertical-align: middle; padding-right: 10px;">then</td> <td rowspan="2" style="vertical-align: middle; font-size: 3em; padding-right: 10px;">{</td> <td>(Reset chaotic counter and indicate escape)</td> </tr> <tr> <td>$chaotic \leftarrow 0$</td> </tr> <tr> <td></td> <td></td> <td>return ($ESCAPE$)</td> </tr> </table>	then	{	(Reset chaotic counter and indicate escape)	$chaotic \leftarrow 0$			return ($ESCAPE$)								
then	{	(Reset chaotic counter and indicate escape)															
		$chaotic \leftarrow 0$															
		return ($ESCAPE$)															
		$P^{(t)} \leftarrow INCREASE(P^{(t)})$															
else	{	(mapping not found, hence add it)															
		$History_Insert(M, t)$															

if steps since last repetition $> THRESHOLD$

then $P^{(t)} \leftarrow DECREASE(P^{(t)})$

return (DO_NOT_ESCAPE)

When there have not been enough repetitions in the past, a gradual reduction strategy is employed. A constant THRESHOLD is used to determine when to decrease $P^{(t)}$.

4.3.2.2 The Escape Mechanism

The reactive mechanism tries to ensure that cycles are avoided. It does not provide any guarantee that the search is not confined to a limited portion of the search space. Becoming trapped in a sub-region of the search space is still possible. In order to avoid this, a diversification phase is provided. It is triggered when too many mappings are repeated too often [Bat96]. Figure 4.3.3 shows the escape mechanism in detail.

The escape mechanism, clears the history and takes random steps from the current mapping. At each step the mapping visited is made *tabu* to avoid returning to the old region. If a better mapping is found, the M^* mapping is also updated.

The main drawback with the RTS approach is the use of large amounts of memory to retain the history. But as Battiti et al. state, the overhead caused by the memory usage and adaptation mechanisms of RTS are usually outweighed by avoiding possible “traps” in the search process [BT94].

Hint-based search, SA and RTS use different mechanisms to find the best mappings. Each algorithm has its strengths and weaknesses. In order to assess their abilities, we analyzed each algorithm in terms of its behavior and performance. Results from our analysis are included in the next two chapters.

Algorithm 4.3.3: DIVERSIFY(M)

comment: Random mappings in the search space are visited and be-

come tabu as soon as visited

(Clean the memory structure or history)

$steps \leftarrow$ (Random number of steps to be executed)

$f^{(t)} \leftarrow M$

$P^{(t)} \leftarrow 0$

for $i \leftarrow 1$ **to** $steps$

do $\left\{ \begin{array}{l} \mu \leftarrow \text{(Random operator from the given set of operators)} \\ f^{(t+1)} \leftarrow \mu(f^{(t)}) \\ \text{(Update tabu list, time and best so far)} \\ P^{(t)} \leftarrow INCREASE(P^{(t)}) \\ t \leftarrow t + 1 \\ \text{if } E(f^{(t)}) > E_b \\ \quad \text{then } \left\{ \begin{array}{l} E_b \leftarrow E(f^{(t)}) \\ M^* \leftarrow f^{(t)} \end{array} \right. \end{array} \right.$

Chapter 5

Hyperbolic Visualization

In the previous chapter, we studied the three search strategies that are being studied within ViA. In order to compare these strategies, we examine the regions of the search space explored by each algorithm.

5.1 Motivation

One way to investigate the areas explored by a search strategy, is to visualize the search space for the problem domain and mark the visited regions. Intuitively, a search algorithm that visits only a small number of states, yet still manages to locate most (or all) of the goal states is highly efficient. A visualization should, therefore, show the desirability of each state within the search space and the nodes visited by the strategy during the search process. ViA's search space can be mapped to a directed graph where nodes represent solutions and edges represent operators. ViA's search space grows combinatorially with the number of

visual features and changes to user constraints, and hence the graph of solutions is large in most cases. Also, the graph may contain cycles of arbitrary length, since the application of an operator to one state can lead to a previous state from which it was derived.

Coherent and effective layout of large graphs is a difficult problem. Size and cross-linking of nodes are some of the major issues studied in graph visualization [HMM00]. One method to overcome these problems is to convert the graph into a tree. Although the visualization of a tree also needs careful layout, it does not suffer from cross-links or cycles. The approach that we have adopted in our system is to reduce ViA's directed graph of solutions to a directed tree.

A tree representation of a graph contains the same nodes as the graph, but with some of the edges discarded to remove cycles. Since we want to display the edges traversed by the search algorithms in ViA, arbitrarily removing edges might pose a problem. We therefore use a modified breadth first traversal algorithm which uses domain knowledge to construct trees that try to retain the edges visited by the different search algorithms. Given this tree representation, the search algorithm can be viewed as trying to identify a path from the root of the tree to a node with an optimal state. The problem of generating a visualization of the search space can now be reduced to finding a meaningful spatial mapping of the tree of solutions onto the display area.

We define a meaningful spatial mapping as one which facilitates two main goals. First, we need to visualize contextual information about the search space as a whole. At the same time, the visualization should convey useful information at a more detailed level. Thus our second goal is to facilitate focus on nodes of interest.

One conventional approach is to map the search space onto the screen and use scrolling mechanisms to move within it. Although this is simple, it violates our first goal. The context of the display with respect to its overall structure is lost. To avoid this, we can reduce the amount of screen space allocated to each node. Unfortunately, this increases clutter and reduces the expressiveness of the visualization, violating our second goal: making it easy to focus on nodes of interest.

One limitation of these traditional approaches is that their display techniques use Euclidean geometry. As the number of visual features and changes to the initial user constraints increase, the search space size grows combinatorially. However, Euclidean space only grows linearly. What we want is a geometric space that grows in a way that can accommodate the combinatorially growing search space. Hyperbolic geometry provides one such solution. Hyperbolic space has been widely used in different domains of information visualization. The first hyperbolic visualization system was the 2D hyperbolic tree browser, designed by researchers at Xerox PARC [LR94]. A more recent system, H3, visualizes large trees in 3D hyperbolic space [Mun97]. In this thesis, we adapt a similar approach to visualize the search space of ViA.

5.2 Hyperbolic Geometry

Hyperbolic geometry, also called saddle geometry or Lobachevskian geometry, is obtained by replacing the parallel postulate of Euclidean geometry with a hyperbolic postulate. The parallel postulate of Euclidean geometry states that, given a point P and a line

L , there is exactly one line which passes through P and is parallel to L . In contrast, in hyperbolic geometry there are infinitely many lines which pass through P and are parallel to L . Also, as opposed to Euclidean geometry, parallel lines in hyperbolic geometry are not equidistant. As shown in Figure 5.1, two lines L_1 and L_2 are perpendicular to a third

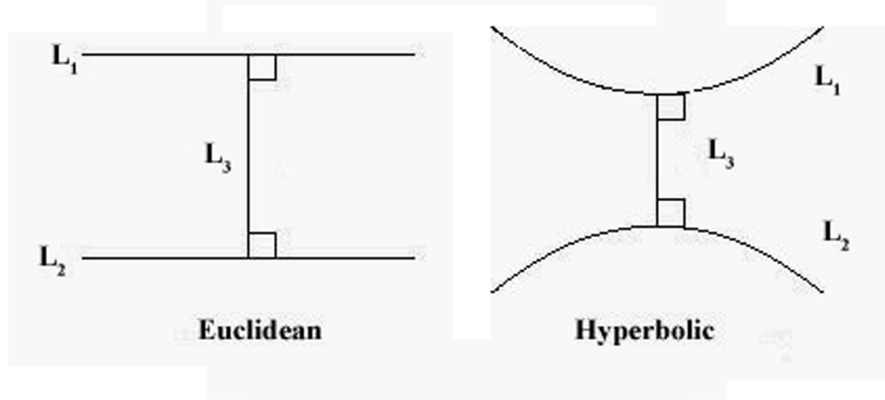


Figure 5.1: Parallel Lines in Euclidean geometry are equidistant. Distance between the parallel lines increases as one goes away from the center in hyperbolic geometry

line, L_3 . In Euclidean geometry, any line segment perpendicular to L_1 and L_2 has the same length as L_3 . However, in hyperbolic geometry, the distance between lines L_1 and L_2 increases as we move away from L_3 .

An interesting property of hyperbolic geometry is that the circumference and area of a circle grow exponentially with its radius. The circumference, c , and area, a , of a circle with radius r in a hyperbolic plane are defined as $c = 2\pi \sinh(r)$ and $a = 4\pi \sinh^2(\frac{r}{2})$. For small values of r , $c \approx 2\pi r$ and $a \approx \pi r^2$ as in Euclidean space. As r increases, c and a increase exponentially. This explains the exponential growth in circumference and area in a hyperbolic space. [Mar75], [Wol45], [Cox88], [Mun97] provide detailed explanations

of hyperbolic geometry and its application in different domains.

In order to visualize the hyperbolic space, we need to map it onto the Euclidean space. Several models have been defined to map the hyperbolic world onto the Euclidean space. In this thesis, we have chosen to use the Poincare disk model which maps the entire hyperbolic space onto a unit disk in Euclidean space. Consider a point $P(d, \theta)$ at distance d and angle θ from the origin in hyperbolic space. Poincare disk model maps $P(d, \theta)$ to $P'(f(d), \theta)$, in Euclidean space where $f(d) = \frac{e^d + e^{-d} - 2}{e^d + e^{-d} + 2}$. Figure 5.2 shows $f(d)$ plotted as a function of d . It can be seen that as $d \rightarrow \infty$, $f(d) \rightarrow 1$.

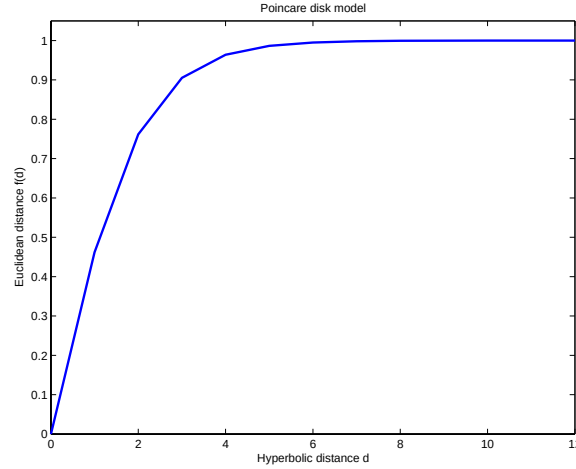


Figure 5.2: Poincare Disk Model.

Consider concentric circles with radii $\{r_1, r_2 \dots r_n\}$ such that $r_{n+1} = r_n + x$ (where x is a constant in hyperbolic space). When these circles are mapped to Euclidean space using the Poincare model, it produces radii $\{r'_1, r'_2, \dots r'_n\}$ in Euclidean space, with $r'_n \rightarrow 1$ as $r_n \rightarrow \infty$. Thus the distance between two consecutive circles $(r'_{n+1} - r'_n) \rightarrow 0$ as we move away from the center (Figure 5.3).

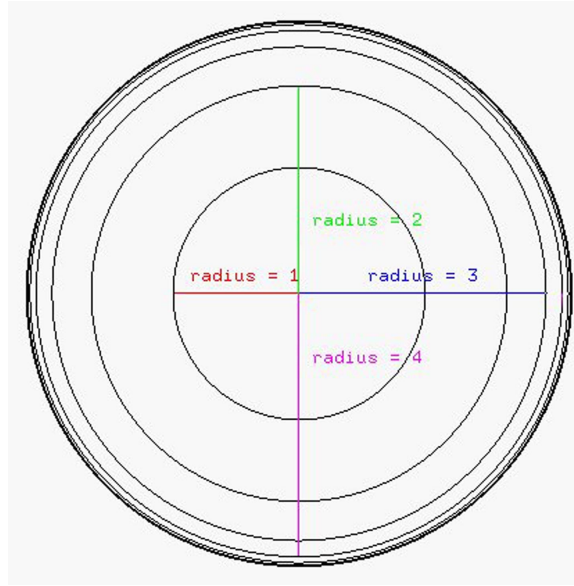


Figure 5.3: Concentric circles with constantly increasing radii in hyperbolic space are mapped to Euclidean space using Poincare model.

Circles near the center of the hyperbolic disk get a proportionately larger display area than circles away from the origin. This allows the creation of visualizations that display the details of objects that are placed near the center of the disk and at the same time display an overview of the information space towards the periphery of the disk. Using the tools provided by hyperbolic geometry and the Poincare model, we have developed a focus + context visualization system, for visualizing the search space of ViA.

5.3 Implementation

In order to visualize ViA's search space, we begin by generating the nodes that make up the search space. This is done by a systematic and exhaustive application of operators. We start with a node containing an initial mapping of data-attributes to visual features and

the attribute characteristics as specified in the initial input to ViA. Then the *feature_swap* operator is applied repeatedly to generate all permutations of attribute to feature mapping. We then modify the initial attribute characteristics by applying the *discretize* and *importance_weight_modify* operators to generate additional nodes. Application of an operator to an existing parent node leads to the generation of child node. Only the nodes that satisfy the rules in ViA are used as a part of the search space. For instance, applying an operator can cause a loss of information. This is disallowed by ViA. Hence, nodes formed by applying such operators are not a part of the search space. The collection of valid nodes, along with the operators that convert between nodes form a graph representation of ViA's search space. Once the graph is formed the modified breadth first search is used to convert the graph into a directed tree. Visualization of the tree is then generated by defining the layout of the nodes in the search space in the hyperbolic plane and projecting them onto the Euclidean plane.

We next lay out the nodes of the search space represented as a tree evenly in the hyperbolic plane. The placement of a node is defined by a pair (r, ϕ) , where r is the distance from the center of the parent and ϕ denotes the angle. The root of the tree is placed at the center of the circle. The child nodes are placed at a fixed distance from the parent. The angular space around the parent node is then equally divided among the child nodes.

Once the location of each node on the hyperbolic plane is determined, we use the Poincare disk model to map the nodes from hyperbolic plane onto the Euclidean plane. Each node is visualized as a cylinder with constant radius. The height of the cylinder rep-

resents the evaluation weight of the mapping at the given node (taller for higher evaluation weight). To bolster the recognition of good mappings, we also use color to encode the evaluation weight of a mapping. Red cylinders represent mappings with higher evaluation weights, while blue cylinders represent mappings with lower evaluation weights. The point of focus can be changed by clicking on the desired node and a detailed view of the node is provided by transforming it to the center of the disk. Figure 5.4 illustrates a hyperbolic visualization for a search space of 120 mappings with three visual features and three data attributes. The short blue cylinders highlight mappings with low evaluation weights, and the tall red cylinders highlight the desired high evaluation weight mappings.

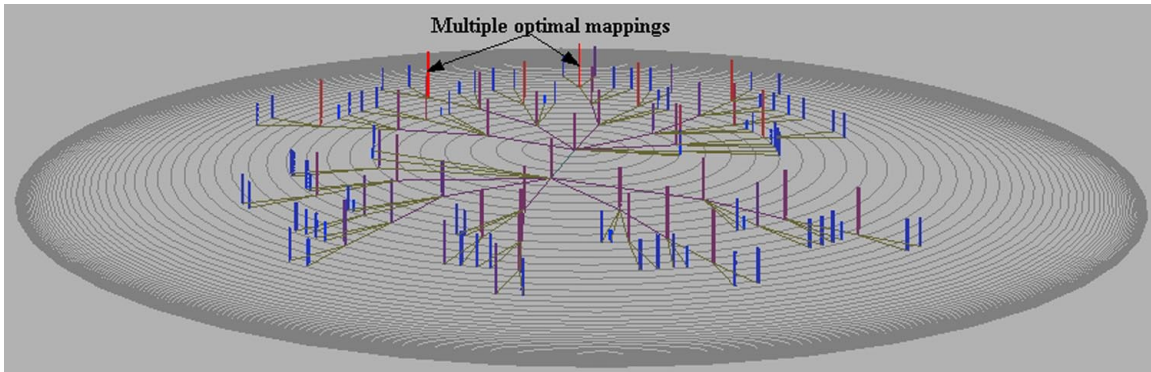


Figure 5.4: Hyperbolic Visualization of Search space with 3 features and 3 attributes.

Some conclusions that can be drawn from this visualization include:

- There are a number of mappings with the same evaluation weight in different regions of the search space.
- There can be multiple mappings with the highest evaluation weight (i.e. multiple goal states). These mappings can be reached using different paths from the initial

node. This means that different search strategies can explore different regions in the search space and still find an optimal solution.

Hyperbolic visualization provides a tool that allows us to explore the details of areas of interest while still maintaining the overall context of the search space.

Chapter 6

Analysis

In order to study the performance of hint-based search, SA and RTS, we evaluated each strategy based on the following metrics:

- Time Efficiency: How long does it take to find the solution?
- Number of optimal solutions found: How many of the optimal solutions does the strategy find (if more than one exists)?
- Space Efficiency: How much memory does it require to perform the search?
- Optimality: Does the strategy find an optimal solution?

To compare the search strategies, we applied ViA to a real world dataset representing monthly weather conditions in $\frac{1}{2}^\circ$ latitude and longitude steps across the continental United States. The weather conditions for each month represent monthly averages over the years 1961 to 1990 and were provided by the Intergovernmental Panel on Climate Change (IPCC).

Attribute	Domain	Frequency	Task	Importance
<i>temperature</i>	discrete (7 unique values)	high	search	1.0
<i>wind speed</i>	discrete (23 unique values)	low	boundary	0.75
<i>pressure</i>	discrete (296 unique values)	low	boundary	0.15
<i>precipitation</i>	discrete (82 unique values)	high	search	0.75

Table 6.1: Characteristics of each attribute: domain type (including the number of unique values for each attribute), frequency type, analysis task, and normalized importance weight.

The dataset contains eleven data attributes: *temperature*, *wind speed*, *pressure*, *precipitation*, *cloud cover*, *diurnal temperature range*, *ground-frost frequency*, *maximum temperature*, *minimum temperature*, *radiation*, *vapor pressure* and *wet day frequency*.

We used ViA to visualize four attributes: *temperature*, *wind speed*, *pressure* and *precipitation*. Table 6.1 summarizes the characteristics of these attributes. ViA uses four visual features: colour, density, orientation and height to represent these attributes. We begin our analysis by visualizing the search space of data-feature mappings that can be generated by ViA and observing the areas of the search space explored by each algorithm. We then describe the performance of each algorithm with respect to the stated performance metrics.

6.1 Search Space Visualization

Figure 6.1 shows the search space for the weather dataset with four data attributes and four visual features. The search space contains 1680 nodes. The tall, bright red cylinders show

the nodes with highest evaluation weights. There are 21 optimal mappings with the highest evaluation weight of 0.844.

In order to compare the areas of the search space explored by ViA when using different search strategies, we ran ViA on the weather dataset with an initial mapping of *temperature* $\rightarrow$ *density*, *wind speed* $\rightarrow$ *height*, *pressure* $\rightarrow$ *colour*, and *precipitation* $\rightarrow$ *orientation*. For all the experiments described in this chapter, the parameters used were as follows: For each run of SA, we empirically chose an initial temperature of 17° and a final temperature of 0.001° , with a proportional cooling schedule at the rate of 0.95. For RTS, MAXREP, the maximum allowed repetitions before marking a mapping as often repeated, was two. CHAOS was two or more often repeated mappings. Hint-based search does not require any control parameters from the user. We used a termination criterion of 200 evaluations for each run. The number of evaluations and not the number of iterations was chosen as the termination criteria since the amount of work done per iteration by each search algorithm is different and hence the number of iterations is not a good metric of comparison. During each run, the nodes explored by each search strategy were recorded and visualized as bright yellow glyphs. The edges explored are marked in white. The optimal nodes found by a search strategy are marked in green.

The initial node representing the initial data-feature mapping is at the center of the disk and is colored white. The entire search space can be divided into five main regions that are branches of the search tree: B1, B2, B3, B4 and B5. B1, B2, B3 and B4 are the branches explored by one or more of the search algorithms. B5 comprises of the rest of the search space. Normally, some large set of nodes would be unexplored since we

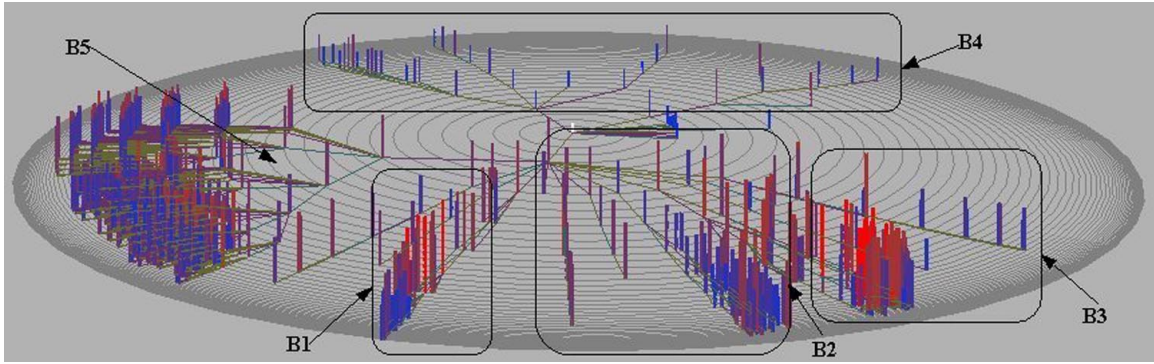


Figure 6.1: Search space with focus on area of interest.

terminate our search after evaluating a total number of nodes that represents only a small percentage of the entire search space. In our visualization, these unexplored nodes are contained in the region B5. The specific nodes left unexplored depend on the manner in which an algorithm pursues the goal state. We highlight this partitioning by using the modified breadth first traversal algorithm described in Chapter 5 to construct our tree. This algorithm gives preference to the edges of the search space that have been explored. Nodes that are not visited are included into the search tree last. This concentrates unexplored nodes in a single branch of the tree.

There are 16 optimal mappings in the four main branches explored by the search algorithms, four in B1 and 12 in B3. The remaining five optimal mappings are located in B5 (and therefore were not found by any of the search algorithms in our 200 evaluations limit). The next sections show the branches of the search space explored and identify which optimal mappings were found by each search algorithm.

6.1.1 Search Space Explored by Hint-Based Search

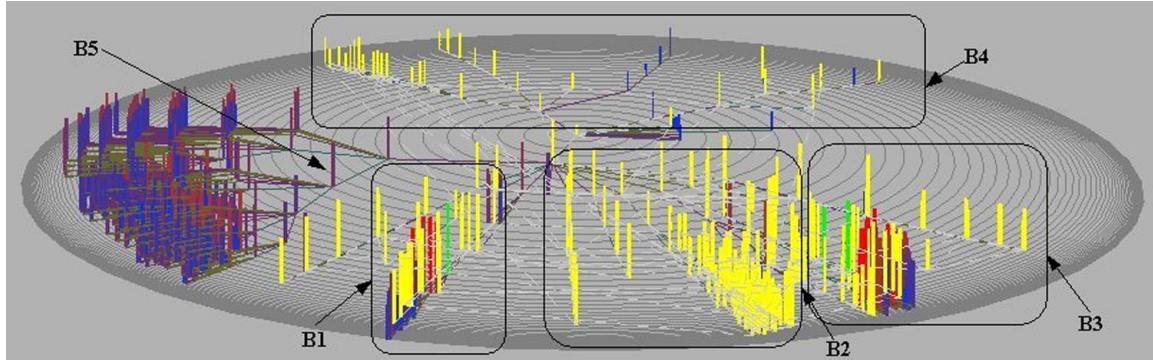


Figure 6.2: Search space explored by hint-based search.

Figure 6.2 depicts the areas explored by hint-based search. Hint-based search samples the first four named regions of the search tree. Since hint-based search performs multi-step moves using a combination of operators, these nodes are distributed widely over the areas of interest. Hint-based search finds three of the optimal mappings, one in B1 and two in B3 in the allowed number of evaluations. The first optimal mapping is found after 76 evaluations. The reason hint-based search could not find the other optimal mappings in B1 and B3 is because it uses perceptual rules that provide estimated promising moves that should be made at the next step independent of the available operators.

6.1.2 Search Space Explored by SA

Figure 6.3 displays the nodes explored by SA. Since SA is random in nature, multiple runs of SA for the same input, visit different regions of the search space and lead to different results. The visualization shown in Figure 6.3 displays the nodes encountered during one representative run of SA. SA visits branches B1, B3 and B4. Since SA evaluates a random

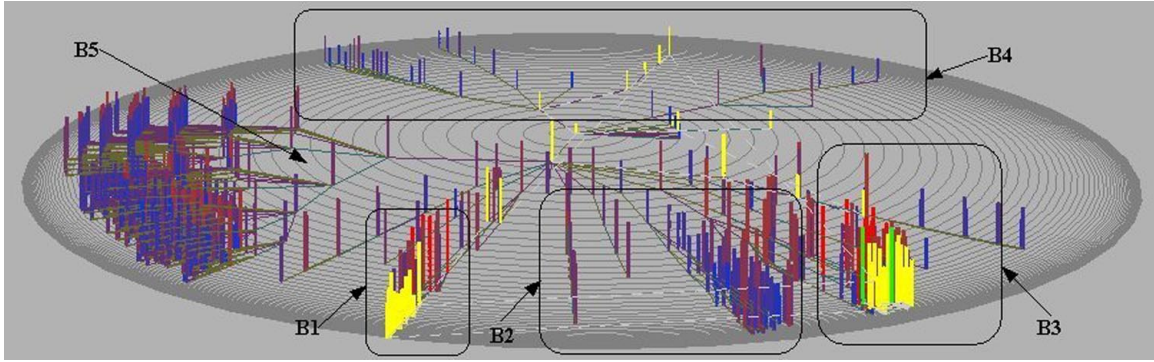


Figure 6.3: Search space explored by SA.

neighbor at each step, the visited nodes are concentrated within each region. SA finds two of the 16 optimal mappings, both in B3. The optimal mappings found by SA are different from those found by hint-based search. This shows that different search strategies can find different optimal mappings. The first optimal mapping was found after 134 evaluations.

6.1.3 Search Space Explored by RTS

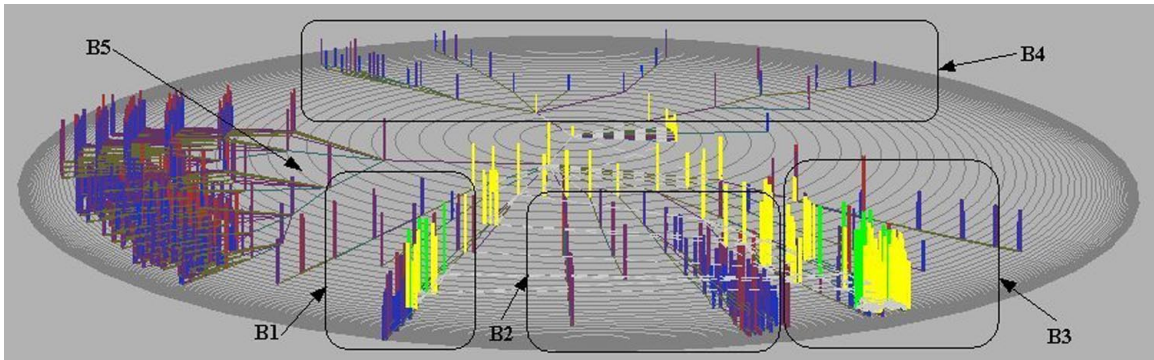


Figure 6.4: Search space explored by RTS.

The nodes explored by RTS are shown in Figure 6.4. RTS performs a complete neighborhood search, that is, it explores all the neighbors of the current node. RTS starts by

evaluating the neighbors of the root, which is the initial state. It chooses to spend almost all its time in branches B1 and B3. It finds all 16 optimal mappings in these regions. The first optimal mapping is found after 68 evaluations.

6.2 Performance

In order to further compare hint-based search with SA and RTS, several runs of ViA with different initial inputs were made, and the results were compared for time and space efficiency, ability to find multiple optimal solutions, and optimality.

6.2.1 Time Efficiency

For timing measurements, clock time spent by the algorithm does not provide a reliable estimate of the time required to find an optimum state. We instead use the number of evaluations performed, which is proportional to the time spent to find an optimal state.

6.2.1.1 Hint-based Search Versus RTS

To compare the performance of hint-based search with RTS, several runs with different initial mappings were performed. These mappings are described in Tables 6.2-6.5. The results are shown in Figures 6.5(a)-6.5(d). In each of these graphs, the horizontal axis represents the number of evaluations and the vertical axis represents the evaluation weight of the best mapping, M^* , seen to date, represented as a percentage of the evaluation weight of the optimal mapping. The search space explored by each algorithm is as shown by the

visualization in Figure 6.1. As mentioned earlier, optimal mappings have an evaluation weight of 0.844.

The circles on each performance line in Figure 6.5 represent the iterations where a step is taken by the corresponding algorithm. A step defines a move from the current node to the next node. The graph of hint-based search appears as a thick line because a step is taken every time a mapping is evaluated. Hence, the circles overlap each other giving the appearance of a thick line. RTS visits the entire neighborhood before taking its next step and therefore evaluates a larger region of the search space than hint-based search before making a move.

Hint-based search reaches approximately 90% of the optimal value within the first 20 evaluations. In fact, in Figure 6.5(a), it improves the mapping by almost 25% in the first 10 evaluations. At each step, hint-based search uses the improvement estimates provided by the evaluation engines to make its decision about the next move. Hence, if the improvement estimates provided are always correct, hint-based search will always move towards an optimal mapping. The multi-step moves provided by the hint chains can also improve the efficiency of hint-based search. This explains the rapid increase in the best evaluation weights found by hint-based search during its initial phase.

However, hint-based search can become trapped for long periods without improvement. For example, in Figure 6.5(a) hint-based search does not find a better mapping between point A and point B, in spite of a large number of evaluations. The best mapping found holds at 98% of the optimal until point B is reached. The same problem can be seen in

Attribute	Feature	Domain	Frequency	Task	Importance
<i>temperature</i>	density	discrete(7)	high	search	1.0
<i>wind speed</i>	height	discrete(23)	low	boundary	0.75
<i>pressure</i>	colour	discrete(296)	low	boundary	0.15
<i>precipitation</i>	orientation	discrete(82)	high	search	0.75

Table 6.2: Initial state for run (a) with evaluation weight of 0.537

Attribute	Feature	Domain	Frequency	Task	Importance
<i>temperature</i>	density	discrete(7)	high	search	1.0
<i>wind speed</i>	colour	discrete(23)	low	boundary	0.75
<i>pressure</i>	orientation	discrete(296)	low	boundary	0.15
<i>precipitation</i>	height	discrete(82)	high	search	0.75

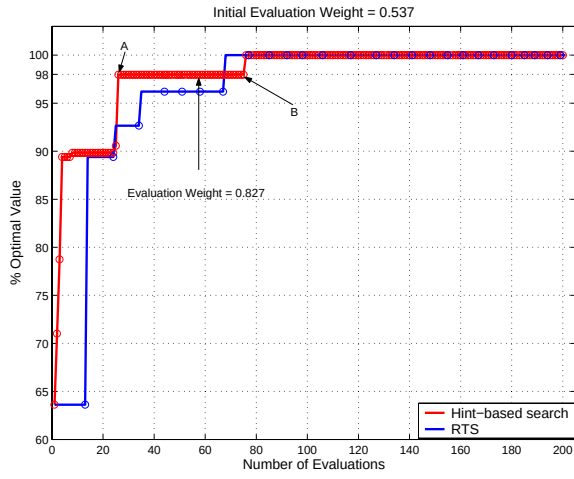
Table 6.3: Initial state for run (b) with evaluation weight of 0.671

Attribute	Feature	Domain	Frequency	Task	Importance
<i>temperature</i>	orientation	discrete(12)	high	search	1.0
<i>wind speed</i>	colour	discrete(23)	low	boundary	0.75
<i>pressure</i>	height	discrete(296)	low	boundary	0.15
<i>precipitation</i>	density	discrete(82)	high	search	0.85

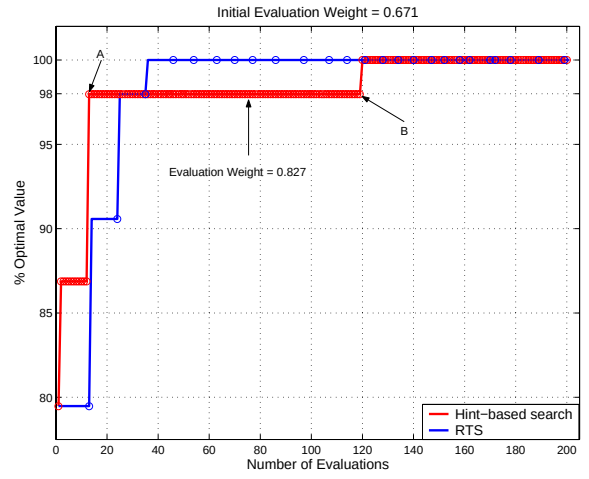
Table 6.4: Initial state for run (c) with evaluation weight of 0.671

Attribute	Feature	Domain	Frequency	Task	Importance
<i>temperature</i>	height	discrete(5)	high	search	0.85
<i>wind speed</i>	colour	discrete(23)	low	boundary	0.85
<i>pressure</i>	density	discrete(296)	low	boundary	0.15
<i>precipitation</i>	orientation	discrete(82)	high	search	0.75

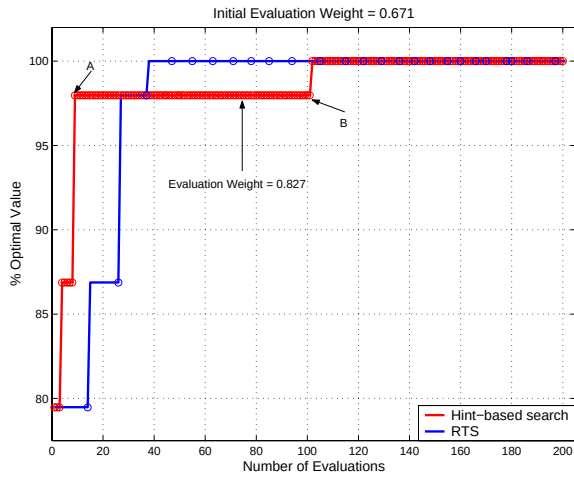
Table 6.5: Initial state for run (d) with evaluation weight of 0.827



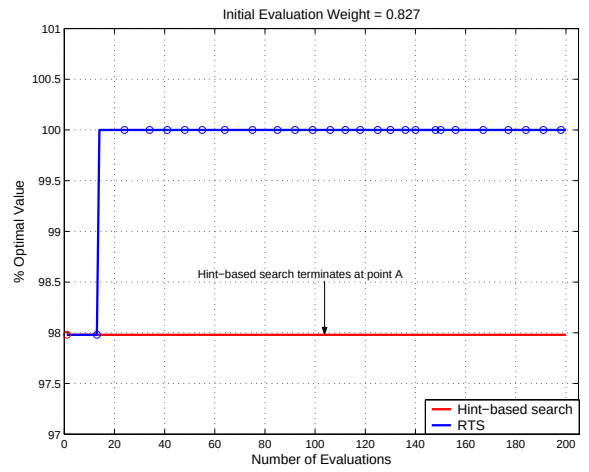
(a)



(b)



(c)



(d)

Figure 6.5: Number of evaluations needed to reach the optimal evaluation weight

Figures 6.5(b) and 6.5(c). In order to study this behavior, we conducted a final experiment where the initial mapping was set to the state at point A in Figure 6.5(a). The result obtained is illustrated in Figure 6.5(d). This experiment is explained in detail in the next section.

6.2.1.2 Initial input as Point A

We repeated our analysis with the initial mapping set to the state at Point A in Figure 6.5(a). The mapping and attribute characteristics at point A are shown in Table 6.5. Using hyperbolic visualization, we studied the neighbors of point A (Figure 6.6). It can be clearly seen that point A is only one step away from an optimal node. One operator, namely *discretize(12)*, needs to be applied to *wind speed* to reach the optimal node. In spite of this, hint-based search is unable to find the solution. This is due to the behavior of the evaluation engines.

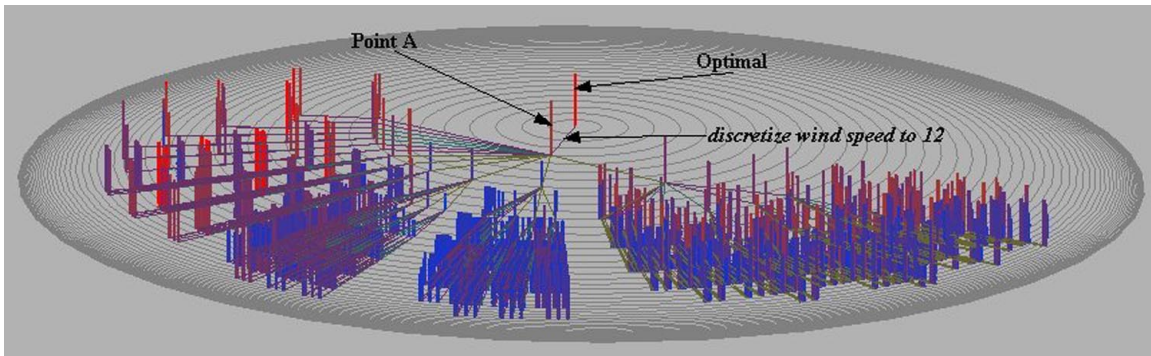


Figure 6.6: Optimal is one step away from Point A.

At each step in the search process, hint-based search uses a subset of the operators as the hints to make the next move. The evaluation engines generate hints based on rules of human perception, independent of the actual operators that may be available at the current

search state. Hints are those perceptually optimal operators that improve the evaluation of a data-feature pair in the mapping. At point A, *wind speed* is mapped to colour. When the colour engine evaluates the mapping, it realizes that colour can best support seven discrete values, so it suggests discretizing *wind speed* to seven values. Discretization of *wind speed* to seven values violates ViA's rules of minimizing data loss and hence this operator is rejected. A request to discretize wind speed to 12 values is never made, since the colour evaluation engine does not see that choice as perceptually valid. Hint-based search is therefore not able to find the optimal solution from point A. In the previous experiments shown in Figures 6.5(a)-6.5(b), hint-based search did find the optimal mapping because the priority queue maintains all the hint chains that have been generated but so far not applied. When hint-based search was not able to proceed from point A, it began to consider lower-weight hint chains generated from earlier mappings. When these previously ignored mappings were more fully explored, a different path to the optimal mapping was found. When we start with point A as the initial mapping, the priority queue does not contain any hint chains from previously evaluated mappings and hence the search terminates at the initial mapping.

6.2.1.3 RTS in a Local Maximum

Although, RTS found an optimal result in each case, it is not without weaknesses. The efficiency of RTS is negatively affected when it is trapped in a local maximum. This behavior of RTS is shown in Figures 6.8(a) and 6.8(b). The initial states for Figure 6.8(a) and 6.8(b) are shown in Tables 6.6 and 6.7. Point C in Figure 6.7, is a local maximum (i.e. none of

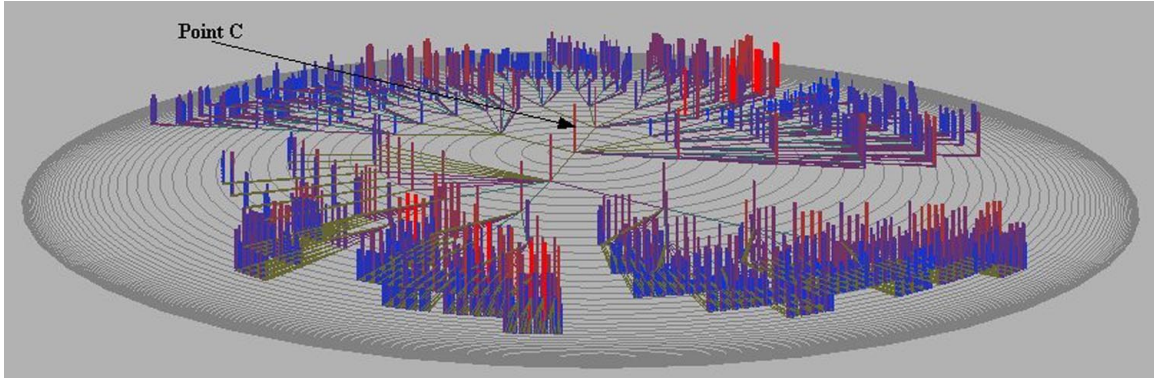
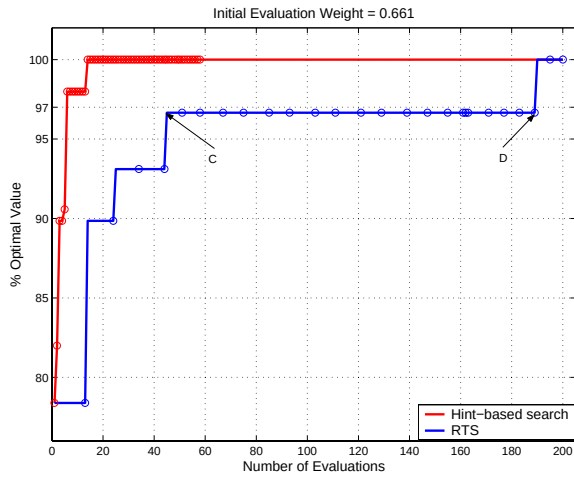
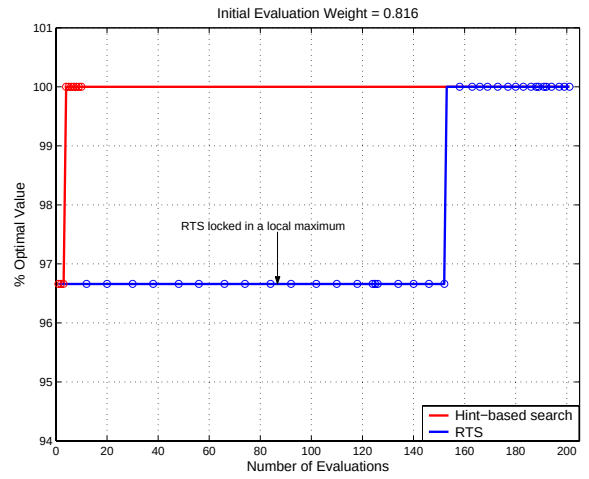


Figure 6.7: Point C is the local maximum.

its neighbors are taller or have higher intensities of red). RTS takes a long time to decide that an escape mechanism is necessary. In this experiment, MAXREP and CHAOS are set to two. Hence, at least four repetitions of a state are needed before escape can be activated. Use of tabu nodes discourages repetitions. Also, at each modification, RTS evaluates the entire neighborhood of the current node. Hence, a large number of evaluations are done before diversification occurs. This explains the long period of time spent between point C and point D without any progress. In fact, in Figure 6.8(a), the algorithm first entered the local maximum region after 45 evaluations, but the escape mechanism is activated only after 189 evaluations. This degrades the efficiency of RTS. Figure 6.8(b) shows similar results with point C chosen to be the initial mapping. Table 6.8 summarizes the number of evaluation taken by each algorithm to reach the highest evaluation weight, i.e. find the first optimal mapping.



(a)



(b)

Figure 6.8: RTS spends more time in a local maximum

Attribute	Feature	Domain	Frequency	Task	Importance
<i>temperature</i>	colour	discrete(12)	high	search	0.8
<i>wind speed</i>	height	discrete(23)	low	boundary	0.8
<i>pressure</i>	density	discrete(296)	low	boundary	0.15
<i>precipitation</i>	orientation	discrete(82)	high	search	0.85

Table 6.6: Initial state for Figure 6.8(a)

Attribute	Feature	Domain	Frequency	Task	Importance
<i>temperature</i>	height	discrete(5)	high	search	0.8
<i>wind speed</i>	orientation	discrete(12)	low	boundary	0.8
<i>pressure</i>	density	discrete(296)	low	boundary	0.15
<i>precipitation</i>	colour	discrete(82)	high	search	0.85

Table 6.7: Initial state for Figure 6.8(b)

Experiment	Number of Evaluations by hint-based search	Number of evaluations by RTS
Table 6.2	76	68
Table 6.3	120	36
Table 6.4	102	44
Table 6.5	Did not find ¹	14
Table 6.6	14	190
Table 6.7	4	153

Table 6.8: Hint-based search Vs RTS: Number of evaluations to find the first optimal mapping

6.2.1.4 Hint-based Search Versus SA

Being random in nature, SA performs a different number of evaluations over multiple runs for the same input and the same initial state. Hence, a single graph of the number of evaluations versus the percentage of optimal value reached cannot be used for evaluation. Instead, we conducted 1000 runs of SA for each input shown in Tables 6.2-6.7, with the same control parameters. The number of evaluations performed to reach the first optimal mapping and the best mapping found are averaged in each experiment. Table 6.9 compares the average number of evaluations performed by SA with the number of evaluations performed by hint-based search to find the first optimal mapping.

When hint-based search fails to choose the operators needed to move to the next best mapping (as shown in Figures 6.5(a)-6.5(d)), it can take more evaluations than SA to terminate. However, out of the 1000 runs for each input, SA could not find an optimal for approximately 230 runs i.e. for about 23% of the runs³. Whereas, hint-based search is not able to find an optimal mapping in one of the six experiments (i.e. Table 6.5).

¹However, it reaches 98% of the optimal value.

³However, it reaches approximately 97% of the optimal value.

Experiment	Number of Evaluations by hint-based search	Average number of evaluations by SA
Table 6.2	76	94
Table 6.3	120	98
Table 6.4	102	80
Table 6.5	Did not find ²	74
Table 6.6	14	90
Table 6.7	4	79

Table 6.9: Hint-based search Vs SA: Number of evaluations to find the first optimal mapping

Both hint-based search and reactive tabu search perform well. The main drawback of hint-based search is if the hints for important operators are discarded, it can fail to find an optimum mapping. RTS suffers in terms of efficiency when trapped in a local maximum. SA, due to its random nature, performs on average more evaluations, and also fails to find an optimal mapping in many cases.

6.2.2 Number of Optimal Mappings Found

There are a number of mappings in our search space that have the optimal evaluation weight. If a search algorithm returns more than one such mapping, the user will have a number of perceptually optimal ways to visualize the data. This can be important, since looking at the same data in different ways often highlights different properties of interest. Thus, the number of optimal mappings found by each search algorithm can also be viewed as a measure of the performance of a search algorithm.

In order to compare each search algorithm in terms of the number of optimal mappings

²However, it reaches 98% of the optimal value.

found, the same set of experiments described in Tables 6.2-6.7 were repeated and the number of optimal mappings found were recorded. For SA, multiple runs were made for each input with the counts being averaged. Table 6.9 shows the results we obtained. As can be seen, RTS is able to find more optimal mappings than both hint-based search and SA.

Experiment	Hint-based search	RTS	SA
Table 6.2	3	16	4
Table 6.3	5	17	4
Table 6.4	3	17	4
Table 6.5	0 ⁴	17	4
Table 6.6	4	6	3
Table 6.7	2	6	4

Table 6.10: Number of optimal mappings found by each algorithm

6.2.3 Space Efficiency

Space efficiency defines the amount of memory or space used by an algorithm. RTS and hint-based search remember information about the states that have been previously visited. Hint-based search also stores hint chains that are generated for each evaluation on a priority queue. Hence, the space requirements of hint-based search are greater than those of RTS. SA, on the other hand, does not store any history and so has the smallest memory requirements of the three algorithms.

⁴However, it reaches 98% of the optimal value.

6.2.4 Optimality

Optimality is defined as the ability to find the best mapping. SA, due to its random nature, cannot be guaranteed to find the optimal. Hint-based search could not find an optimal mapping when the evaluation engines could not generate hints to direct the search towards an optimal. Even though RTS could find an optimal solution in all our experiments, it cannot always be guaranteed to be optimal, since it does not perform an exhaustive search. Hence, none of these algorithms are guaranteed to be optimal. In our experiments, however, RTS found a larger number of optimal solutions and found at least one optimal more often than either hint-based search or SA.

SA, on average, performs worse than hint-based search since it could not find an optimal mapping for about 23% of its runs. RTS shows performance degradation when locked in a local maximum. Both RTS and hint-based search use memory proportional to the number of iterations, whereas, SA does not remember any history. The results obtained from these experiments provide us with possible directions to improve the efficiency of hint-based search.

Chapter 7

Conclusions and Future Work

The importance of visualization is well stated in the phrase, “Seeing is believing”. It strongly suggests the significance of visualization in understanding, analyzing and gaining insight into data. Effective visualization of large, multidimensional data is a known problem. ViA tries to solve this problem, in a domain independent manner, with the use of perceptual guidelines and an intelligent search engine.

Hint-based search is a heuristic algorithm to efficiently search for perceptually salient mappings. The evaluation engines use perceptual guidelines, derived from research on the low level human visual system, to measure the quality of a data-feature mapping. The guidelines are also used to determine favorable operators at each step in the search process. To ensure global motion, hint-based search uses the most promising combinations of operators to make the next move. In order to study the efficiency of the current hint-based search approach, we compared it with two other heuristic algorithms: simulated annealing

and reactive tabu search.

The three algorithms differ in their strategy of choosing the operator to be applied at each step and in the mechanisms used to avoid becoming trapped in a local maximum. SA chooses a random operator at each step and adopts a probabilistic model to escape a local maximum. RTS chooses the best neighbor at each step and avoids returning to recently visited nodes. Each algorithm has its own strengths and weaknesses.

Hint-based search is efficient and quickly reaches approximately 95% of the global optimal, often more quickly than SA and RTS. One of the major drawbacks of hint-based search, revealed by our experiments, is that it can fail to find the optimal mapping if the choice of operators suggested are rejected as producing too much information loss. There is currently no way of avoiding such a situation. SA, due to its random nature takes varying numbers of steps to reach the optimal. The efficiency of SA is better than hint-based search in the scenarios where hint-based search fails locate the optimal solution. In all other cases, hint-based search outperforms SA. Also, SA is not able to find the optimal result in about 23% of its runs. RTS, on the other hand, illustrates good efficiency and is able to find an optimal in all our experiments. One of the major problems with RTS is, due to its exhaustive neighborhood search, large numbers of evaluations are done before it decides to escape a local maximum. This often degrades the efficiency of RTS.

We can conclude that hint-based search performs better than SA, except when important operators are discarded. Both RTS and hint-based search are efficient in their search for

perceptually salient mappings, but have their own weaknesses. The experiments also suggest promising directions for modifying hint-based search in order to increase its efficiency.

Some of the ways in which we can improve the performance of hint-based search are:

- Improvement in the choice of operators might help resolve the failure to locate an optimal solution. For example, one solution could be that each evaluation engine also knows about other engines and takes the requirements of the other engines into consideration when suggesting hints.
- Hint-based search can also be modified to adopt the diversify mechanism of RTS, to escape from a situation where no improvements are found, that is, hint-based search can perform random moves in the search space to escape the region of local maximum.
- Hint-based search can be modified to operate in two phases. In the first phase, the normal generation of hints occurs. When no valid operators are found, hint-based search can switch to the complete neighborhood evaluation phase of RTS to find the best neighbor and ensure forward motion.

To summarize, our analysis helped us better understand the performance of hint-based search and identifies its weaknesses. It also suggests ways to overcome these weaknesses to ensure rapid convergence to effective, perceptually salient visualizations.

Bibliography

- [Bat96] Roberto Battiti. Reactive search: Toward self-tuning heuristics. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search Methods*, pages 61–83. John Wiley & Sons Ltd., Chichester, 1996.
- [BR03] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
- [BT94] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
- [BT95a] R. Battiti and G. Tecchiolli. Local search with memory: Benchmarking rts, 1995.
- [BT95b] Roberto Battiti and Giampietro Tecchiolli. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185–1200, September 1995.
- [CMS99] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. Using vision to think. pages 579–581, 1999.
- [Cox88] H.S.M. Coxeter. *Non-Euclidean Geometry*. The Mathematical Association of America, 1988.
- [Das01] Mehdi Dastani. The role of visual perception in data visualization, November 29 2001.
- [DDB00] Douglas H. Deutschman, Catherine Devine, and Linda A. Buttel. The role of visualization in understanding a complex forest simulation model. *SIG-GRAPH Comput. Graph.*, 34(1):51–55, 2000.
- [Euc56] Euclid. *The Elements*. Dover, 1956. Translated by Sir Thomas L. Heath.
- [FR94] J. Foley and B. Ribarsky. Next generation data visualization tools. *Scientific Visualization Advances and Challenges*, pages chapter7, 103–127, 1994.
- [Gar80] Ann .V.D.L. Gardner. Search: An overview. *AI Magazine*, 2(1):2–6,23, 1980.

- [GHP⁺97] Perry Gray, William Hart, Laura Painton, Cindy Phillips, Mike Trahan, and John Wagner. A survey of global optimization methods. <http://www.cs.sandia.gov/opt/survey/main.html>, 1997.
- [HA99] C. G. Healey and R. St. Amant. Via: A perceptual visualization assistant. In *Proceedings applied imagery pattern recognition workshop '99*, pages 1–11. AIPR, 1999.
- [HE98] C. G. Healey and J. T. Enns. Building perceptual textures to visualize multi-dimensional datasets. In *Proceedings IEEE visualization '98*, pages 111–118. IEEE, 1998.
- [HE99] C. G. Healey and J. T. Enns. Large datasets at a glance: combining textures and colors in scientific visualization. *IEEE transactions on visualization and computer graphics*, 5(2):145–167, 1999.
- [HE02] C. G. Healey and J. T. Enns. Perception and painting: a search for effective, engaging visualizations. *IEEE computer graphics and applications*, 22(2):10–15, 2002.
- [Hea92] Christopher G. Healey. Visualization of multivariate data using preattentive processing, 1992.
- [Hea96] C. G. Healey. Choosing effective colours for data visualization. In *Proceedings IEEE visualization '96*, pages 263–270. IEEE, 1996.
- [HMM00] Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, /2000.
- [HSC01] C. G. Healey, R. St.Amant, and J. Chang. Assisted visualization of e-commerce auction agents. In *Proceedings graphics interface 2001*, pages 201–208. GI, 2001.
- [KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
- [Koc02] Sarat M. Kocherlakota. Perception driven search strategies for effective multi-dimensional visualization. Master’s thesis, North Carolina State University, 2002.
- [LPV03] Lyman, Peter, and Hal R. Varian. How much information? 2003. <http://www.sims.berkeley.edu/how-much-info-2003>, 2003.
- [LR94] John Lamping and Ramana Rao. Laying out and visualizing large trees using a hyperbolic space. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 13–14. ACM Press, 1994.

- [LRP95] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM Press/Addison-Wesley Publishing Co., 1995.
- [Mar75] George E. Martin. *The Foundations of Geometry and the Non-Euclidean Plane*. Springer-Verlag, 1975.
- [MBM87] Defanti McCormick B.H. and T.A. Brown M.D. Visualization in scientific computing. *Computer Graphics*, 21(6), November 1987.
- [Mun97] T. Munzner. H3: laying out large directed graphs in 3d hyperbolic space. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, page 2. IEEE Computer Society, 1997.
- [NGMW95] Jurg Nievergelt, Ralph Gasser, Fabian Maser, and Christoph Wirth. All the needles in a haystack: Can exhaustive search overcome combinatorial chaos? In *Computer Science Today*, pages 254–274. 1995.
- [Nil71] Nils Nilsson. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.
- [RK90] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill Higher Education, 1990.
- [Tai91] E. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.
- [Tuf86] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, 1986.
- [Wol45] Harold E. Wolfe. *Introduction to Non-Euclidean Geometry*. Dryden Press, New York, 1945.