A Decompositional Search Algorithm for Efficient Diagnosis of Multiple Disorders

by

Thomas Dee Wu

Submitted to the Department of
Electrical Engineering and Computer Science
on June 1992, in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Abstract

This thesis develops a new approach to the diagnosis of multiple disorders called decompositional search. Decompositional search finds plausible decompositions of a given problem into subproblems. Each subproblem is represented as a cluster of symptoms, which is explained by a set of disorders called a differential diagnosis. Differential diagnoses are defined by commonality and disjointness constraints that arise from the symptom clusters in a decomposition.

We design and implement an algorithm for decompositional search and compare its efficiency with a non-decompositional diagnostic algorithm called candidate generation. Experimental runs on a large medical knowledge base demonstrate that decompositional search is more efficient than candidate generation by several orders of magnitude. Further analysis reveals that decompositional search gains much of its efficiency by exploiting decompositional structure inherent in the domain. We extend the decompositional search approach to account for probabilistic relationships between symptoms and disorders.

Decompositional search increases the efficiency of diagnostic problem solving and thereby expands our ability to solve problems in complex domains. Problem decompositions also facilitate one's understanding of the structure of a problem and thereby contribute towards more effective decision making.

Thesis Supervisor: Ramesh S. Patil

Acknowledgements

I would like to thank my thesis committee for supervising and guiding this research. Their advice and comments greatly influenced the content and form of this thesis. Randy Davis focused on general, fundamental issues and thereby helped clarify basic ideas in this thesis. Peter Szolovits sought reasons underlying the success of the algorithm and thereby motivated the analytical perspective of this thesis. And Ramesh Patil, my advisor, emphasized the practical aspects of algorithm design and testing and thereby gave this thesis its strong experimental flavor.

The research reported here was supported by National Institutes of Health grant R01 LM04493 from the National Library of Medicine and by National Research Service Award T32 GM07753 from the Department of Health and Human Services. Randolph Miller of the University of Pittsburgh School of Medicine allowed use of the QMR knowledge base for testing purposes.

This work was performed in the Clinical Decision Making Group of the MIT Laboratory for Computer Science. I thank my colleagues in the group for making it such a stimulating and enjoyable place to work. I also thank them for patiently sharing their Lisp machines and SparcStations for many experimental runs.

Finally, this thesis would not have been possible without the support of those outside the laboratory. My friends tolerated long stretches of silence during my thesis research and writing, but still provided encouragement and companionship when I needed it. Most of all, I am indebted to my parents and my brothers, Robert and Perry, for their love throughout.

Contents

Figures

Chapter 1

Introduction

Divide each problem that you examine into as many parts as you can and as you need to solve them more easily.

— René Descartes, *Discours de la Méthode* (1637)

This rule of Descartes is of little use as long as the art of dividing . . . remains unexplained . . . . By dividing his problem into unsuitable parts, the unexperienced problem-solver may increase his difficulty.

— Gottfried Leibniz, *Philosophische Schriften* (c. 1690)

1.1 Overview

As Descartes and Leibniz noted over 300 years ago, a problem can often be simplified by decomposing it into smaller subproblems—when it is decomposed correctly. Today, even with powerful computers, decompositional methods are used extensively to solve complex problems. These methods appear in various forms as parallel programming, dynamic programming, and divide-and-conquer algorithms [8]. However, decompositional techniques are largely limited to problems that have an obvious recursive structure. This prerequisite excludes many problems that are "ill structured", where the task of finding the correct decomposition poses a difficult task in itself [67]. This task—"the art of dividing a problem"—is the subject of this thesis.

Specifically, we develop and analyze a method called *decompositional search* that generates plausible decompositions for a given problem. Decompositional search extends the range of decompositional techniques to cover a broad class of problems, those that form hypotheses to explain a set of evidence. Such problems perform abductive reasoning, or inference to the best explanation [25, 53]. Abductive problems arise in several guises; one typical application is diagnosis in the presence of multiple disorders. In this task, the goal is to explain a given set of evidence in terms of a set of multiple coexisting disorders. The task of multidisorder diagnosis constitutes an important and practical example of abductive problem solving. Accordingly, multidisorder diagnosis serves as a testbed for our development of a decompositional search algorithm.

Our work on decompositional search is motivated by two factors. First, an appropriate problem decomposition can simplify a complex problem, thereby

allowing it to be solved more efficiently. Second, a problem decomposition may be useful end product in itself, giving insight into a problem without necessarily generating its solution. Of these two potential advantages, efficiency and utility, the former is tested more easily. In this thesis, we test efficiency by implementing the decompositional search algorithm in a computer program called SYNOPSIS. With this implementation, we then compare decompositional search with a non-decompositional algorithm called candidate generation. We select this algorithm because of its predominance in the literature and because of its close relationship with decompositional search. Our experimental results, using a large, real-world knowledge base, show that decompositional search is more efficient than candidate generation by several orders of magnitude. Consequently, decompositional search greatly expands our capability to solve problems in complex diagnostic domains.

The second advantage, utility, is not so easily tested by experiment, and so for this claim, we appeal primarily to rhetorical arguments. The utility of decompositional search derives from its novel data structure, called a *problem decomposition*. A problem decomposition structures the given evidence by grouping it into subproblems. In the decompositional search paradigm, diagnosis therefore becomes a search for structure. The structure of a problem can convey important information, especially when there is too little evidence for a definitive hypothesis. Moreover, a problem decomposition represents a set of hypotheses, thereby providing a useful level of abstraction. The search for structure in decompositional search contrasts with the traditional paradigm, where diagnosis is a search for individual hypotheses. This view lacks any notion of structure or abstraction over sets of hypotheses.

To achieve both efficiency and utility, the decompositional search approach relies on structure inherent in the underlying domain. Therefore, we not only develop a decompositional search algorithm but also investigate decompositional structure in diagnostic domains. We describe such domains as having a structure that is *nearly decomposable*, meaning that it consists of several groups of evidence and hypotheses called syndromes. Within syndromes, there are relatively many causal relationships between evidence and hypotheses; between them, there are relatively few. In this thesis, we characterize and quantify this type of domain structure and determine experimentally its role in diagnostic complexity. Our results indicate that decompositional search does indeed exploit domain decomposability to a significant extent.

In this thesis, then, we investigate decomposable structure, both as an

algorithmic device and as a natural phenomenon. As an algorithmic device, decompositional search increases the efficiency of multidisorder diagnosis and enables diagnostic systems to discover and communicate the structure of a problem. As a natural phenomenon, near decomposability improves our understanding of not only the computational complexity of diagnosis but also the nature of the diagnostic task itself.

1.2 The Nature of Diagnosis

To better understand the issues involved in decompositional search, it is useful to look at diagnostic reasoning from a broad perspective. In this section, we examine diagnosis as both a form of abductive reasoning and a type of combinatorial problem solving.

1.2.1 Abductive Reasoning

Diagnosis is a process of reasoning "backwards" about causal events. Such reasoning is backwards because it reverses the natural progression from cause to effect, beginning with the effect and ending up with the most plausible cause. Backwards reasoning can be modeled by abductive inference, which takes the following form:

Tuberculosis can cause fever.
Fever is present.

⇒ (Abduction) Tuberculosis may be present.

Abduction differs from other forms of inference, such as induction and deduction. Induction is the process of hypothesizing a general rule from particular instances [29]. For example, if all patients with tuberculosis have fever, we might infer inductively that tuberculosis causes fever. On the other hand, deduction is the process of making logically correct inferences. In contrast with abduction, deductive inference makes "forward" inferences:

If tuberculosis is present, then fever is present.
Tuberculosis is present.

⇒ (Deduction) Fever is present.

Abduction is closely tied to causality, while deduction is not [5]. In contrast, deduction handles logical implication. For example, suppose that all patients at hospital M have fever. That is,

If patient is in hospital M, then fever is present.

Then if we know that Smith is in hospital M, we can conclude deductively that he has fever. In contrast, abductive inferences do not make sense unless the underlying domain is causal. If we knew that Smith had fever, the backward inference that he is in hospital M would be a poor explanation for fever. Abduction fails because the statement about hospitals and fever is not a causal relationship.

Abduction also depends heavily on notions of plausibility or likelihood. Although probabilistic logics exist, deductive systems in logic and theorem proving generally do not require notions of probability. Thus, causal statements in abduction usually are not deterministic, but indicate only possible causes. For example, in a patient with fever, tuberculosis may not be the actual explanation or even the best one, because there may exist a more likely cause, such as the flu.

Because it is causal and probabilistic, abduction is well suited to many problem-solving tasks. Abduction provides a model not only for diagnosis but also for weighing evidence and making hypotheses, particularly for events that are causal. Because causality is an important organizing principle in our thinking, many problem-solving tasks share features with diagnosis, including applications in expert systems [6, 65], natural language understanding [4, 28], machine learning [42], logic programming [9], and default reasoning [13, 64].

1.2.2 Multiple Disorders

Diagnosis is most challenging when more than one disorder may be present simultaneously. In multidisorder diagnosis, multiple symptoms are allowed, and multiple disorders may plausibly explain them. For example, figure 1-1 shows a case where the flu and common cold together, but not individually, can explain the symptoms of fever and cough. This paradigm contrasts with diagnosis under the single-fault assumption, where only a single disorder is assumed to explain all the symptoms [11]. Explanations that hypothesize one or more coexisting disorders are called *candidates*.

By allowing candidates to express multiple disorders, we expand the power and scope of diagnostic methods beyond the single-fault assumption. Medical diagnosis, for instance, is characterized by patients who often have a combination of acute and chronic diseases. Multidisorder diagnosis also provides a framework to address other issues in diagnosis. For example,

Tuberculosis, the flu, and malaria can cause fever.
Tuberculosis, the common cold, and asthma can cause cough.
Fever and cough are present.

⇒ (Abduction) One of the following statements holds:
- Tuberculosis may be present.
- The flu and the common cold may both be present.
- The flu and asthma may both be present.
- Malaria and the common cold may both be present.
- Malaria and asthma may both be present.

Figure 1-1 An abductive inference with multiple disorders.

multidisorder diagnosis potentially provides a framework for handling false evidence. In such cases, a symptom that is erroneously reported to be true could be explained by a second "disorder" that represents measurement error.

The price for this expanded power and scope is increased computational complexity. The number of possible candidates grows exponentially with the number of disorders under consideration [3]. This computational behavior places a premium on finding efficient algorithms for multidisorder diagnosis, especially if diagnostic programs are to scale up to large, real-world knowledge bases.

1.3 Formalizing Diagnosis

To study multidisorder diagnosis in depth, we move from our general discussion to a formal model. In this section, we provide background information about diagnostic knowledge bases and describe the particular problem that decompositional search addresses. We also describe the candidate generation algorithm that is currently used to solve this problem.

1.3.1 Diagnostic Knowledge Bases

A diagnostic knowledge base represents the domain knowledge needed for diagnostic problem solving. It consists of symptoms and disorders, and the causal relationships between them. A symptom is a piece of evidence about the behavior of a system. Symptoms may express subjective findings as well as objective signs and test results. Evidence is obtained when we assign a

truth value to a symptom: "positive" (or "present"), "negative" (or "absent"), or "unknown". We allow "unknown" symptoms because evidence is almost always incomplete: values of symptoms are frequently unavailable or irrelevant to a given problem. The total sum of evidence constitutes a diagnostic problem, or *case*. A case contains a set of positive symptoms and a set of negative symptoms. All other symptoms are assumed to be unknown.

Just as symptoms are components of evidence, disorders are components of hypotheses. A disorder can assume a wide variety of forms. It can be a faulty component, such as a loose connection or a burned-out light bulb, or it can be a failure mode not connected to any particular component, such as an accidental connection between two separate wires. A disorder might even be an entirely global failure mode, such as an excessively high or low operating temperature that affects the entire system. Finally, a disorder might explain evidence without implicating the system at all, such as attributing it to measurement error. Disorders can express any of these possibilities. All that matters is that a disorder is accepted as a possible explanation by experts in the domain of interest. Of course, what is acceptable depends on the available domain knowledge and may be open to debate, thereby affecting the content of the knowledge base. One might imagine, for instance, that a medical knowledge base earlier in this century might have contained disorders such as "dropsy" (now the symptom edema) and "consumption" (tuberculosis).

These symptoms and disorders are linked by causal relationships, which are also represented in a diagnostic knowledge base. These relationships can be represented as a bipartite graph, with disorders on one side and symptoms on the other. An example of a diagnostic knowledge base is shown in figure 1-2. In this graph, a *causal link* between a disorder and symptom means that the disorder is a possible cause for the symptom. Conversely, the absence of a link indicates that the disorder is not a possible cause for the symptom. Note that causal links specify possibility and not necessity. When a disorder is present, all, some, or none of its possible effects may be positive.

This type of knowledge base is called a *categorical* knowledge base, because it specifies only the presence or absence of causal associations between disorders and symptoms, and not their probabilistic strength [75]. We can, however, generalize a categorical knowledge base to a *probabilistic* knowledge base. In such a knowledge base, each disorder and link has an attached probability, indicating respectively the prior probability of the disorder and the probability of the disorder causing the symptom. In this thesis, we concentrate on developing a theory of decompositional search for the categorical

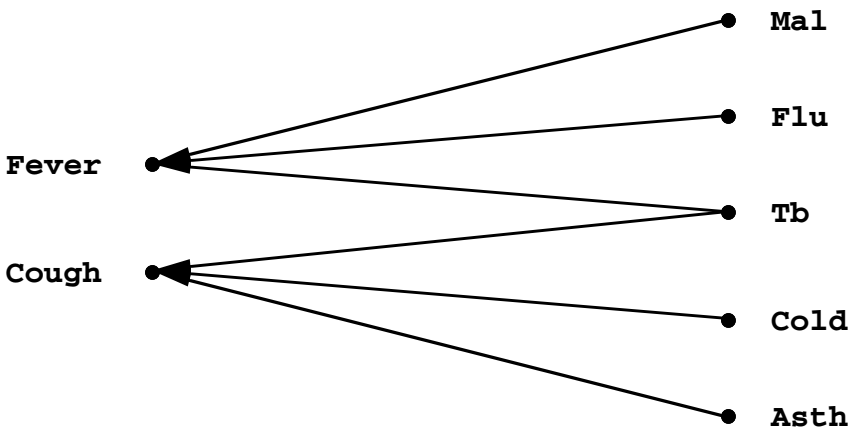

Figure 1-2 A diagnostic knowledge base. Symptoms are shown on the left; disorders on the right. Causal links connect the two sets.

case. However, our work serves as a first step towards a probabilistic theory of decompositional search, and in fact, we will present preliminary results of such a theory.

A diagnostic knowledge base encodes information for a particular *domain*, the set of diagnostic problems for which it is valid. Diagnostic knowledge bases of the appropriate form exist in several domains. For example, the diagnostic program INTERNIST [39], now available as QMR [38], contains a large medical knowledge base. This knowledge base contains information on 600 diseases, or 80 percent of those seen commonly in general internal medicine. The existence of such large knowledge bases enables us to test the ability of diagnostic algorithms to scale up to real-world problems. They also provide additional motivation to develop efficient algorithms that can operate on such diagnostic knowledge bases.

1.3.2 Candidate Generation

A knowledge base constitutes only part of a diagnostic program. In addition, a diagnostic program requires a computational process, or diagnostic algorithm, to make inferences based on the knowledge base. This part of the program is also called an "inference engine" in the expert systems literature [2]. Since the knowledge base contains domain-specific information, the diagnostic algorithm is domain-independent. Thus, work on diagnostic algorithms can be applied to any domain knowledge base of the appropriate

form.

One predominant diagnostic algorithm is *candidate generation* [14, 23, 59]. In candidate generation, the goal is to explain a set of positive symptoms with a conjunction of disorders, or a *candidate*. A set of disorders is a candidate if it explains every positive symptom in a given case. In other words, for every positive symptom in the case, a possible cause for it must exist in the proposed candidate. We represent candidates in square brackets, e.g., [`Flu`,`Cold`], meaning that the symptoms `Flu` and `Cold` are present.

Note that in defining a candidate we have ignored negative symptoms. This is because causal links express possible causation, not necessary causation. Hence we cannot rule out any disorder based on a negative symptom. However, negative symptoms may make a candidate less likely. Thus, in this paradigm, negative symptoms exert their role by modifying the probability of a candidate, once it is generated. We will consider the role of negative symptoms in our chapter on probabilistic decompositional search.

Diagnostic algorithms such as candidate generation seek those explanations that are most plausible. In domains where probabilistic information is available, the most plausible explanation is that which has the highest conditional probability, given a particular case. But in categorical domains, diagnostic algorithms require *plausibility criteria* instead to select the best explanations [54]. Most recent diagnostic algorithms use *minimality* as their plausibility criterion. A candidate is minimal when none of its subsets explains all of the positive symptoms. If the symptoms `Fever` and `Cough` are both positive, then the candidate [`Flu`,`Cold`] is minimal because neither [`Flu`] nor [`Cold`] sufficiently explain both symptoms. Note that a candidate makes no hypothesis about disorders not in the candidate. The status of these other disorders is essentially "unknown". Thus, a single symptom may have more than one cause, including disorders within or outside of the candidate.

Other plausibility criteria besides minimality exist. One criterion that is stronger than minimality is *minimal cardinality*. A candidate has minimal cardinality when it explains the given symptoms using the fewest possible disorders. For `Fever` and `Cough`, the single-disorder candidate [`Tb`] is of minimal cardinality, meaning that all two-disorder candidates, such as [`Flu`,`Cold`], are not. Although minimal cardinality greatly reduces the number of "plausible" candidates, most researchers consider this criterion too restrictive. For example, a single rare disorder may exist that potentially explains the given symptoms, but two or more common disorders may be more likely causes. So a single-disorder candidate may indeed be less plausible than a multidisorder

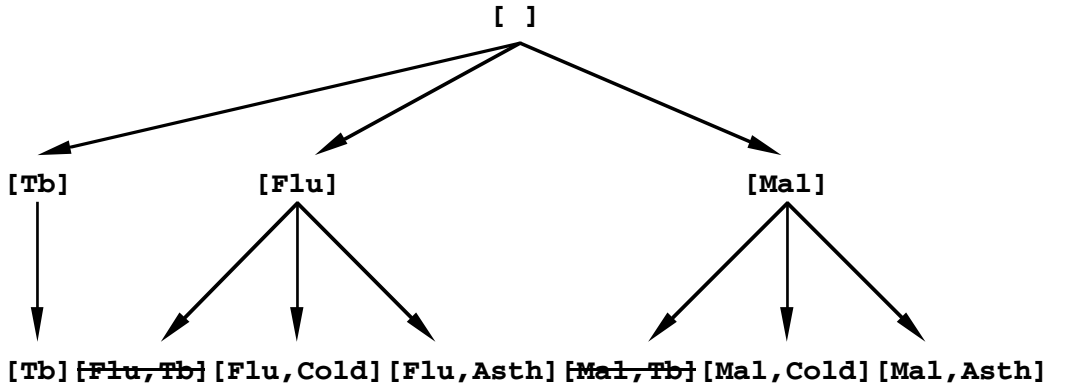

Figure 1-3 Example of candidate generation. The first row shows the diagnosis of `Fever`; the second row, of `Fever` and `Cough`. Nonminimal candidates are shown crossed out.

candidate. For instance, tuberculosis may explain both a fever and cough, but it is much less common than a coexisting flu and cold.

Plausibility criteria, like minimality, identify a set of plausible candidates. The predominant algorithm for producing minimal candidates is candidate generation. This technique forms the basis of both model-based diagnosis [12] and set-covering diagnosis [58]. An example of candidate generation is shown in figure 1-3. This example uses the knowledge base in figure 1-2 to solve the case where symptoms `Fever` and `Cough` are both positive. The search tree begins by explaining `Fever`, by creating the three candidates [`Tb`], [`Flu`], and [`Mal`]. Each candidate corresponds to a possible cause for `Fever`. Each minimal candidate is then tested for its ability to explain the next symptom, `Cough`. Candidate [`Tb`] already explains `Cough`, so it is kept in the search tree unaltered. Candidate [`Flu`], however, does not explain `Cough`, so it must be expanded by the possible causes for `Cough`, namely, disorders `Tb`, `Cold`, and `Asth`. This yields the two-disorder candidates [`Flu`,`Tb`], [`Flu`,`Cold`], and [`Flu`,`Asth`]. However, candidate [`Flu`,`Tb`] is nonminimal and is pruned because disorder `Tb` already explains both `Fever` and `Cough`. Similarly, candidate [`Mal`] does not explain `Cough` either, so it is expanded by the possible causes for `Cough` to generate [`Mal`,`Tb`], [`Mal`,`Cold`], and [`Mal`,`Asth`]. Candidate [`Mal`,`Tb`] is pruned because it is nonminimal. The end result is the five minimal candidates [`Tb`], [`Flu`,`Cold`], [`Flu`,`Asth`], [`Mal`,`Cold`], and [`Mal`,`Asth`].

Note that much of the reasoning in candidate generation is redundant. Candidates [`Flu`] and [`Mal`] are expanded for the same reason: they explain only `Fever`. In addition, both two-disorder candidates containing `Tb` are pruned for the same reason: `Tb` explains both `Fever` and `Cough`. The two-

disorder candidates that are not pruned are those that contain one disorder that explains `Fever` but not `Cough` (i.e., `Flu` or `Mal`) and one that explains `Cough` but not `Fever` (i.e., `Cold` or `Asth`).

This redundancy suggests that a more efficient method may be possible. The amount of redundant reasoning might be reduced if we could group candidates based on the rationale by which they explain the symptoms. Then decisions about whether to expand or prune could be applied to the entire group of candidates, rather than having to be computed for each candidate individually.

1.4 Near Decomposability

In computer science, redundancy is a sign of structure. When a task is iterative, it suggests an underlying list structure; when a task is recursive, it suggests an underlying recursive structure. Likewise, the redundancy observed in candidate generation suggests a type of structure that is decomposable. In this section, our attention turns to whether decomposable structure can be exploited to reduce the complexity of diagnosis.

1.4.1 Redundancy, Structure, and Complexity

Searching for minimal candidates is inherently computationally complex. Finding the set of minimal candidates belongs to a class of problems that is called NP-hard [3, 19]. Essentially, this means that there exists a worst-case series of problem instances for which all known algorithms behave poorly, requiring time that is exponential in the size of the problem. Thus, it is unlikely that we could design an algorithm for computing minimal candidates that is efficient in the worst-case.

However, we are not particularly interested in worst-case analyses, especially in an empirically-based task like diagnosis, where the theoretical worst case probably never arises. A diagnostic problem is not intended to push a diagnostic system to its computational limits. Rather, a diagnostic problem comes from a real-world domain knowledge base and a particular case, both of which are shaped fundamentally by the particular system being diagnosed.

Computational complexity theory, which is typically concerned with the worst case, offers little to say about such naturally occurring instances. In such instances, the performance of two diagnostic algorithms may dif-

fer markedly, even though their performance in the asymptotic worst case remains the same. Thus, we now ask whether a diagnostic algorithm can exploit structural features of diagnostic domains and cases to increase efficiency.

1.4.2 The Structure of Complex Systems

Since structure in diagnostic knowledge bases derives from the systems being diagnosed, structure must be present in the systems themselves. Although we must be careful not to overgeneralize, complex systems typically share a number of features. The feature that we identify and exploit in this thesis is the fact that complex systems are often *nearly decomposable*. The notion of near decomposability derives from Herbert Simon, who describes it as follows [68, p. 209–210]:

> In hierarchic systems we can distinguish between the interactions *among* subsystems, on the one hand, and the interactions *within* subsystems—that is, among the parts of those subsystems—on the other. The interactions at the different levels may be, and often will be, of different orders of magnitude. In a formal organization there will generally be more interaction, on the average, between two employees who are members of the same department than between two employees from different departments. In organic substances intermolecular forces will generally be weaker than molecular forces, and molecular forces weaker than nuclear forces.
>
> In a rare gas the intermolecular forces will be negligible compared to those binding the molecules—we can treat the individual particles for many purposes as if they were independent of each other. We can describe such a system as *decomposable* into the subsystems comprised of the individual particles .... As a second approximation we may move to a theory of *nearly decomposable* systems, in which the interactions among the subsystems are weak but not negligible.

In other words, near decomposability means that complex systems can often be subdivided into subsystems, where the behaviors of subsystems are independent or only weakly dependent on one another.

There are various explanations for why near decomposability should exist. Simon proposes that nearly decomposable systems are more stable and hence more likely to develop by evolution or by design [68]. But regardless of the reason, it appears that many diagnostic domains are indeed nearly decomposable. Automobiles are composed of subsystems for ignition, steering, power transmission, braking, and so on. In medical diagnosis, the human body is divided into organ systems, such as those for the cardiovascular (heart), pulmonary (lung), and renal (kidney) systems. Note that these subdivisions are not completely decomposable, only nearly so. For example, many pulmonary diseases cause shortness of breath, but so do many cardiovascular diseases. Some diseases, such as systemic infections, affect multiple organ systems. Nevertheless, near decomposability remains an important and widespread organizing principle for complex systems.

1.4.3 Syndromic Structure in Knowledge Bases

To see how near decomposability in a system might manifest in a diagnostic knowledge base, let us examine figure 1-4. The overall appearance of the knowledge base, shown in figure 1-4(a), exhibits little structure. But the structure becomes apparent if the knowledge base is represented as in figure 1-4(b). This representation is equivalent, containing all of the links that are present in the original knowledge base. However, the links are organized into *syndromes*, where a syndrome is composed of a conjunction of symptoms and a disjunction of disorders. In a syndrome, any one of the disorders could potentially explain all of the symptoms. The knowledge base can therefore be represented as two syndromes instead of 17 causal links.

The two syndromes in this example are not completely decomposable. They overlap in both symptoms and disorders, namely, at symptom s_4 and disorder d_3. But they are nearly decomposable to the extent that causal links within each syndrome are relatively dense and between each syndrome are relatively sparse. Thus, the syndromes interact relatively weakly, and indicate that the knowledge base does indeed possess nearly decompositional structure.

Since causal relationships are complex, there may be several ways to decompose a knowledge base. Figure 1-4(c) shows an alternative representation that contains only a single syndrome, instead of two. This syndrome is possible because there exists a single disorder, namely d_3, that can explain all of the symptoms. This syndrome is not equivalent to the entire knowledge

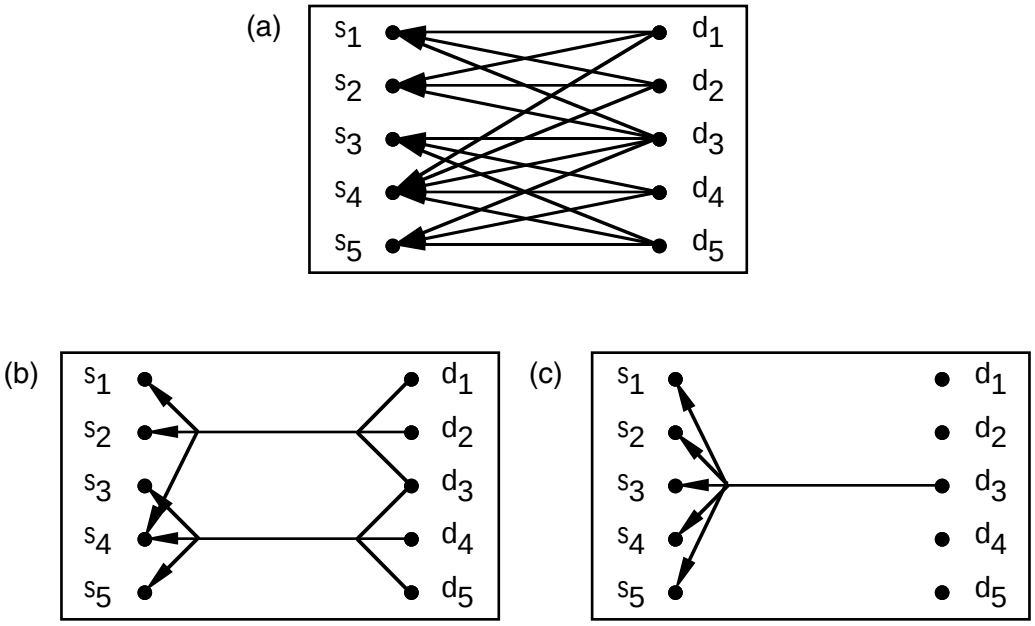

Figure 1-4 Near decomposability in a diagnostic knowledge base. (a) Original knowledge base. (b) An equivalent representation with two syndromes. (c) An alternate decomposition with one syndrome.

base, since not all of the links are represented. However, it does express an important organizing idea, namely, that all five symptoms share a common possible cause.

The existence of several possible decompositions might be expected in a real-world knowledge base. This is because complex systems are often constructed hierarchically, with subsystems at various levels of abstraction. A low-level failure might cause only a few specific symptoms. On the other hand, a high-level failure might cause several nonspecific symptoms. Levels of abstraction are reflected in the knowledge base, which contains symptoms, disorders, and causal links of varying specificity. A medical knowledge base might have general, high-level disorders, such as "infection", as well as specific, low-level disorders, such as "abscess". Likewise, symptoms may range from the general, such as "malaise", to the specific, such as "elevated white cell count". Another reason for multiple decompositions is that domains are often structured according to multiple criteria. For instance, medical diseases are defined not only on the basis of organ system, but also on the type of pathophysiologic process, such as infections, neoplasms, metabolic defects, and so on. Consequently, there may be several ways to decompose a diagnostic knowledge base, any one of which might be useful in a particular case.

1.5 Decompositional Search

1.5.1 Clusters and Differential Diagnoses

We have seen that near decomposability is apparently characteristic of diagnostic domains and their associated knowledge bases. By extension, we hypothesize that diagnostic cases should be decomposable to the same extent. The decompositional structure in a case can may help reduce the complexity of search. If diagnostic cases exhibit nearly decomposable structure, then the evidence should should yield one or more plausible decompositions. The idea of case-specific problem decomposition is the basis of the decompositional search approach.

A case-specific decomposition is called a *problem decomposition*. A problem decomposition assigns the positive symptoms in a case to separate *clusters*. The structure of a decomposition appears in figure 1-5. We represent a decomposition using parentheses, each pair of parentheses enclos-

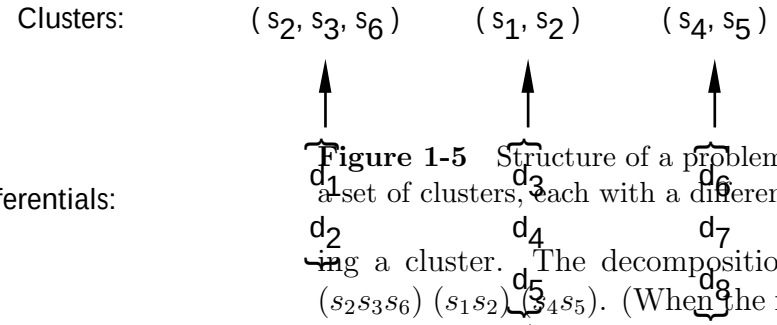

Figure 1-5 Structure of a problem decomposition. A problem decomposition is a set of clusters, each with a differential diagnosis.

ing a cluster. The decomposition shown in the figure can be written as $(s_2s_3s_6)$ (s_1s_2) (s_4s_5). (When the notation is clear, we will eliminate commas within clusters.) A symptom may be in more than one cluster, so clusters may overlap; however, by definition, each cluster must have at least one symptom not in any other cluster. Associated with each cluster is a *differential diagnosis* (or differential, for short), a disjunctive set of disorders that can explain all symptoms in that cluster. We represent differential diagnoses by braces. In figure 1-5, $\{d_1,\ d_2\}$ is the differential diagnosis for cluster $(s_2s_3s_6)$. This means that either d_1 or d_2 is a possible explanation for symptoms s_2, s_3, and s_6.

The cluster and differential diagnosis together comprise a *task*. Thus, a task contains both a subproblem and a set of solutions to that subproblem. A task is similar to a syndrome, except that a syndrome is a decomposable subunit of a knowledge base, while a task is a decomposable subunit of a diagnostic case. We refer to the differential corresponding to a cluster as its *associated* differential, and to differentials that explain other clusters as *adjacent* differentials.

1.5.2 Decompositions as Constraints

The clusters of a problem decomposition represent a set of assumptions, or constraints, about the way the evidence can be explained. Symptoms in the same cluster are assumed to be caused by the same disorder, while symptoms in different clusters are assumed to be caused by different disorders. Thus, the structure of a problem decomposition represents a set of *commonality* and *disjointness constraints*. The commonality constraint states that all symptoms in a cluster must be explainable by a single disorder. The disjointness constraint states essentially that each cluster must be explainable by a "unique" disorder that does not explain another cluster. If these constraints can be satisfied, the decomposition is *coherent*; otherwise, it is *incoherent*.

The commonality and disjointness constraints define the differential diagnoses for the decomposition. The process of computing differential diagnoses

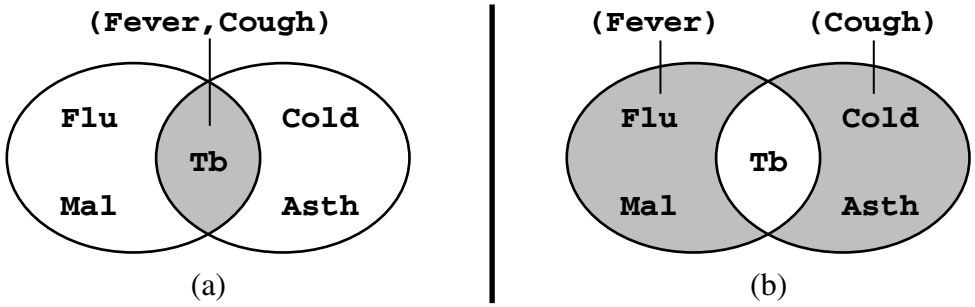

Figure 1-6 The concept of differential formulation. Differential diagnoses are shown as shaded areas. (a) Differential diagnosis resulting from the decomposition (`Fever`,`Cough`). (b) Differential diagnoses resulting from the decomposition (`Fever`) (`Cough`).

is called *differential formulation*; the basic process is illustrated in figure 1-6. Here we have a Venn diagram of possible causes for symptoms. The possible causes for `Fever` are `Tb`, `Flu`, and `Mal`; the possible causes for `Cough` are `Tb`, `Cold`, and `Asth`. With two symptoms, only two problem decompositions are possible: (`Fever`,`Cough`) and (`Fever`) (`Cough`). For (`Fever`,`Cough`), the commonality constraint means that the differential diagnosis contains only `Tb`, the only explanation common to both symptoms. This situation is shown in figure 1-6(a). On the other hand, for (`Fever`) (`Cough`), the commonality constraint allows `Tb`, `Flu`, and `Mal` for the first cluster, and `Tb`, `Cold`, and `Asth` for the second cluster. This situation is shown in figure 1-6(b). The disjointness constraint removes `Tb` from both differentials because it can explain both clusters. `Tb` is removed from this decomposition because it is a unifying explanation for both clusters, which argues against the existence of two separate clusters.

Symptoms in a cluster therefore perform two functions: (1) they help define their associated differential via a commonality constraint, and (2) they help define adjacent differentials via disjointness constraints. The commonality constraint is satisfied by taking the intersection of the possible causes for each symptom in the cluster. Only those disorders can be in the differential diagnosis.

The disjointness constraint is slightly more complicated. It is satisfied by defining a *justifying set* for each cluster, a subset of symptoms in the cluster that disorders in adjacent differentials are constrained not to explain. We define a symptom as justifying its cluster if, for every adjacent differential, it cannot be explained by all disorders in that differential. The intuition

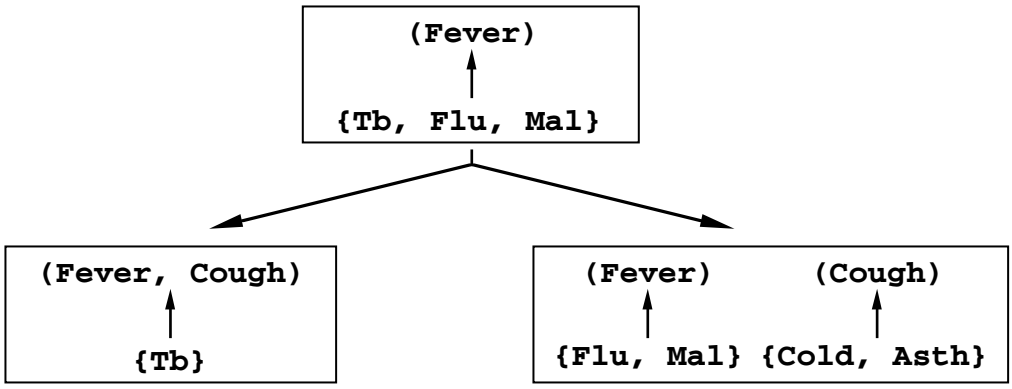

Figure 1-7 Example of decompositional search. The first row shows the diagnosis of `Fever`; the second row, of `Fever` and `Cough`.

behind this definition is that in order for a separate cluster to be warranted, it must contain some symptom that cannot be explained by other clusters. With this definition, we can now explain the disjointness constraint fully. The disjointness constraint states that disorders in a differential cannot explain the justifying set in any adjacent cluster. Note that the definition of differential is recursive. A differential is defined by the justifying symptoms in adjacent clusters, and the justifying symptoms in a cluster are defined by the differentials in adjacent differentials. Nevertheless, as we shall see in this thesis, this recursive description is well defined and can be computed.

The commonality and disjointness constraints result in a set of differential diagnoses. For disorders not in a differential diagnosis, a problem decomposition makes no hypothesis about their presence or absence. It only hypothesizes the minimum disorders necessary to explain the symptoms, and other disorders may in fact be present. A problem decomposition also makes no hypothesis about causal relationships other than between a differential diagnosis and its associated cluster, so that a disorder that explains one cluster may also explain some symptoms in another cluster.

Problem decompositions are synthesized by a search process. An example of decompositional search for the previous example of fever and cough is shown in figure 1-7. In this example, the first symptom, `Fever`, can only be decomposed in one way. This yields the differential {`Tb`, `Flu`, `Mal`}. The next symptom, `Cough`, can either be placed in the same cluster as `Fever` or be assigned to its own cluster. If `Fever` and `Cough` are in the same cluster, the differential is {`Tb`}. If `Fever` and `Cough` are in different clusters, the

differentials for them are {`Flu`, `Mal`} and {`Cold`, `Asth`}, respectively. In this search tree, both decompositions are coherent. But some decompositions might not be coherent, because they posit a set of commonality and disjointness constraints that cannot be satisfied. Incoherency is signaled by one or more differential diagnoses that are empty, meaning that no disorders could meet the constraints. Such decompositions are pruned from the search tree.

1.6 Features of Decompositional Search

Since decompositional search is a new approach to diagnosis, it introduces some new concepts to problem solving and representation. We now present some of the major features of decompositional approach. Other features are discussed in the final chapter.

1.6.1 Cartesian Product Representation

A problem decomposition represents a set of candidates in a compact *Cartesian product representation*. A problem decomposition hypothesizes the presence of at least one disorder in each differential. The set of disorders so chosen explain the positive symptoms and is therefore a candidate. The total set of candidates that can be chosen is equivalent to the Cartesian product of the differentials. However, we interpret the answers as unordered sets rather than ordered sets. For example, the decomposition in figure 1-5 represents the following set of candidates:

$$\{d_1,\ d_2\} \times \{d_3,\ d_4,\ d_5\} \times \{d_6,\ d_7,\ d_8\} =$$
$$\begin{array}{lll} [d_1, d_3, d_6], & [d_1, d_3, d_7], & [d_1, d_3, d_8], \\ [d_1, d_4, d_6], & [d_1, d_4, d_7], & [d_1, d_4, d_8], \\ [d_1, d_5, d_6], & [d_1, d_5, d_7], & [d_1, d_5, d_8], \\ [d_2, d_3, d_6], & [d_2, d_3, d_7], & [d_2, d_3, d_8], \\ [d_2, d_4, d_6], & [d_2, d_4, d_7], & [d_2, d_4, d_8], \\ [d_2, d_5, d_6], & [d_2, d_5, d_7], & [d_2, d_5, d_8] \end{array}$$

We refer to the left side of the equation as the implicit or Cartesian product representation and to the right side as an explicit representation. The explicit representation lists each candidate individually, while the implicit one represents a set of candidates in a Cartesian product representation. Of the two representations, the implicit one is more compact, requiring only

space proportional to the sum of the differential sizes. In this case, the space required is $2+3+3=8$ disorders. The explicit representation, on the other hand, requires space proportional to the product of the differential sizes. The space required is $2\times3\times3=18$ candidates, each of which contains 3 disorders.

By generating and transforming sets of candidates in implicit representation, decompositional search performs diagnosis at a more abstract level than candidate generation. In candidate generation, each node in the search tree represents an individual candidate. On the other hand, in decompositional search, each node defines a set of candidates, defined by the commonality and disjointness constraints of the problem decomposition. Thus, diagnosis in decompositional search is a process of manipulating constraints. The constraints are posted and tested without having to compute the candidates explicitly. The abstract strategy of manipulating constraints on candidates, rather than the candidates themselves, enables decompositional search to have a smaller search space and greater efficiency.

1.6.2 Causal Equivalence

A problem decomposition groups disorders into differential diagnoses. These groupings essentially categorize disorders according to their ability to explain the symptoms. For example, consider the decompositions in figure 1-5. `Tb` is in one differential because it is the only disorder that can explain both `Fever` and `Cough`. `Flu` and `Mal` are in the same differential because they can explain `Fever` but not `Cough`. And `Mal` and `Asth` are in the same differential because they can explain `Cough` but not `Fever`. We say that disorders that explain the same subset of symptoms are *causally equivalent*.

Grouping by causal equivalence makes the search process less redundant. We have seen redundant reasoning previously in our example of candidate generation (figure 1-3). There, candidate generation could not detect that the disorders `Flu` and `Mal` are causally equivalent. Thus, it duplicates the same expansion and pruning sequence for each disorder. Likewise, candidate generation cannot determine that [`Flu`,`Cold`], [`Flu`,`Asth`], [`Mal`,`Cold`], and [`Mal`,`Asth`] all have the same causal structure, explaining `Fever` and `Cough` using the same configuration of causal links. In contrast, decompositional search groups all of these candidates together. By grouping disorders that are causally equivalent, decompositional search is able to make decisions for a whole set of candidates at once and thereby avoids redundancy in its search process.

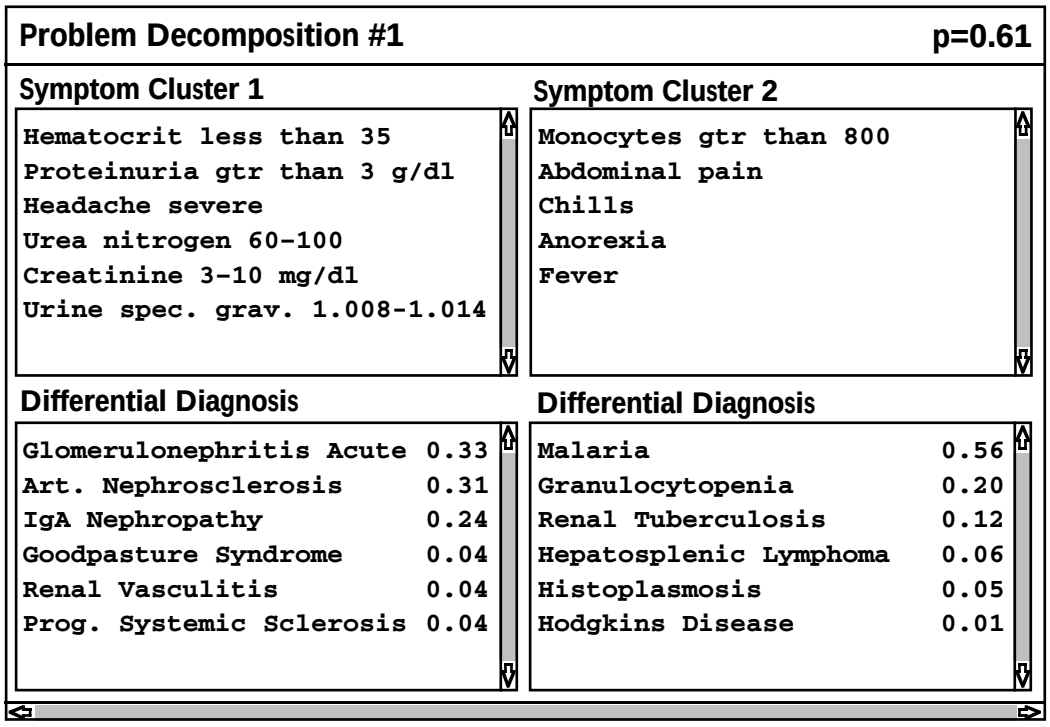

Figure 1-8 Output of a decompositional search system. This figure illustrates a possible interface.

The notion of causal equivalence may not only facilitate efficiency, but may also provide a potentially useful mode of problem solving. A problem decomposition represents a possible structuring of the evidence and solutions. Such a structuring may help a user understand the structure of a problem. The output of a decompositional system might look something like figure 1-8. This example contains two clusters, each of which has a separate differential diagnosis. The first cluster contains symptoms and disorders related to a renal disease, while the second contains symptoms and disorders related to a separate infectious process. This structuring of the problem may facilitate the user's understanding of a diagnostic situation, more so than a listing of plausible candidates. Thus, grouping by causal equivalence may be just as important to our comprehension of a problem as it is to the computation of an algorithm.

1.7 Guide to the Thesis

This thesis develops the decompositional search approach through theory, experiment, and analysis. Chapter 2 presents an extended example of decompositional search. Chapters 3 and 4 develop the theory behind decompositional search: first the "static" theory of problem decompositions, then the "dynamic" theory of generating decompositions by a search process. These chapters present a revised and expanded version of the ideas presented originally in [80, 81]. Given this theoretical background, we test the efficiency of decompositional search empirically in chapter 5. These experimental results are an expanded version of the results presented in [81, 84]. These results are then analyzed in chapter 6, with particular attention to the role that domain structure plays in diagnostic complexity. The analytical results in this chapter extend the results of [82]. Chapter 7 extends decompositional search to the probabilistic case. This chapter is an improved presentation of the results in [83]. In the concluding chapter, we summarize our major findings and place this work in perspective.

Chapter 2

Example

> *Most physicians attempt consciously or unconsciously to fit a given problem into one of a series of syndromes. The syndrome is a group of symptoms and signs of disordered function, related to one another by means of some anatomic, physiologic, or biochemical peculiarity .... The diagnosis is greatly simplified if a clinical problem conforms neatly to a well-defined syndrome, because only a few diseases need be considered in the differential diagnosis.*
>
> — *Harrison's Principles of Internal Medicine, 12th ed.* (1991)

Decompositional strategies are often used in the real world. As Harrison's textbook of medicine [78, p. 3] observes, physicians often formulate plausible symptom clusters and match them to known syndromes. In this chapter, we consider a simple example, which also happens to be from clinical medicine. This example uses a portion of the QMR knowledge base that has been simplified to help contrast candidate generation and decompositional search.

2.1 The Problem

The knowledge base for this example, shown in figure 2-1, is taken from the area of renal medicine, dealing with diseases of the kidney. It contains 4 symptoms, 16 disorders, and 32 causal links. The causal links are shown in a table, which is representationally equivalent to a bipartite graph.

The symptom `Cr` indicates that the patient's bloodstream has a high level of creatinine, which is normally excreted by the kidneys. The symptom `Bun` corroborates this finding, showing that the patient's bloodstream has a high level of urea, which should also be excreted by the kidneys. The symptom `Dehyd` states that the patient is dehydrated, perhaps because the kidneys are losing too much water or because the patient is not drinking enough. Finally, the symptom `Uo` indicates that the patient's urine output is abnormally low, again suggesting some kidney dysfunction. In the table of causal links, an 'X' for a disease-symptom pair means that the disease is a possible cause for the symptom. There are two diseases, `Atn` and `Pra`, that can each explain all the symptoms, while the remaining 14 diseases can each explain only some of the symptoms.

Symptoms:

Cr	Serum creatinine 3 to 10 mg/dl
Bun	Blood urea nitrogen 30 to 59
Dehyd	Dehydration
Uo	Urine output less than 400 ml/day

Disorders:

Atn	Acute tubular necrosis
Pra	Prerenal azotemia
Lit	Nephrolithiasis
Rcc	Renal cell carcinoma
Pn	Acute pyelonephritis
Ano	Anorexia nervosa
Dm	Diabetes mellitus
Mal	Malaria
Pan	Pancreatitis
Vol	Hypovolemic shock
Prt	Portal hypertension
Anl	Analgesic nephropathy
Agn	Acute glomerulonephritis
Sln	Salt-losing nephritis
Tb	Renal tuberculosis
Rta	Renal tubular acidosis

Links:

	Atn	Pra	Lit	Rcc	Pn	Ano	Dm	Mal
Cr	X	X						
Bun	X	X			X			
Dehyd	X	X				X	X	X
Uo	X	X	X	X	X			

	Pan	Vol	Prt	Anl	Agn	Sln	Tb	Rta
Cr				X	X		X	
Bun		X	X	X	X	X	X	X
Dehyd	X	X	X	X		X		
Uo					X	X		

Figure 2-1 Example knowledge base. Symptoms and disorders are listed in both abbreviated and full forms. Causal links are represented in tabular form.

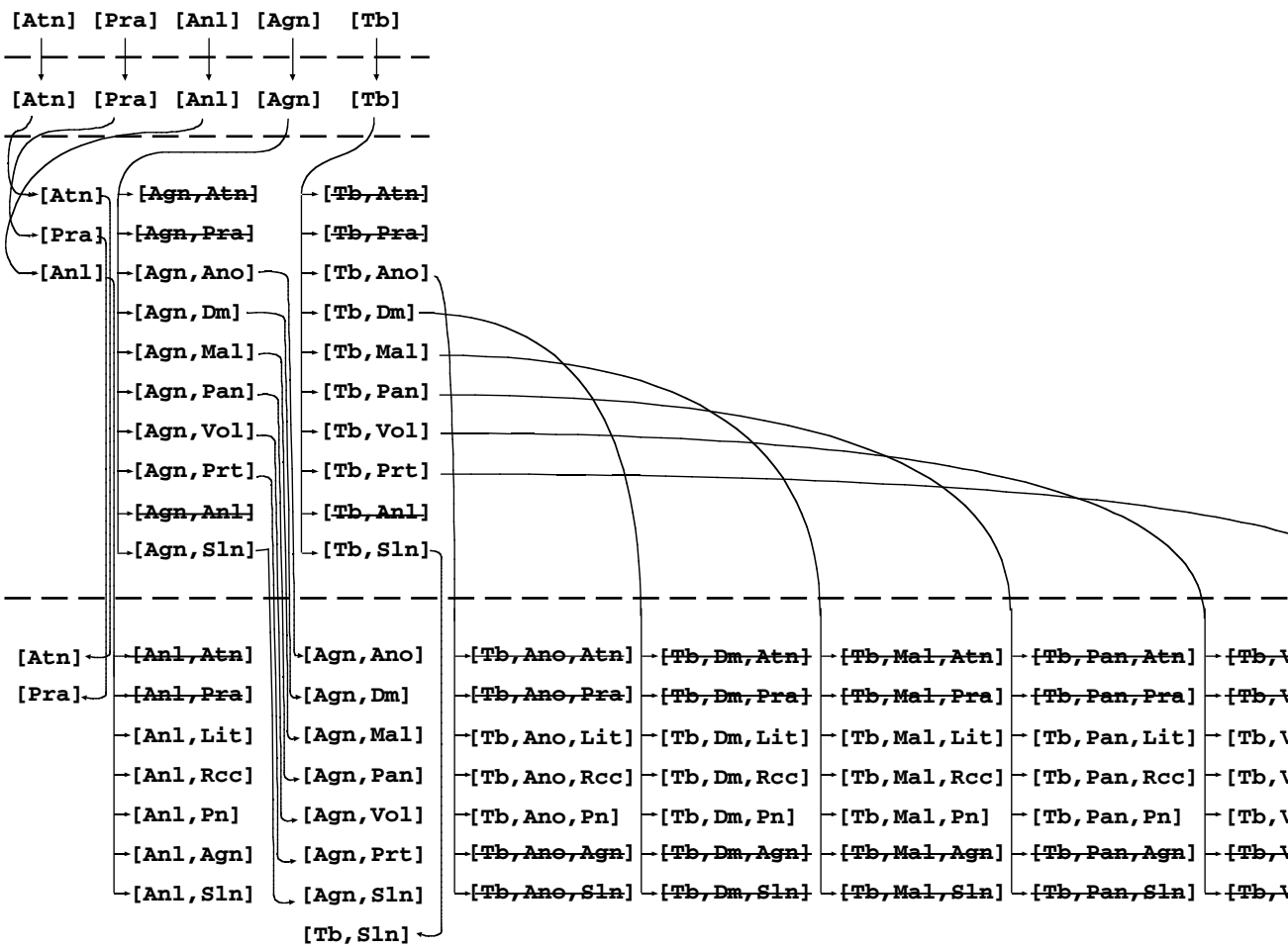

Figure 2-2 Candidate generation search tree for example. Dashed lines separate each frontier of intermediate candidates. Arrows indicate the expansion of intermediate candidate. Nonminimal candidates are shown crossed out.

2.2 Candidate Generation

The four symptoms can be solved by the candidate generation method, as shown in figure 2-2. The end result of this method is the 33 minimal candidates shown on the bottom row of the figure. Intermediate results, shown on each preceding row, derive from considering each symptom sequentially. Each intermediate node contains a minimal candidate that explains the symptoms considered so far. If a candidate already explains the new symptom, the candidate is kept as is. Otherwise, the candidate is expanded by creating a new set of candidates, one for each possible cause of the new symptom. Each

of these newly expanded candidates is then tested for minimality and pruned if nonminimal.

For example, after symptoms Cr and Bun are processed, there are 5 minimal candidates: [Atn], [Pra], [Anl], [Agn], and [Tb]. The set of minimal candidates does not change between the first and second symptoms. This is because Cr is a more specific symptom than Bun. Only a few diseases can give a high creatinine level, whereas many additional diseases can cause a high urea nitrogen level. Thus, we say that Bun is more general than Cr, or conversely, Cr is more specific than Bun.

The third symptom, Dehyd, illustrates redundancy in candidate generation. The candidates [Atn], [Pra], and [Anl] are not expanded, but kept as one-disorder candidates. This is because the disorders Atn, Pra, and Anl are each able to explain all three symptoms. However, the candidates [Agn] and [Tb] are expanded, because they explain Cr and Bun but not Dehyd. They are each expanded by the 10 possible causes for Dehyd, of which 3 are pruned. The remaining candidates derived from [Agn] and those derived from [Tb] are minimal. Thus, there are only two patterns of candidates at this point: (1) single-disorder candidates, and (2) two-disorder candidates containing either Agn or Tb, plus one of the disorders explaining Dehyd alone.

On the fourth symptom, Uo, candidate generation also exhibits redundancy. The candidates [Atn] and [Pra] explain all four symptoms, so they are kept unchanged. The candidate [Anl], however, does not explain Uo, so it is expanded by the 7 possible causes for Uo. The next 8 candidates all explain Uo, so they remain unchanged. Finally, the next 6 candidates all fail to explain Uo, so they are expanded by the 7 possible causes for Uo. For each of these candidates, only those containing Lit, Rcc, or Pn are minimal. Those containing Atn, Pra, Agn, or Sln are pruned.

2.3 Decompositional Search

In decompositional search, the symptoms in our example can be explained by five coherent decompositions, ranging from one to three clusters:

One cluster	Two clusters	Three clusters
(Cr,Bun,Dehyd,Uo)	(Cr,Bun,Dehyd) (Uo)	(Cr,Bun) (Dehyd) (Uo)
	(Cr,Bun,Uo) (Dehyd)	
	(Cr,Bun) (Bun,Dehyd,Uo)	

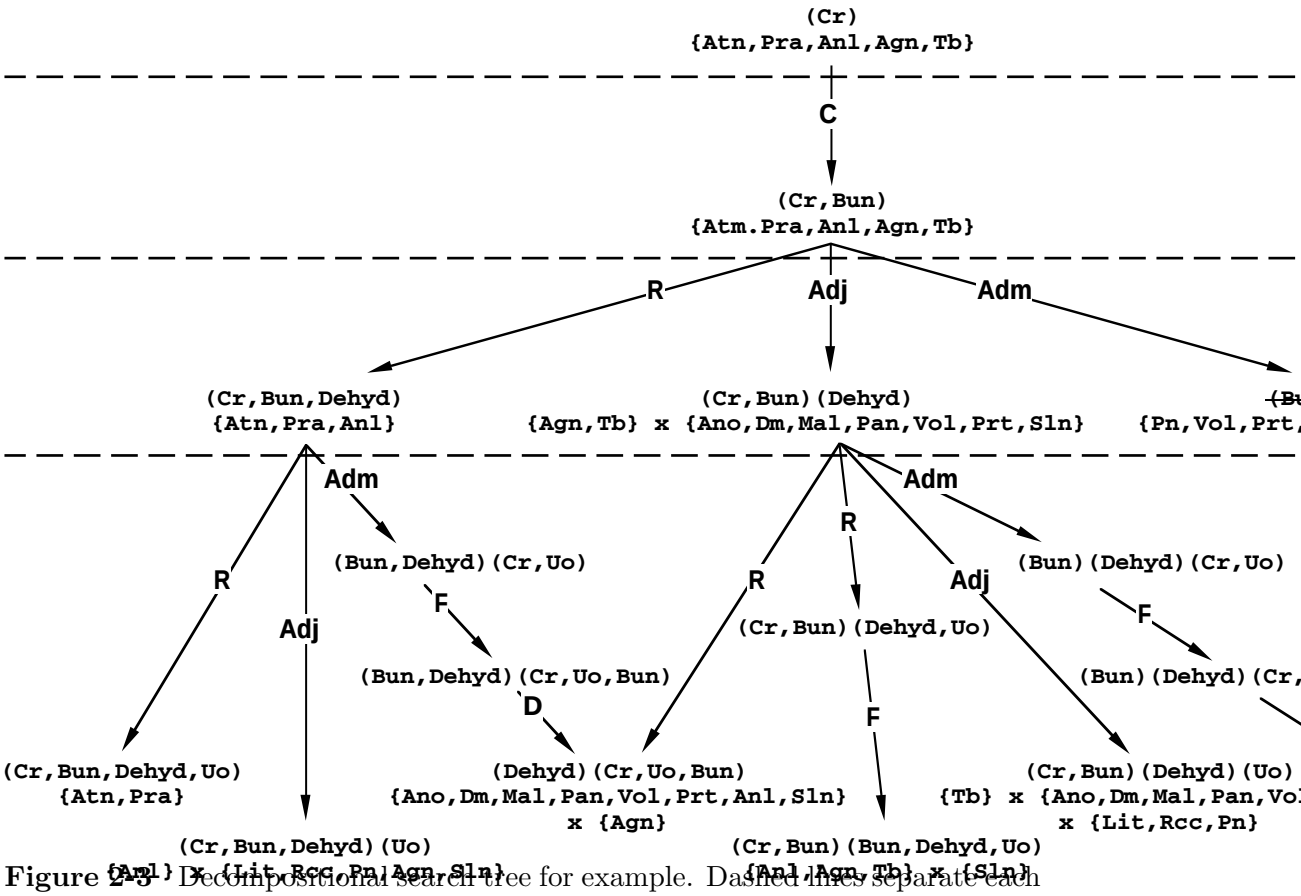

Figure 2-3 Decompositional search tree for example. Dashed lines separate each frontier of intermediate decompositions. Arrows indicate expansion of decompositions. Arrows are labeled according to the operator applied: C = covering, R = restricting, Adj = adjoining, Adm = admixing, F = forward ambiguation, B = backward ambiguation (not shown), and D = disambiguation. Incoherent decompositions are shown crossed out.

The search process that generates these decompositions is shown in figure 2-3. We now trace the steps in this search process.

Symptom 1: High Serum Creatinine

The first symptom, `Cr`, can be decomposed in only one way, leading to the following problem decomposition:

Decomposition 1a:

Cluster	Common Causes	Differential
(Cr)	Atn, Pra, Anl, Agn, Tb	Atn, Pra, Anl, Agn, Tb

This decomposition contains one cluster and its corresponding differential diagnosis. The set of common causes consists of diseases that satisfy the commonality constraint, explaining all symptoms in a cluster. In this case, the common causes consist of the five diseases that explain Cr. Since there is only a single task, no disjointness constraints apply. Thus, the differential is equivalent to the common cause set.

Symptom 2: High Blood Urea Nitrogen

The second symptom, Bun, can either be assigned to the existing cluster or be placed in a new cluster. If we place it in the existing cluster, we obtain the following decomposition:

Decomposition 2a:

Cluster	Common Causes	Differential
(Cr,Bun)	Atn, Pra, Anl, Agn, Tb	Atn, Pra, Anl, Agn, Tb

This decomposition contains one task, consisting of the cluster (Cr,Bun) and the differential that explains it. The common causes are the five diseases that explain both Cr and Bun. These diseases can be found by taking the intersection of the causes for each symptom. As we noted before, Bun is more general than Cr, because the possible causes for Bun are a superset of those for Cr. Thus, the common causes remain unchanged. Since the possible causes for Bun subsume (or cover) the common causes for (Cr), this process is called *covering*. As before, since there is only a single task, the differential equals the set of common causes.

The alternative assignment that might have been tried places Bun in a new cluster. This process is called *adjoining*, and it gives rise to the following decomposition:

Decomposition 2b (incoherent):

Cluster	Common Causes	Differential
(Cr)	~~Atn~~, ~~Pra~~, ~~Anl~~, ~~Agn~~, ~~Tb~~	
(Bun)	~~Atn~~, ~~Pra~~, Pn, Vol, Prt, ~~Anl~~, ~~Agn~~, Sln, ~~Tb~~, Rta	Pn, Vol, Prt, Sln, Rta

This decomposition has two tasks. The first task explains Cr, while the second task explains Bun. The common cause sets for the two tasks are simply the possible causes for Cr and Bun, respectively. However, the differentials are now different from the common cause sets. The disjointness constraint

excludes some disorders from each differential, shown crossed out in each common causes set. The disjointness constraint removes disorders that are unifying, essentially those that explain two or more clusters. In this case, all of the explanations for (Cr) also explain (Bun). Thus, all of the potential explanations for (Cr) are removed, leaving no disorders in the differential for (Cr). The empty differential for (Cr) means that the decomposition is incoherent and can be pruned. Incoherency means that a decomposition posits commonality and disjointness constraints that cannot be satisfied.

The decompositional search algorithm actually did not generate decomposition 2b, because it could foresee that it would be incoherent. When the algorithm created decomposition 2a, it realized that the possible causes for Bun subsume the common causes for (Cr). Thus, placing Bun in its own cluster would have created a common causes set that subsumed another common causes set, resulting in an empty differential. In general, then, when covering is possible, no other assignment for a symptom need be considered.

Symptom 3: Dehydration

The third symptom, Dehyd, gives us the same option as before: assignment either to the existing cluster or to a new cluster. There are 10 possible causes for Dehyd. This time, covering is not possible because the possible causes for Dehyd do not subsume the common causes for (Cr,Bun). Instead, when we place Dehyd in the cluster (Cr,Bun), the set of common causes for that cluster becomes smaller. This process is called *restricting*, as opposed to covering. The resulting decomposition contains three disorders in its differential, instead of five:

Decomposition 3a:

Cluster	Common Causes	Differential
(Cr,Bun,Dehyd)	Atn, Pra, Anl	Atn, Pra, Anl

Because covering was not possible, other assignment operators are performed. A second decomposition is created by adjoining the new symptom. This gives the decomposition (Cr,Bun) (Dehyd), containing two tasks:

Decomposition 3b:

Cluster	Common Causes	Differential
(Cr,Bun)	~~Atn~~, ~~Pra~~, ~~Anl~~, Agn, Tb	Agn, Tb
(Dehyd)	~~Atn~~, ~~Pra~~, Ano, Dm, Mal, Pan, Vol, Prt, ~~Anl~~, Sln	Ano, Dm, Mal, Pan, Vol, Prt, Sln

Again, differentials are formulated by removing unifying explanations for both common cause sets. The two differentials are non-null, so this decomposition is coherent. This decomposition hypothesizes the presence of a disorder from each differential diagnosis: either Agn or Tb, plus one of the disorders explaining Dehyd alone.

In addition to covering, restricting, and adjoining, there is a fourth way to create a decomposition, called *admixing*. This operator places the new symptom in its own cluster along with one symptom that has been previously assigned. The preconditions for a previously assigned symptom to admix with a new symptom is that the previously assigned symptom must restrict its cluster. In this case, the symptom Cr restricts the cluster (Cr,Bun), because removing it yields cluster (Bun), which has more common causes. Admixing Dehyd with Cr therefore gives the decomposition (Bun) (Cr,Dehyd).

Decomposition 3c (incoherent):

Cluster	Common Causes	Differential
(Bun)	~~Atn~~, ~~Pra~~, Pn, Vol, Prt, ~~Anl~~, Agn, Sln, Tb, Rta	Pn, Vol, Prt, Agn, Sln, Tb, Rta
(Cr,Dehyd)	~~Atn~~, ~~Pra~~, ~~Anl~~	

This decomposition yields a null differential for the cluster (Cr,Dehyd), so this decomposition is incoherent and can be pruned.

Symptom 4: Low Urine Output

At this point, we have two coherent decompositions, (Cr,Bun,Dehyd) and (Cr,Bun) (Dehyd), with which to incorporate the fourth symptom, Uo. We consider each decomposition separately. For the first decomposition, covering is not possible because the causes for Uo do not subsume the common causes for (Cr,Bun,Dehyd). Thus, decompositions are created by restricting, adjoining, and admixing operators. Restriction gives the following decomposition:

Decomposition 4a:

Cluster	Common Causes	Differential
(Cr,Bun,Dehyd,Uo)	Atn, Pra	Atn, Pra

The differential contains the two diseases that explain all of the symptoms.

A second decomposition, (Cr,Bun,Dehyd) (Uo), is created by the adjoining operator:

Decomposition 4b:

Cluster	Common Causes	Differential
(Cr,Bun,Dehyd)	~~Atn~~, ~~Pra~~, Anl	Anl
(Uo)	~~Atn~~, ~~Pra~~, Lit, Rcc, Pn, Agn, Sln	Lit, Rcc, Pn, Agn, Sln

Third, we can admix Uo with Cr, which restricts its cluster, to initially create the following decomposition:

Decomposition 4c:

Cluster	Common Causes	Differential
(Bun,Dehyd)	~~Atn~~, ~~Pra~~, Vol, Prt, Anl, Sln	Vol, Prt, Anl, Sln
(Cr,Uo)	~~Atn~~, ~~Pra~~, Agn	Agn

However, a second step, called *ambiguation* can now be performed. Note that Bun covers the adjacent cluster, (Cr,Uo). This means that all of the common causes for that cluster also explain Bun. Hence, Bun could be explained at least as well by the new cluster. If we copy Bun to the new cluster we obtain the following decomposition:

Decomposition 4c′:

Cluster	Common Causes	Differential
(Bun,Dehyd)	~~Atn~~, ~~Pra~~, Vol, Prt, Anl, Sln	Vol, Prt, Anl, Sln
(Cr,Bun,Uo)	~~Atn~~, ~~Pra~~, Agn	Agn

Note that the common causes and differential have not changed. The symptom Bun has been tentatively ambiguated and now appears in two clusters. The meaning of an ambiguous symptom is that it has become general enough to cover more than one cluster.

At this point, we have copied Bun to a new cluster, but have not determined whether it should remain in its previous cluster. So a third step, called *disambiguation*, determines whether any previous assignments of ambiguous symptoms are still warranted. An ambiguous symptom should cover multiple clusters, but Bun restricts its previous cluster, (Bun,Dehyd), so that it should not remain in two clusters. We prefer to remove Bun from the cluster it restricts, so that additional common causes from that cluster are freed. The disambiguation step therefore changes the decomposition from (Bun,Dehyd) (Cr,Bun,Uo) to (Dehyd) (Cr,Bun,Uo):

Decomposition 4c″:

Cluster	Common Causes	Differential
(Dehyd)	~~Atn~~, ~~Pra~~, Ano, Dm, Mal, Pan, Vol, Prt, Anl, Sln	Ano, Dm, Mal, Pan, Vol, Prt, Anl, Sln
(Cr,Bun,Uo)	~~Atn~~, ~~Pra~~, Agn	Agn

Overall, the common causes and differential for (Dehyd), which was previously (Bun,Dehyd), have been increased, while the common causes and differential for (Cr,Bun,Uo), previously (Cr,Uo), are unchanged. The total effect of ambiguation and disambiguation has been to free the unnecessary constraint of Bun on the first cluster. An unnecessary constraint occurs when a symptom restricts its current cluster but could be assigned to another cluster that it covers. In this case, assigning Bun to the first cluster posts an unnecessary constraint because Bun restricted that cluster but covered the second cluster. Ambiguation and disambiguation remove that constraint by effectively moving Bun from the first cluster to the second.

So far, we have fully expanded the decomposition (Cr,Bun,Dehyd). The remaining decomposition, (Cr,Bun) (Dehyd), can also be expanded by Uo to produce decompositions. Uo can restrict (Cr,Bun) to give the decomposition (Cr,Bun,Uo) (Dehyd). However, this decomposition was already generated by the admixing operator, followed by ambiguation and disambiguation, namely, decomposition 4c″. We need not compute the differentials for this decomposition again. This example indicates that a decomposition can be generated by more than one path.

Uo can also restrict the second cluster, (Dehyd), to give the decomposition (Cr,Bun) (Dehyd,Uo). This decomposition qualifies for ambiguation because Bun covers the new cluster. However, it does not qualify for disambiguation because Bun does not restrict its current cluster (Cr,Bun). The restricting and ambiguation steps yield the following decomposition:

Decomposition 4d:

Cluster	Common Causes	Differential
(Cr,Bun)	~~Atn~~, ~~Pra~~, Anl, Agn, Tb	Anl, Agn, Tb
(Bun,Dehyd,Uo)	~~Atn~~, ~~Pra~~, Sln	Sln

Without ambiguation, we would obtain two coherent decompositions, (Cr,Bun) (Dehyd,Uo) and (Cr) (Bun,Dehyd,Uo), with the same common cause sets and differentials. Ambiguation avoids the arbitrary placement of Bun by assigning it to both clusters. The decompositional search algorithm checks

for two types of ambiguation. *Forward ambiguation* occurs whenever a previously assigned symptom covers the newly modified or created cluster. *Backward ambiguation* occurs whenever the new symptom covers more than one existing cluster. In this case, the assignment of Bun to the new cluster was an instance of forward ambiguation.

In addition to restriction, the new symptom can be adjoined to give the decomposition (Cr,Bun) (Dehyd) (Uo):

Decomposition 4e:

Cluster	Common Causes	Differential
(Cr,Bun)	~~Atn~~, ~~Pra~~, ~~Anl~~, ~~Agn~~, Tb	Tb
(Dehyd)	~~Atn~~, ~~Pra~~, Ano, Dm, Mal, Pan, Vol, Prt, ~~Anl~~, ~~Sln~~	Ano, Dm, Mal, Pan, Vol, Prt
(Uo)	~~Atn~~, ~~Pra~~, Lit, Rcc, Pn, ~~Agn~~, ~~Sln~~	Lit, Rcc, Pn

This is an example of a three-task decomposition. Differentials are computed by removing unifying explanations, those disorders that are shared by two or more common cause sets. The resulting decomposition explains the four symptoms by positing one disorder from each of the three differentials.

Finally, Uo can be admixed with Cr, which restricts its cluster, (Cr,Bun). This would give the decomposition (Bun) (Dehyd) (Cr,Uo). But Bun covers the cluster (Cr,Uo), so ambiguation would give the decomposition

(Bun) (Dehyd) (Cr,Bun,Uo).

However, Bun restricts its previous cluster, (Bun), because the common causes for an empty cluster is defined to be the universe of all disorders. The disambiguation step therefore removes Bun from that cluster, yielding the decomposition

( ) (Dehyd) (Cr,Bun,Uo).

Since null clusters are disallowed, this decomposition is syntactically invalid. Thus, this decomposition is incoherent and can be pruned.

2.4 Comparing the Algorithms

Despite their differences, candidate generation and decompositional search are closely related. In fact, decompositional search can be seen as a way of

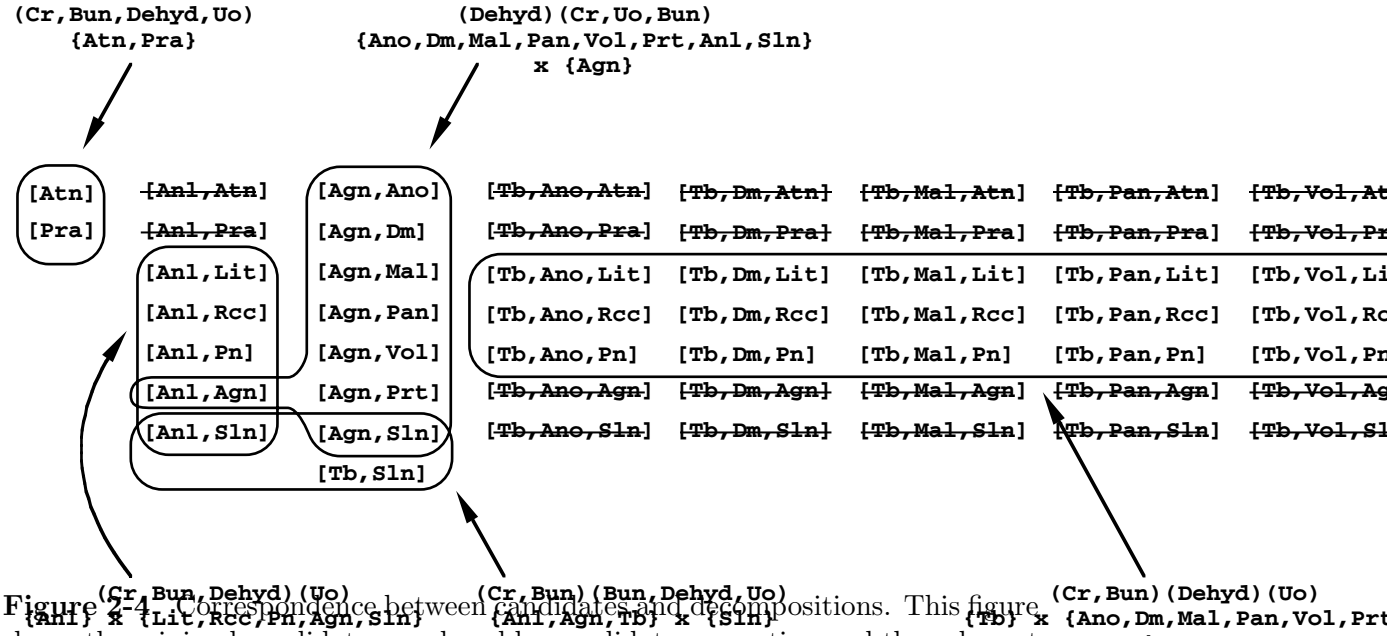

Figure 2-4. Correspondence between candidates and decompositions. This figure shows the minimal candidates produced by candidate generation and the coherent problem decompositions produced by decompositional search. Arrows link each decomposition with the group of minimal candidates that it entails.

generating minimal candidates. A problem decomposition posits the presence of at least one disorder from each differential. The set of all such combinations can be obtained by taking the Cartesian product of the differentials. The result is called the candidate set for the given decomposition.

For instance, we can compute the candidate set for decomposition 4e as follows:

$$\{\texttt{Tb}\} \times \{\texttt{Ano}, \texttt{Dm}, \texttt{Mal}, \texttt{Pan}, \texttt{Vol}, \texttt{Prt}\} \times \{\texttt{Lit}, \texttt{Rcc}, \texttt{Pn}\} =$$
$$\begin{array}{lll} [\texttt{Tb},\texttt{Ano},\texttt{Lit}], & [\texttt{Tb},\texttt{Ano},\texttt{Rcc}], & [\texttt{Tb},\texttt{Ano},\texttt{Pn}], \\ [\texttt{Tb},\texttt{Dm},\texttt{Lit}], & [\texttt{Tb},\texttt{Dm},\texttt{Rcc}], & [\texttt{Tb},\texttt{Dm},\texttt{Pn}], \\ [\texttt{Tb},\texttt{Mal},\texttt{Lit}], & [\texttt{Tb},\texttt{Mal},\texttt{Rcc}], & [\texttt{Tb},\texttt{Mal},\texttt{Pn}], \\ [\texttt{Tb},\texttt{Pan},\texttt{Lit}], & [\texttt{Tb},\texttt{Pan},\texttt{Rcc}], & [\texttt{Tb},\texttt{Pan},\texttt{Pn}], \\ [\texttt{Tb},\texttt{Vol},\texttt{Lit}], & [\texttt{Tb},\texttt{Vol},\texttt{Rcc}], & [\texttt{Tb},\texttt{Vol},\texttt{Pn}], \\ [\texttt{Tb},\texttt{Prt},\texttt{Lit}], & [\texttt{Tb},\texttt{Prt},\texttt{Rcc}], & [\texttt{Tb},\texttt{Prt},\texttt{Pn}] \end{array}$$

The same expansion for the other decompositions is shown in figure 2-4. This figure shows the close relationship between candidate generation and decompositional search. In this case, decompositional search produces the same set of minimal candidates as candidate generation.

The figure shows that problem decompositions represent a set of candidates in a compact form, grouping candidates that behave identically in the

candidate generation search tree. Therefore, decompositional search avoids much of the redundant reasoning performed in candidate generation. Decompositional search avoids redundancy because it groups disorders that are causally equivalent with respect to the given symptoms. Consequently, it avoids much of the combinatorial inefficiency of candidate generation.

Chapter 3

Problem Decomposition

Philosophers of science have repeatedly demonstrated that more than one theoretical construction can always be placed upon a given collection of data.

— Thomas Kuhn, *The Structure of Scientific Revolutions* (1970)

The basic construct of decompositional search is a problem decomposition. To paraphrase Kuhn [34, p. 76], problem decompositions are alternatives to candidates as constructions that can be placed on a given set of evidence. In this chapter, we provide a theoretical grounding for using problem decompositions as a framework for diagnostic problem solving.

3.1 Preliminaries

3.1.1 Diagnostic Problems

We begin by defining diagnostic knowledge bases and diagnostic cases. A *diagnostic knowledge base* KB is a triple, $\langle U_D, U_S, L\rangle$, containing a universal set U_D of disorders, a universal set U_S of symptoms, and a set L of causal links. A *causal link* is a pair, $\langle d, s\rangle$, containing a disorder $d \in U_D$ and a symptom $s \in U_S$. A disorder d *explains* symptom s if there exists a causal link $\langle d, s\rangle \in L$ between them. Notationally, we denote the fact that d explains s, or equivalently that $\langle d, s\rangle \in L$, by "$d \longrightarrow s$". We denote the opposite situation, when d does not explain s ($\langle d, s\rangle \notin L$), by "$d \not\longrightarrow s$". A causal link signifies that a disorder is a possible cause (or equivalently, a possible explanation) for a symptom.

Diagnostic knowledge bases provide the domain knowledge to diagnose a particular case. A *diagnostic case* is a pair, $\langle P, N\rangle$, where $P \subseteq U_S$ is a set of positive symptoms, and $N \subseteq U_S$ is a set of negative symptoms. The positive symptoms are symptoms that we wish to explain, while the negative symptoms provide information to help support or disprove proposed explanations. The positive symptoms P and negative symptoms N are exclusive but not exhaustive subsets of U_S. In other words,

$$\begin{aligned} P \cap N &= \emptyset \\ P \cup N &\subseteq U_S \end{aligned}$$

This means that a symptom in U_S is either positive, negative, or neither, in which case we consider that the value of the symptom is unknown. The input

to a diagnostic problem solver consists of a diagnostic knowledge base, representing domain knowledge, and a diagnostic case, representing a particular behavior that we wish to explain.

3.1.2 Causation and Explanation

In diagnosis, we are not concerned with individual causal links as much as the set of disorders that can cause a given symptom. This is called the set of *possible causes* for a symptom:

$$\text{Causes}(s) \;=\; \{d \in U_D \mid d \longrightarrow s\} \tag{3.1}$$

The goal of decompositional search is to explain not individual symptoms, but sets of symptoms. We say that a disorder d *explains* a set S of symptoms if d explains every symptom in S. Notationally, we express this as "$d \longrightarrow S$". The set of disorders that explains a set of symptoms is called the set of *common causes* for a set of symptoms:

$$\text{Causes}(S) \;=\; \{d \in U_D \mid \forall s \in S.\ d \longrightarrow s\} \tag{3.2}$$

Despite the similar mathematical notation, we use the term "possible cause" for a singleton symptom, but for a set of symptoms, we use the term "common causes". We can compute the common causes for a set of symptoms by taking the intersection of the possible causes for each symptom in the set:

$$\text{Causes}(S) \;=\; \bigcap_{s \in S} \text{Causes}(s) \tag{3.3}$$

In decompositional search, we deal not only with sets of symptoms but also sets of disorders. We say that a set D of disorders *disjunctively explains* a symptom s if each disorder in D explains s. Likewise, we say that a set D of disorders disjunctively explains a set S of symptoms if each disorder in D explains every symptom in S. We express these events by the notation "$D \stackrel{\vee}{\longrightarrow} s$" and "$D \stackrel{\vee}{\longrightarrow} S$", respectively. We use the term "disjunctive" to contrast this notion from that in candidate generation where a candidate H hypothesizes that all disorders in H are present. We say that H *conjunctively explains* S, represented as $H \stackrel{\wedge}{\longrightarrow} S$. The distinction between disjunctive and conjunctive explanation holds only for disorders. When we speak about explaining a set of symptoms, we always mean that every element in that symptom is explained.

A simple test to determine whether a set of disorders disjunctively explains a symptom or set of symptoms is provided by the following theorem.

Theorem 1 *A set D of disorders disjunctively explains a symptom s if and only if $D \subseteq$ Causes(s). Likewise, a set D of disorders disjunctively explains a set S of symptoms if and only if $D \subseteq$ Causes(S).*

Proof The quantity Causes(s) contains all disorders that can explain s. If D is a subset of Causes(s), then all disorders in D can explain s; otherwise, some disorder in D cannot explain s. This argument also holds when S is substituted for s. ∎

This theorem tells us that a symptom is disjunctively explained by a set D of disorders if its set of possible causes subsumes D. Likewise, a set of symptoms is disjunctively explained by D if its set of common causes subsumes D.

3.2 Problem Decompositions

3.2.1 A Set of Symptom Clusters

Having considered sets of symptoms, we now define the main construct of decompositional search: the problem decomposition, which is essentially a collection of sets of symptoms.

Definition 1 *A* problem decomposition $\mathcal{C}$ *for a set P of positive symptoms is a collection of subsets of P (called* clusters*) such that*

1. *Every positive symptom $s \in P$ appears in some cluster $C \in \mathcal{C}$, and*
2. *Each cluster in $\mathcal{C}$ must have a symptom s not in any other cluster.*

The intuition for the above definition is that a decomposition represents a structure in which every cluster has a separate disjunctive explanation. The first condition ensures that every positive symptom is explained. The second condition helps ensure that each cluster is necessary. If a cluster does not have a unique symptom s, then all symptoms in that cluster would already be explained by virtue of their presence in other clusters.

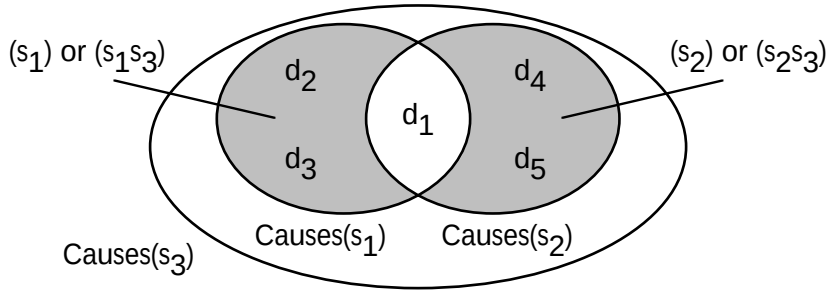

Figure 3-1 The concept of ambiguity. This figure shows the decomposition (s_1) (s_2). Symptom s_3 could be assigned to either cluster equally well.

Example Consider a case with four positive symptoms: s_1, s_2, s_3, and s_4. Then $(s_1 s_2 s_3)$ $(s_1 s_4)$ is a decomposition, because it contains all positive symptoms and each cluster has a unique symptom. However, $(s_1 s_2)$ $(s_1 s_3)$ is not a decomposition, because it does not contain symptom s_4. Moreover, $(s_1 s_2 s_3)$ $(s_1 s_4)$ $(s_2 s_4)$ is not a decomposition, because the third cluster lacks a symptom that does not appear in the first or second cluster. ∎

3.2.2 Ambiguous and Instantiated Decompositions

A problem decomposition is similar to a partition of symptoms, except that in a partition every symptom in a cluster must be unique to that cluster. In contrast, a problem decomposition allows symptoms to appear in more than one cluster, as long as at least one symptom in each cluster is unique. We allow non-unique assignments of symptoms because a symptom may be explained equally well by more than one cluster. When this occurs, we say that the symptom is *ambiguous* with respect to the decomposition. For instance, consider the example shown in figure 3-1. Here we show a Venn diagram representation of three symptoms, where symptoms s_1 and s_2 are placed in separate clusters. There is no clear preference for placing s_3 in either of these clusters. A representation based on partitions would force us to place s_3 in each cluster, resulting in two partitions: $(s_1 s_3)$ (s_2) and (s_1) $(s_2 s_3)$. But a representation based on decompositions places s_3 in both clusters to signify its ambiguous assignment: $(s_1 s_3)$ $(s_2 s_3)$.

A symptom can be assigned ambiguously if it subsumes the common cause set of more than one cluster. A decomposition that contains an ambiguous assignment is called an *ambiguous* decomposition. Otherwise, if it contains

no ambiguous assignments, it is an *instantiated* decomposition. An ambiguous decomposition represents a set of instantiated decompositions. These instantiations can be computed by trying every possible assignment of the ambiguous symptoms.

Example Consider the ambiguous decomposition $(s_1 s_2)\ (s_1 s_3 s_4)\ (s_1 s_3 s_5)$. Then s_1 is ambiguous, having assignments in all three clusters. Symptom s_3 is also ambiguous, having assignments in the second and third clusters. By choosing all possible combinations of these assignments, we can compute the instantiations of $\mathcal{C}$:

$$\begin{array}{lll} (s_1 s_2)(s_3 s_4)(s_5), & (s_2)(s_1 s_3 s_4)(s_5), & (s_2)(s_3 s_4)(s_1 s_5), \\ (s_1 s_2)(s_4)(s_3 s_5), & (s_2)(s_1 s_4)(s_3 s_5), & (s_2)(s_4)(s_1 s_3 s_5) \end{array}$$ ∎

Since by definition each cluster has at least one symptom not in any other cluster, each cluster has at least one unambiguous symptom. Therefore, the instantiation procedure is guaranteed not to produce any null clusters.

When a symptom is in several clusters, this does not mean that it is actually caused by every associated differential, but by at least one of them. From the standpoint of explaining the symptoms, these differentials are equivalent, and the exact causal relationship is not a critical decision.

3.3 Differential Diagnoses

3.3.1 Commonality and Disjointness Constraints

A problem decomposition assigns the given positive symptoms into clusters and explains each cluster separately. Each cluster of symptoms is assumed to be caused by a separate disorder. As we have mentioned previously, the list of such disorders is called the differential diagnosis for that cluster, or differential for short. We denote the differential for cluster C as Diff(C). Informally, when we speak about cluster C, we shall refer to Diff(C) as its "associated" differential, and other differentials Diff(C'), where $C' \neq C$, as "adjacent" differentials. Conversely, when we are speaking about differential Diff(C), we shall refer to C has its "associated" cluster, and to $C' \neq C$ as "adjacent" clusters.

While the cluster and differential diagnosis make up the structure of a decomposition, the contents are determined by two types of constraints. A

commonality constraint ensures that each cluster must be explainable by a single disorder. A *disjointness constraint* ensures that each cluster is justified because it cannot be explained fully by an adjacent differential.

The commonality constraint is easy to test. The set of disorders that can explain a cluster is simply the set of common causes for that cluster.

Definition 2 *A disorder d satisfies the* commonality constraint *for cluster C if $d \in \text{Causes}(C)$.*

The disjointness constraint is a bit more complicated. This constraint is satisfied when a disorder cannot explain the "justifying" set of symptoms in any adjacent cluster. In turn, a justifying symptom, or *justification*, is a symptom that cannot be explained by an adjacent differential. If a cluster has such a symptom, then its existence is "justified" because no other differential can explain the cluster. We refer to the justifications of a cluster as $\text{Just}(C)$.

Definition 3 *A disorder d satisfies the* disjointness constraint *for cluster C if for every other cluster $C' \neq C$, d does not explain the justifications* $\text{Just}(C')$ *for that cluster.*

Definition 4 *A symptom s in cluster C is a* justification *for that cluster if for every other cluster $C' \neq C$, s is not explained by the differential* $\text{Diff}(C')$ *for that cluster.*

The two definitions are similar in that they state that a causal relationship does not hold. However, the first definition states that an individual disorder cannot explain a set of justifying symptoms, while the second definition states that a set of disorders cannot explain an individual justifying symptom. Intuitively, the justification set of a cluster is its "core", the part that cannot be explained by any other differential, either collectively or individually.

3.3.2 Definition of Differential Diagnoses

The commonality and disjointness constraints are useful not only to determine whether a decomposition is plausible, but also define solutions to that decomposition. These solutions are in the form of a differential diagnosis, a set of disorders that satisfy the commonality and disjointness constraints for a cluster. The definition of differential diagnoses is slightly complicated by the fact that it is recursive: differentials are used to define justifications,

which in turn, are used to define differentials. We can see the recursion explicitly if we combine definitions 2, 3, and 4:

$$\text{Diff}(C) \;=\; \left\{ d \in \text{Causes}(C) \;\middle|\; \begin{array}{l} \forall C' \in \mathcal{C} - \{C\}. \\ \quad \exists s' \in C'. \\ \quad\quad (d \not\rightarrow s') \wedge \\ \quad\quad \forall C'' \in \mathcal{C} - \{C'\}. \\ \quad\quad\quad (\text{Diff}(C'') \not\overset{\vee}{\rightarrow} s') \end{array} \right\}$$

In other words, we define a differential as those disorders that explain their own cluster but cannot explain, either individually or disjunctively, the justifying symptoms in any other cluster. Recursive definitions can often fail to make sense. However, the next theorem indicates that differentials are indeed well-defined.

Theorem 2 *Given a decomposition $\mathcal{C}$, for each cluster $C \in \mathcal{C}$ there exists a unique set of disorders that satisfy the commonality and disjointness constraints for C.*

Proof Suppose that there exists a set of differentials $\text{Diff}_A(C)$, for $C \in \mathcal{C}$, that satisfies the constraints and there exists another set of differentials $\text{Diff}_B(C)$, for $C \in \mathcal{C}$ that also satisfies the constraints. For the each set of differentials, there is a corresponding set of justification sets, $\text{Just}_A(C)$ and $\text{Just}_B(C)$, respectively, for $C \in \mathcal{C}$. We will argue by *reductio ad absurdum* that for all $C \in \mathcal{C}$, $\text{Diff}_A(C) = \text{Diff}_B(C)$.

Suppose this statement is false. Then a cluster C exists such that either $\text{Diff}_A(C)$ has some disorder d not in $\text{Diff}_B(C)$ or vice versa. Without loss of generality, let us assume that $\text{Diff}_B(C)$ has some disorder d_1 not in $\text{Diff}_A(C)$. Then an infinitely long argument follows:

1. As stated previously, a cluster C exists such that $\text{Diff}_B(C)$ has a disorder d_1 not in $\text{Diff}_A(C)$.

2. Then a cluster C' exists such that d_1 does not explain $\text{Just}_B(C')$, but d_1 does explain $\text{Just}_A(C')$. Hence, there is a symptom s_1 in $\text{Just}_B(C')$ (which d_1 does not explain) that is not in $\text{Just}_A(C')$.

3. Then a cluster C'' exists such that $\text{Diff}_B(C'')$ does not explain s_1, but $\text{Diff}_A(C'')$ does explain s_1. Hence, there is a disorder d_2 in $\text{Diff}_B(C'')$ (which does not explain s_1) that is not in $\text{Diff}_A(C'')$.

4. Then a cluster C''' exists such that d_2 does not explain $\text{Just}_B(C''')$, but d_1 does explain $\text{Just}_A(C''')$. Hence, there is a symptom s_2 in $\text{Just}_B(C''')$ (which d_2 does not explain) that is not in $\text{Just}_A(C''')$.

5. [and so on]

If our supposition were true, this argument must continue infinitely. At each step, either a justification set $\text{Just}_B(C)$ or a differential $\text{Diff}_B(C)$ for the second set of differentials grows. But eventually this process must stop because the justification set for a cluster C cannot be larger than C and because the differential for C cannot be larger than $\text{Causes}(C)$. Hence at some point, the line of reasoning must fail and so the initial supposition must be false. ∎

Thus, differential diagnoses are well defined by the commonality and disjointness constraints. But a well defined set of constraints does not mean that they are satisfiable. In applying the constraints, one or more differentials may be empty. When this occurs, the common and disjointness constraints cannot be satisfied, and we say that the decomposition is *incoherent*. Otherwise, if all differential diagnoses are nonempty, the decomposition is *coherent*.

Definition 5 *A decomposition $\mathcal{C}$ is* coherent *if, for every cluster C in $\mathcal{C}$, there exists a nonempty differential for C. Otherwise, it is* incoherent.

Coherency provides a criterion for plausibility of a given decomposition. If a decomposition imposes a satisfiable set of constraints, it is plausible; otherwise, it is implausible. Coherency can be used as a plausibility criterion to prune decompositions from a search tree, in the same way that minimality is used to prune candidates in candidate generation.

3.4 Differential Formulation

The definitions above cannot be used in an algorithm until they are converted from their declarative form into a procedural form. We now present an procedure for formulating differential diagnoses.

3.4.1 Exclusion Sets and Unifying Disorders

To assist us, we first develop two computational constructs to be used as intermediate quantities in the differential formulation algorithm. The first

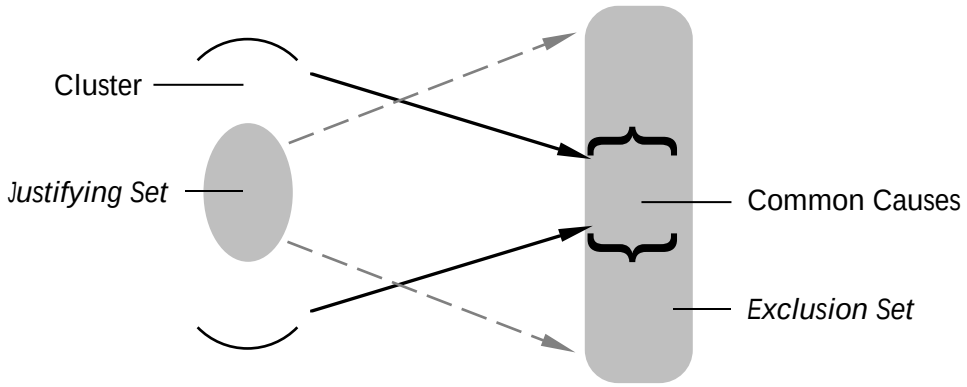

Figure 3-2 Justification and exclusion sets. This figure shows the relationship between the cluster, justification set, exclusion set, and common causes set. The justification set is a subset of its cluster, so the exclusion set is a superset of the common causes set for the cluster.

computational construct is the exclusion set. By Definition 3, a justification set cannot be explained by any disorder in an adjacent differential diagnosis. We define the disorders that can explain a justification set for a cluster as its *exclusion set*. The exclusion set for cluster C is simply the set of common causes for the justification set of C:

$$\text{Excl}(C) = \bigcap_{s \in \text{Just}(C)} \text{Causes}(s) \tag{3.4}$$

Note that since the justification set is a subset of its cluster, the exclusion set will be a superset of the common causes for the cluster. This relation is shown in figure 3-2. The exclusion set specifies those disorders that cannot be in an adjacent differential. The use of this construct is expressed in the following theorem.

Theorem 3 *A disorder d satisfies the disjointness constraint for cluster C in decomposition $\mathcal{C}$ if and only if d is in the exclusion set for no more than one cluster in $\mathcal{C}$.*

Proof ($\Rightarrow$) If d satisfies the disjointness constraint for cluster C, then d cannot explain the justification set for any adjacent cluster $C' \neq C$. Hence, d cannot be in the exclusion set for C'. However, d must be in the exclusion set for C because Just(C) is a subset of C. Thus, d is in the exclusion set for exactly one cluster in $\mathcal{C}$.

Algorithm 1 (Duplicates)

Procedure DUPLICATES (Collection $\mathcal{A}$ of sets)

```
1  Initialize the elements seen: Seen ← ∅
2  Initialize the duplicate elements: Duplicates ← ∅
3  For each set A in 𝒜 do
4      For each element a in A do
5          If the element was seen before [a ∈ Seen] then
6              The element is duplicate: Duplicates ← Duplicates ∪ {a}
7          The element has been seen: Seen ← Seen ∪ {a}
8  Return the duplicate elements: Duplicates
```

Figure 3-3 Algorithm for duplicate elements. This algorithm finds the duplicate elements in a collection of sets.

($\Leftarrow$) If d is not in the exclusion set for cluster C', then d does not explain the justification set $\text{Just}(C')$ for that cluster. If this holds for all clusters C' except for possibly one cluster C, then d satisfies the definition for the disjointness constraint. ∎

This theorem shows that we can use exclusion sets to remove disorders from adjacent differentials. We could therefore remove disorders by subtracting each exclusion set from each differential. This would require $O(|\mathcal{C}|^2)$ time, where $|\mathcal{C}|$ is the number of clusters in decomposition $\mathcal{C}$.

But Theorem 3 also suggests a way to reduce this process to $O(|\mathcal{C}|)$ time. It suggests that we remove disorders contained in two or more exclusion sets. Since these disorders explain the justification sets of more than one cluster, we call these disorders the *unifying disorders* for a decomposition; this is our second computational construct.

The unifying disorders can be computed with the aid of a function to compute duplicate elements from a collection of sets. Let $\mathcal{A}$ be a collection of sets A, and let a represent an element of A. Then the duplicates function is

$$\text{DUPLICATES}(\mathcal{A}) \;=\; \{a \mid \exists A, A' \in \mathcal{A}.\ (A \neq A') \wedge (a \in A) \wedge (a \in A')\}$$

One implementation of this function is given in figure 3-3.

With this function, the set of unifying disorders is simply the duplicate disorders in the exclusion sets of the decomposition. Let the set of exclusion

sets be denoted by

$$\text{ExclSets}(\mathcal{C}) \;=\; \{\text{Excl}(C) \mid C \in \mathcal{C}\}$$

Then the set of unifying disorders is simply

$$\text{Unifying}(\mathcal{C}) \;=\; \textsc{Duplicates}(\text{ExclSets}(\mathcal{C}))$$

This process essentially finds the duplicate elements among a set of exclusion sets. Once the unifying disorders are computed, we can enforce the disjointness constraints by removing the unifying disorders from each differential.

Ironically, one would think that diagnosis should be a process of finding unifying explanations, rather than discarding them. However, decompositional search does retain unifying explanations, but only when the symptoms they explain are in the same cluster. When the symptoms are divided into different clusters, a unifying disorder argues against having separate causes for them. In that context, the unifying disorder is removed.

3.4.2 Algorithm for Differential Formulation

With the exclusion set and unifying disorder set defined, we now present the algorithm for differential formulation in figure 3-4. This algorithm begins by initializing the justification set, exclusion set, and differential for each cluster. Then, it computes the unifying disorders as defined above and subtracts them from each differential. It computes a new justification set for each cluster. If the new justification set is different from the old one, its exclusion set is recomputed and a flag is set, indicating that a justification set has been altered. Finally, if no justification sets were altered, the algorithm halts and returns the coherent decomposition, along with its differential diagnoses. Otherwise, if a justification was altered, the algorithm performs another iteration of the above steps, beginning with the unifying disorders. Along the way, if any differentials or justification sets are found to be empty, the decomposition is incoherent, and the algorithm terminates.

The correctness of this algorithm is expressed by the following theorem.

Theorem 4 *Algorithm 2 terminates. Moreover, it returns $\mathcal{C}$ if and only if $\mathcal{C}$ is coherent. Finally, the differentials satisfy the commonality and disjointness constraints of $\mathcal{C}$.*

Algorithm 2 (Differential Formulation)

Procedure FORMULATE-DIFFERENTIALS (Decomposition $\mathcal{C}$)

```
 1  Initialize the unifying disorders: Unifying(C) ← ∅
 2  For each cluster C in decomposition C do
 3      Initialize its justifications: Just(C) ← C
 4      Initialize its exclusion set: Excl(C) ← Causes(C)
 5      Initialize its differential: Diff(C) ← Causes(C)
 6  Return FORMULATE-DIFFERENTIALS-AUX(C)
```

Procedure FORMULATE-DIFFERENTIALS-AUX (Decomposition $\mathcal{C}$)

```
 7  Find the unifying disorders: Unifying(C) ← DUPLICATES(ExclSets(C))
 8  For each cluster C in decomposition C do
 9      Remove the unifying disorders: Diff(C) ← Diff(C) − Unifying(C)
10      If the differential, Diff(C), is empty then
11          Return "incoherent"
12  Initialize a flag for altered justification sets: Altered? ← nil
13  For each cluster C in decomposition C
14      Remove justifications that are explained by adjacent differentials:
            NewJust ← {s ∈ Just(C) | ∀C' ∈ C − {C}. Diff(C') ↛ s}
15      If the new justification set, NewJust, is empty then
16          Return "incoherent"
17      Else if the new justification set is altered [NewJust ≠ Just(C)] then
18          Store the new justification set: Just(C) ← NewJust
19          Revise the exclusion set: Excl(C) ← ∩_{s∈Just(C)} Causes(s)
20          Set the altered justification flag: Altered? ← t
21  If the altered justification flag, Altered?, is nil then
22      Return the decomposition C
23  Else if the altered justification flag, Altered?, is t then
24      Return FORMULATE-DIFFERENTIALS-AUX(C)
```

Figure 3-4 Algorithm for differential formulation. This algorithm contains a main procedure that initializes variables and a recursive procedure that performs the actual computation.

Proof The algorithm terminates because the justification sets and differents both are monotonically nonincreasing in size. Hence, they must either reach a fixed point, at which point the algorithm terminates, or one of these sets becomes empty, at which point the algorithm also terminates. For correctness, note that a decomposition is incoherent only if one of its differentials is empty. The algorithm returns "incoherent" under this condition. The algorithm also returns "incoherent" if a cluster lacks a justifying symptom. A decomposition would be incoherent under this condition because the exclusion set for a empty justification set is the universe of disorders, so all other differentials would be empty. Note that each differential satisfies the commonality constraints, because it is a subset of the common causes for the cluster. Finally, note that each differential satisfies the disjointness constraints, because any disorder that is in more than one exclusion set is removed when the unifying disorders are computed. ∎

Example Consider the decomposition $(s_1 s_2 s_3)$ $(s_4 s_5)$ (s_6), shown in figure 3-5. The figure shows how the differential formulation algorithm works. For reference, let C_1, C_2, and C_3 denote the first, second, and third clusters, respectively. Then, the first step removes disorder F from the differentials for clusters C_2 and C_3, because it is unifying. In the second step, symptom s_2 is found to be non-justifying, because it is explained by the differential for cluster C_2. Symptom s_3 is also found to be non-justifying because now it is explained by the differential for C_3. The third step reveals disorder G to be a unifying disorder, because it explains not only cluster C_3 but also the only justifying symptom in cluster C_1, namely, s_1. The process terminates after the third step, when the justification sets and differentials reach a fixed point. Removal of the unifying explanations F and G gives us the final set of differentials. The resulting differentials are {A, B}, {C}, and {D, E}. ∎

3.5 Decompositions and Candidates

Candidate generation and decompositional search can be compared at the level of their common denominator, namely, candidates. Problem decompositions represent a set of candidates because one disorder can be selected from each differential, giving rise to a conjunction of disorders that explains

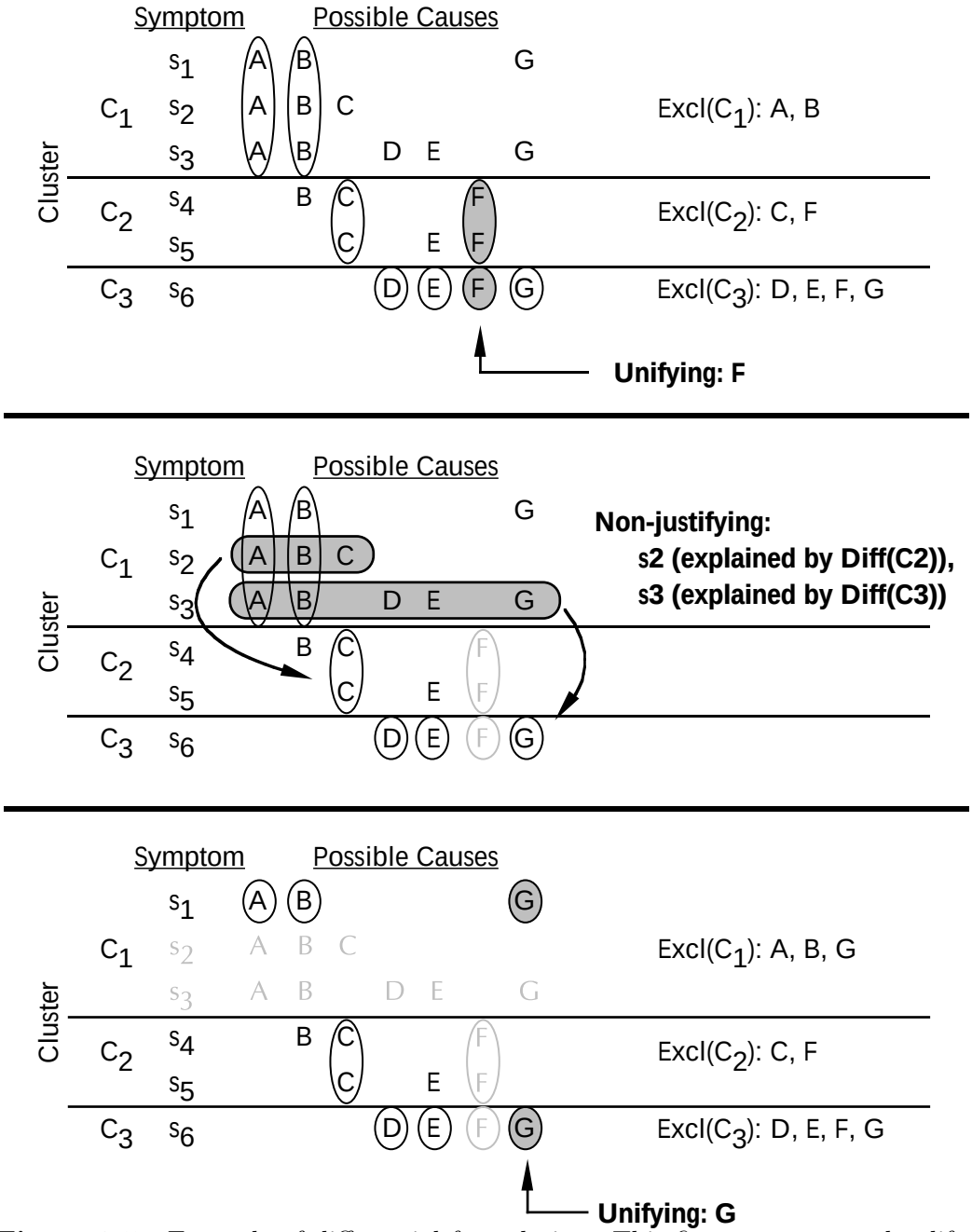

Figure 3-5 Example of differential formulation. This figure computes the differential diagnoses for the decomposition $(s_1 s_2 s_3)$ $(s_4 s_5)$ (s_6). These clusters are separated by horizontal lines, and the possible causes for each symptom are shown. The first step removes the unifying disorder F; the second step makes symptoms s_2 and s_3 non-justifying; the third step removes the unifying disorder G.

the given problem. However, the match between the two algorithms is not perfect. Candidate generation is intended to produce only candidates that are minimal. For decompositional search, though, generation of candidates is an incidental feature and adherence to the minimality criterion is not guaranteed. In this section, we explore the relationship between problem decompositions and minimality.

3.5.1 Candidate Sets

Each cluster in a decomposition is associated with two sets of disorders: its common cause set and its differential diagnosis. A Cartesian product can be performed on either of these sets, giving rise to an initial candidate set and a final candidate set. The initial candidate set is defined as follows:

Definition 6 *A candidate H is in the* initial candidate set *for decomposition $\mathcal{C}$ if there exists a one-to-one correspondence between each disorder d in H and each cluster C in $\mathcal{C}$ such that d explains C.*

This definition tells us whether a candidate is in the initial candidate set. The inverse process is to generate all candidates that satisfy the definition. We can compute the initial candidate set for a decomposition by taking the Cartesian product of the common cause sets for the clusters in the decomposition and accepting the syntactically valid candidates that result. Some of the resulting tuples may be syntactically invalid because they contain duplicate disorders; these disorders can be eliminated.

Example Consider the decomposition $(s_1 s_2 s_3)$ $(s_4 s_5)$ (s_6) shown previously in figure 3-5. The common causes for these clusters are {A, B}, {C, F, H}, and {D, E, F, G}. The initial candidate set can be computed by taking the Cartesian product of these sets:

$$\{\text{A, B}\} \times \{\text{C, F, H}\} \times \{\text{D, E, F, G}\} =$$

[A,C,D],	[A,C,E],	[A,C,F],	[A,C,G],
[A,F,D],	[A,F,E],	~~[A,F,F]~~,	[A,F,G],
[A,H,D],	[A,H,E],	[A,H,F],	[A,H,G],
[B,C,D],	[B,C,E],	[B,C,F],	[B,C,G],
[B,F,D],	[B,F,E],	~~[B,F,F]~~,	[B,F,G],
[B,H,D],	[B,H,E],	[B,H,F],	[B,H,G]

Except for $[A, F, F]$ and $[B, F, F]$, which are syntactically invalid candidates, the result comprises the initial candidate set for the decomposition. ∎

Analogously, we can define the final candidate set for a problem decomposition by using differential diagnoses instead of common cause sets:

Definition 7 *A candidate H is in the* final candidate set *for decomposition $\mathcal{C}$ if there exists a one-to-one correspondence between each disorder d in H and each cluster C in $\mathcal{C}$ such that d is in the differential diagnosis for C.*

When the context is clear, we shall refer to this set simply as the "candidate set" for a decomposition, or Cands($\mathcal{C}$). Since the differentials are subsets of the common cause sets, the final candidate set is a subset of the initial candidate set. We can compute the final candidate set by taking the Cartesian product of the differentials of a decomposition. Because no two differentials contain the same disorder, all of the resulting candidates will be syntactically valid. The formula for computing the candidate set is:

$$\text{Cands}(\mathcal{C}) \quad = \quad \bigtimes_{C \in \mathcal{C}} \text{Diff}(C)$$

Example The decomposition of the previous example has differential diagnoses of {A, B}, {C,}, and {D, E}. Their Cartesian product is:

$$\{\text{A, B}\} \times \{\text{C}\} \times \{\text{D, E}\} \quad = \quad \begin{array}{ll} \text{[A,C,D]}, & \text{[A,C,E]}, \\ \text{[B,C,D]}, & \text{[B,C,E]} \end{array}$$

The result comprises the final candidate set for the decomposition. ∎

As we have seen, a Cartesian product represents a set of candidates compactly. On the other hand, explicit representations of candidates require space proportional to the product of the differential sizes.

3.5.2 Justifications for Disorders and Differentials

Previous chapters have demonstrated a close relationship between the candidate set of a problem decomposition and the minimal candidates produced by candidate generation. In the example presented in chapter 2, for example, decompositional search produced the same set of minimal candidates as candidate generation. The reason for the close relationship stems from the use

of justifications to define differential diagnoses. The concept of justification is closely related to a similar concept that could be used to define minimality. Let us call this concept "disorder-based justification" as defined below.

Definition 8 *Suppose H is a candidate for a set P of positive symptoms. Then symptom s is a* disorder-based justification *for disorder d in H if d explains s and for all other disorders d' in H, where $d' \neq d$, d' cannot explain s.*

The concept of disorder-based justification is similar to our notion of justification, which might be considered "differential-based". Just as differential-based justification defines coherency, disorder-based justification can define minimality. The minimality of a candidate was previously defined by seeing whether any subset of the candidate could explain the positive symptoms. With disorder-based justification, we can create an alternate, but equivalent, definition of minimality: each disorder must have a disorder-based justification. The equivalence of this alternate definition is provided by the following theorem.

Theorem 5 *A candidate H is minimal for a set P of positive symptoms if and only if, for each disorder d in H, there exists a symptom s in P that is a disorder-based justification for d.*

Proof ($\Rightarrow$) Suppose candidate H is minimal for P, but contrary to the hypothesis, some disorder d in H lacks a disorder-based justification. Then there is no symptom explained by d that some other disorder d' in H does not also explain. Then $H - \{d\}$ is a candidate for P, contradicting the supposition that H is minimal.

($\Leftarrow$) Suppose every disorder in H has a disorder-based justification, but contrary to the hypothesis, H is not minimal. Then some subset $H_{\subset}$ of H is also a candidate for P. Suppose d is in $H - H_{\subset}$; since $H_{\subset}$ is a candidate for P, $H - \{d\}$ is also a candidate for P. Consider the set of symptoms $P_{\subset}$ in P that d explains. Either (1) $P_{\subset}$ is empty, in which case d lacks a disorder-based justification, or (2) since $H - \{d\}$ is a candidate for P, each symptom s in $P_{\subset}$ is associated with another disorder d' in H that explains s. Thus d lacks a disorder-based justification, contradicting the supposition. ∎

The above notion of disorder-based justification holds for a single disorder, in contrast with the concept of justification used in decompositional

search, which holds for a set of disorders. Nevertheless, the general idea of justification is similar. In determining minimality of a candidate, one finds a disorder-based justification for each disorder in the candidate. In determining the coherency of a decomposition, one finds a differential-based justification for each cluster and its associated differential in the decomposition. The similarity is apparent when disorder-based and differential-based justification are written in predicate logic:

$$\begin{array}{rcl} \text{d-Just}(s,d,H) & \equiv & (d \longrightarrow s) \quad \wedge \quad \forall d' \in H - \{d\}. \quad (d \not\longrightarrow s) \\ \text{Just}(s,C,\mathcal{C}) & \equiv & (\text{Diff}(C) \stackrel{\vee}{\longrightarrow} s) \quad \wedge \quad \forall C' \in \mathcal{C} - \{C\}. \quad (\text{Diff}(C') \stackrel{\vee}{\not\longrightarrow} s) \end{array}$$

The analogous structure of these equations indicates that differential formulation and minimal candidates may be closely related. We now explore this relationship in more detail.

3.5.3 Candidate Sets and Minimality

We have seen that problem decompositions entail a set of candidates, and that the nature of justifications suggests that they should be similar to minimal candidates. We now consider the relationship between candidate sets and minimality in more detail. Let us consider a problem decomposition as a generator of minimal candidates. The decomposition begins with a set of initial candidates and, after differential formulation, ends with a set of final candidates. We can ask, then, whether differential formulation discards any minimal candidates, which is essentially a question about completeness. We can also ask whether differential formulation retains any nonminimal candidates, which is essentially a question about soundness.

The answer to the first question, as expressed in the following theorem, is that differential formulation is indeed complete.

Theorem 6 *Let $\mathcal{C}$ be a problem decomposition for a set P of positive symptoms. If H is in the initial candidate set for $\mathcal{C}$, and H is minimal, then H is in the final candidate set for $\mathcal{C}$.*

Proof Let H be a minimal candidate in the initial candidate set for $\mathcal{C}$, but contrary to the hypothesis, H is not in the final candidate set for $\mathcal{C}$. Then there must be some disorder d in H that is in the exclusion set for both C and some adjacent cluster $C' \neq C$. The differential for that cluster C' must

also have some disorder in H; call it d'. We now show that H is nonminimal because $H - \{d'\}$ is still a candidate.

Since d is in the exclusion set for C and C', every symptom s' in C' is explained either by d or by the differential for some other cluster $C'' \neq C'$. If $C'' = C$, then d explains every symptom s' in C'. Hence d explains not only C but also C', meaning that d' is not necessary for H to be a candidate. Otherwise, if $C'' \neq C$, then H contains some disorder d'' from the differential of C'' such that every symptom s' in C' is explained by either d or d''. Hence d and d'' explain not only C and C'' but also C', meaning that, again, d' is not necessary for H to be a candidate. In either case, $H - \{d'\}$ is a candidate because all symptoms in P are still explained. Thus, we have contradicted the supposition that H is minimal, proving that H must be in the final candidate set for $\mathcal{C}$. ∎

However, the answer to the question about soundness is false, as expressed in the following theorem.

Theorem 7 *There exists a set P of positive symptoms and a problem decomposition $\mathcal{C}$ for P, such that not all candidates in the final candidate set for $\mathcal{C}$ are minimal.*

Proof Proof by example. Consider the example shown previously in figure 3-5. The final candidate set is [A,C,D], [A,C,E], [B,C,D], and [B,C,E]. The first three candidates are minimal, but [B,C,E] is nonminimal, because [B,E] explains the positive symptoms. ∎

It should not surprise us that both completeness and soundness cannot be achieved simultaneously. If order to maintain soundness in the example above, we would have to eliminate solution [B,C,E]. We could do this by removing either disorder B from the first differential or disorder E from the second differential. But removing a disorder from a differential would eliminate either minimal candidate [B,C,D] or [A,C,E] from the final candidate set, thereby violating completeness. On the other hand, to maintain completeness and express the three minimal candidates in a single Cartesian product form, we must include the nonminimal candidate [B,C,E], thereby violating soundness.

This conflict can be seen from a geometric perspective. If we view each of the n differentials of a decomposition along a different dimension, we get an n-dimensional candidate space, as shown in figure 3-6. Each point in this

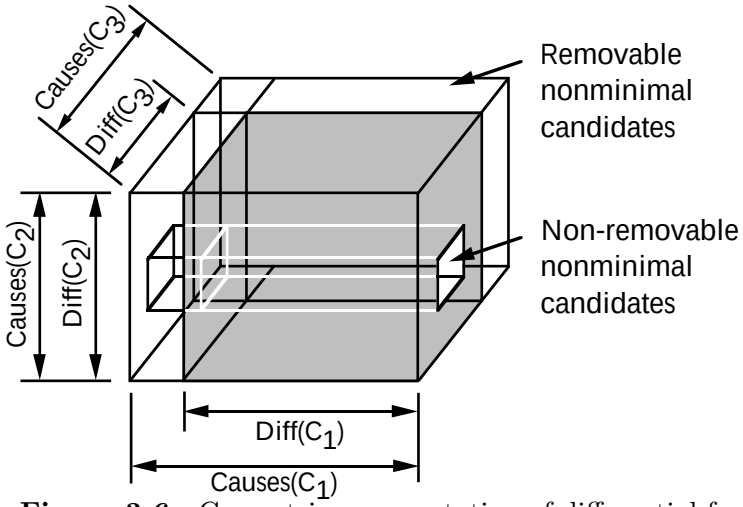

Figure 3-6 Geometric representation of differential formulation. Each axis represents a different task, each with a common causes set and differential. The Cartesian product of the common causes set contains the initial candidate set for the decomposition. Differential formulation removes disorders from the common causes set to compute the differentials. The Cartesian product of the differentials contains the final candidate set for the decomposition. Some nonminimal candidates cannot be removed by differential formulation because their elements also define minimal candidates.

space represents a candidate. Initially, the common cause sets define the set of initial candidates. But differential formulation removes disorders from the differential diagnoses, which is analogous to slicing off subregions of the candidate space. Eventually, the differentials define the set of final candidates. All minimal candidates within the space remain, but some nonminimal candidates may also remain. These nonminimal candidates occupy subregions within the minimal candidate space, so they cannot be removed without also eliminating minimal candidates.

Our definition of differential diagnoses favors completeness at the expense of soundness. One might expect that soundness should be a more important criterion, but we should keep in mind the goal of abductive reasoning. In a deductive system, the worry is that we will derive an erroneous statement, an error of commission. Hence, soundness is usually preferred over completeness in such systems. But in an abductive system, the worry is that we will miss a diagnosis, an error of omission. Hence, in diagnosis, we should prefer completeness over soundness. Furthermore, there is nothing particularly special about minimality that warrants a sound algorithm. Minimality is only one of several conceivable plausibility criteria. Coherency is another plausibility criterion that happens to be closely related to minimality but can be considered a standard in its own right, especially when performing decompositional search.

Chapter 4

Decompositional Search

> *After having decomposed the problem, we try to recombine its elements in some new manner .... There are, of course, unlimited possibilities of recombination. Difficult problems demand hidden, exceptional original combinations, and the ingenuity of the problem-solver shows itself in the originality of the combination.*
>
> — George Polya, *How to Solve It* (1945)

Creative problem solving is often seen as a divine or inspirational process. As Polya [56, p. 73] remarks, one criterion for creative problem solving is finding original combinations of subproblems. Although we do not claim that a routine algorithm will meet Polya's standard for originality, our method of diagnostic reasoning matches the decompositional style he advocates. In the previous chapter, we discussed how to formulate differentials, once a problem decomposition has been generated. In this chapter, we present a search process to generate plausible problem decompositions.

4.1 Search Trees

The search tree for decompositional search is similar to the recursive method for generating partitions. In this method, suppose we have a partition containing $|\mathcal{C}|$ clusters. Then, a new positive symptom can added to one of the existing $|\mathcal{C}|$ clusters, or added as a singleton cluster to create a partition containing $|\mathcal{C}| + 1$ clusters. This means that there are two ways to expand an existing partition with a new symptom: either assign it to an existing cluster or adjoin a new cluster with the symptom by itself. Together, these two expansion operators are sufficient to generate all possible partitions.

However, for reasons explained in the previous chapter, we are interested not in partitions but in decompositions, which allow symptoms to appear in more than one cluster. Problem decompositions create a potential search space, shown in figure 4-1, and complicates the search process somewhat. The search process involves three major steps: assignment, ambiguation, and disambiguation. These three steps are repeated for each positive symptom in the diagnostic case. We will discuss each of these steps in the following sections.

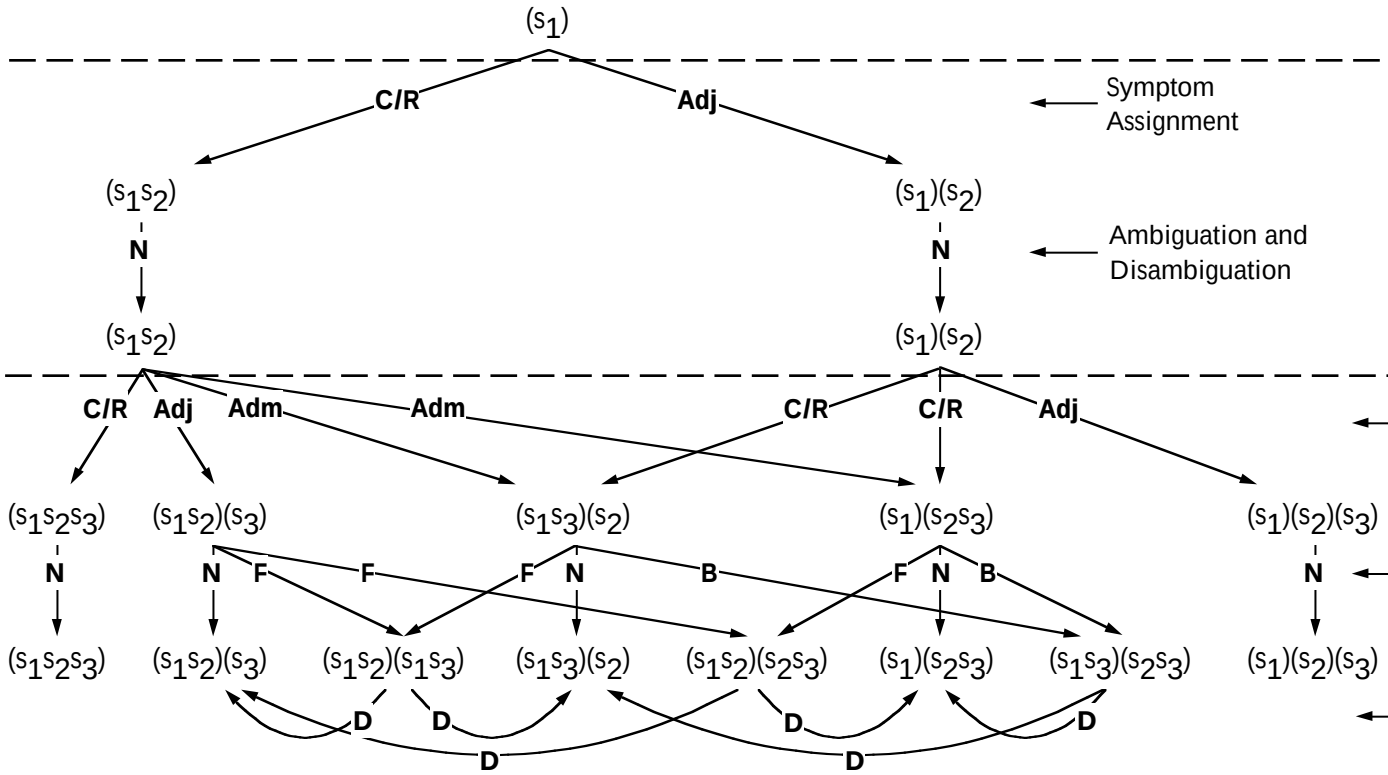

Figure 4-1 Decompositional search space. This figure shows the complete decompositional search space for up to three symptoms. appliesArrows are labeled according to the operator applied: C/R = covering or restricting, Adj = adjoining, Adm = admixing, F = forward ambiguation, B = backward ambiguation, N = no ambiguation, and D = disambiguation. The arrows show only possible transformations. Not all combinations of transformations are possible in the same problem. For example, backward ambiguation is applicable only after the covering operator.

4.2 Symptom Assignment

The first step in decompositional search is symptom assignment. There are four ways to assign a symptom to an existing decomposition: covering, restricting, adjoining, and admixing. The first two operators assign a symptom to an existing cluster, while the other two operators use the new symptom to create a new cluster. This contrasts with the method for generating partitions, which requires only two operators. The operators for decompositional search are summarized graphically in figure 4-2.

The covering and restricting operators have the same effect, assigning a new positive symptom s to an existing cluster C. They differ only depending on whether the possible causes for s subsume the common causes for C. When the possible causes for s subsume the common causes for C, the symptom *covers* the cluster; otherwise, it *restricts* the cluster. The difference is illustrated in the top two rows of figure 4-2. When a symptom s covers a cluster C, the common causes for the cluster remain unchanged when s is added to C. On the other hand, when s restricts C, the common causes become smaller.

The distinction between covering and restricting will recur in our presentation of decompositional search. We define the two notions as follows:

Definition 9 *A symptom* s covers *a cluster* C *if*

$$\text{Causes}(C - \{s\}) \;=\; \text{Causes}(C \cup \{s\})$$

Otherwise, a symptom s restricts *a cluster* C *if*

$$\text{Causes}(C - \{s\}) \;\subset\; \text{Causes}(C \cup \{s\})$$

These definitions essentially constitute a perturbation test. If the symptom already belongs to a cluster, remove the symptom and see whether the set of possible causes for that cluster enlarges. Or, if the symptom does not belong to the cluster, add the symptom to the cluster and see whether the set of possible causes for the cluster shrinks. Any change indicates that the symptom restricts the cluster.

The adjoining operator creates a new singleton cluster containing only the new symptom. For example, given decomposition (s_1) and a new symptom s_2, adjoining would add a new cluster (s_2) to give the decomposition (s_1) (s_2). This operation is shown in the third row of figure 4-2.

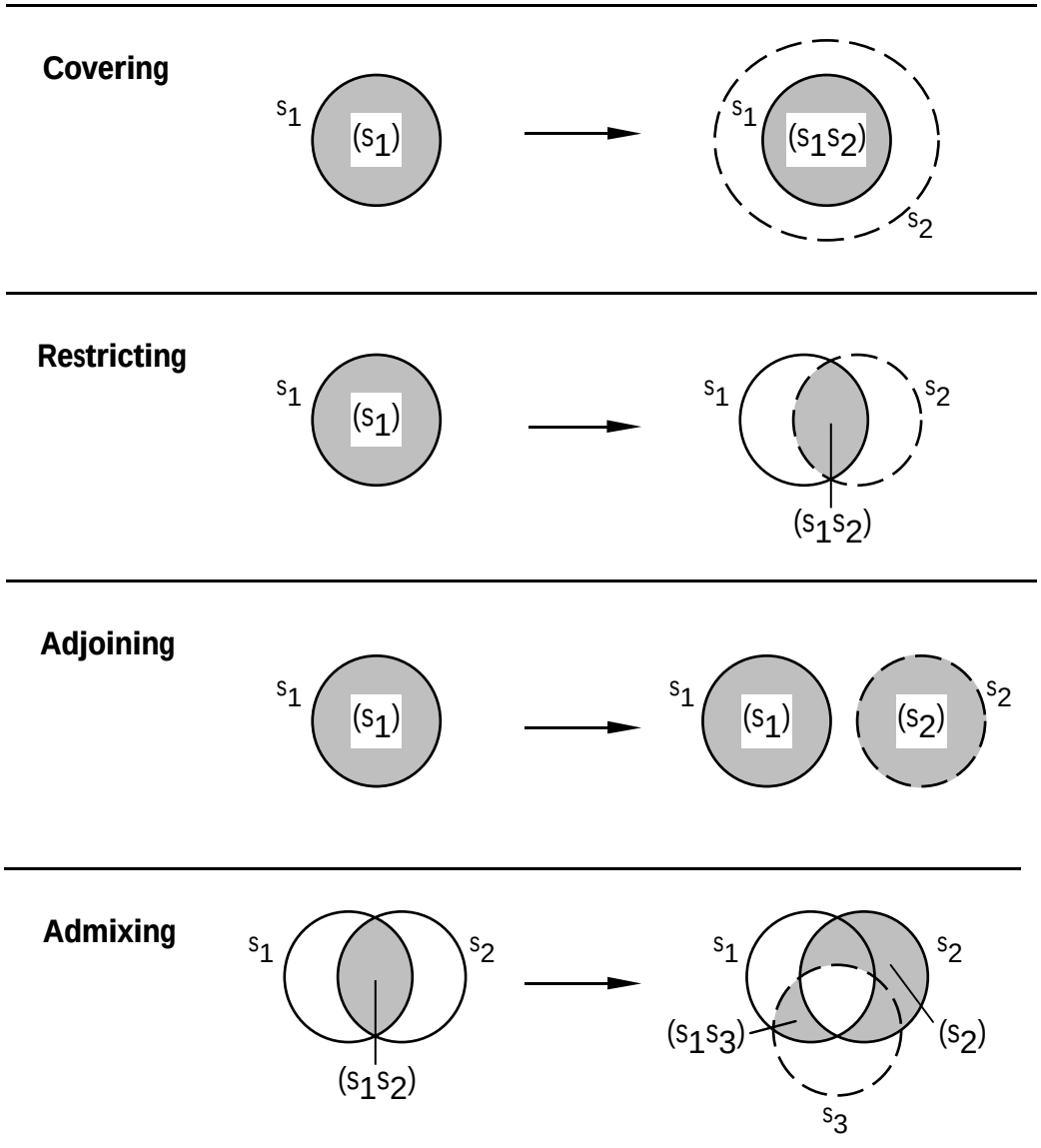

Figure 4-2 Assignment operators for decompositional search. Each row illustrates a different operator: covering, restricting, adjoining, and admixing.

The admixing operator also creates a new cluster, but one containing both the new symptom and one previously assigned. For example, given decomposition $(s_1 s_2)$ and new symptom s_3, the admixing operator could admix s_3 with either symptom s_1 or s_2. If s_1 were admixed with s_3, the new cluster would be $(s_1 s_3)$ and the resulting decomposition would be $(s_1 s_2)$ (s_2), as shown in the last row of figure 4-2. The symptoms that are eligible for admixing are those that restrict their cluster and also restrict the new symptom:

$$\textsc{Admixable}(s', C) \quad = \quad \{s \in C \mid \text{Restricts}(s, C) \wedge \text{Restricts}(s, (s'))\}$$

where s' is the new symptom.

Admixing provides alternate pathways for generating coherent decompositions. These alternate pathways are useful because an incoherent decomposition may have descendants that are coherent. Hence, admixing allows the coherent descendants to be generated by a different pathway. An example of the need for admixing is shown in figure 4-3. In this figure, the coherent decomposition (s_1) (s_2) $(s_3 s_4)$ cannot be generated by covering, restricting, or adjoining. Of these three operators, the only one potentially applicable, restricting, cannot be performed because the required parent, (s_1) (s_2) (s_3), is incoherent and would have been pruned by the decompositional search algorithm. However, admixing provides an alternate pathway from the parent $(s_1 s_3)$ (s_2), which is coherent.

The problem arises in part because symptom s_3 is presented before s_4. Because s_3 covers those disorders that cause s_1 and not s_2, the cluster (s_1) cannot exist as long as (s_2) also exists. But s_4 restricts the possible causes for s_3, so that it no longer covers the unique causes of s_1. This allows (s_1) to exist. Thus, if s_4 were presented before s_3, the decomposition (s_1) (s_2) (s_4) would be coherent, and s_3 could simply restrict cluster (s_4) to create the desired decomposition. Admixing, then, is a way to remedy anomalies resulting from symptom ordering. By combining the new symptom with previously assigned ones, admixing can pair symptoms that might not otherwise pair.

Given these four operators, the algorithm for assigning a new symptom to an existing decomposition is shown in figure 4-4. This algorithm first checks to see if the new symptom s' covers any clusters in the decomposition. If so, it adds s' to all clusters that it does cover and returns the single decomposition. When covering is possible, the other assignment operators are unnecessary.

Otherwise, the algorithm performs restricting, adjoining, and admixing. First, the restricting operator examines every cluster C in the decomposition.

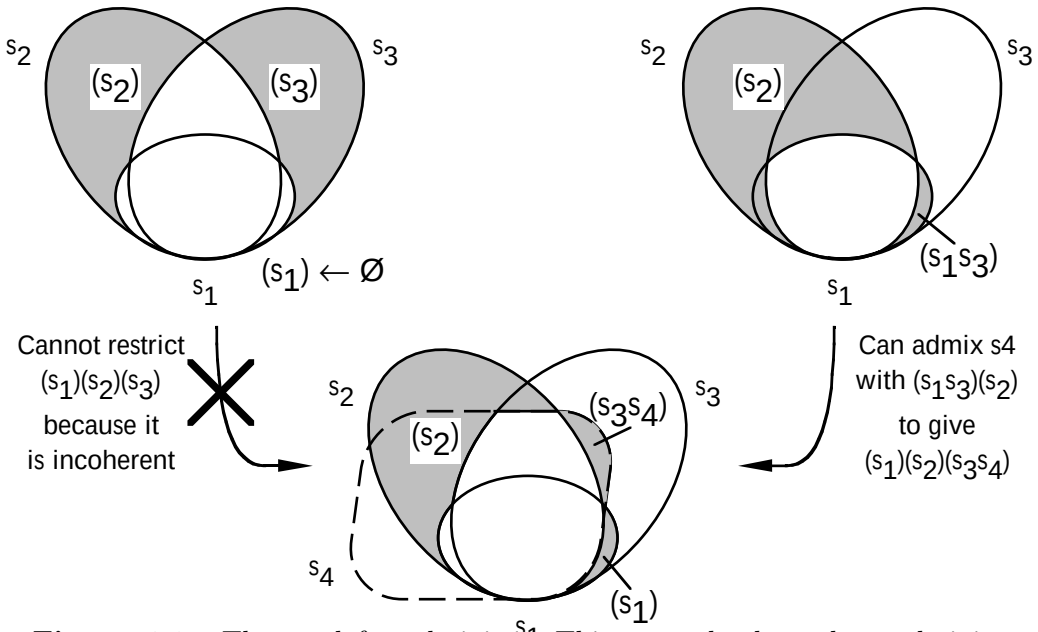

Figure 4-3 The need for admixing. This example shows how admixing can generate a decomposition, (s_1) (s_2) $(s_3 s_4)$ that cannot be generated by any other symptom assignment operator. Restriction of the decomposition (s_1) (s_2) (s_3) will not work because that decomposition is incoherent and thereby pruned by the decompositional search algorithm.

Algorithm 3 (Symptom Assignment)

Procedure ASSIGN (New symptom s', Decomposition $\mathcal{C}$)

```
 1  If symptom s' covers some cluster C ∈ 𝒞 then
 2      Create a new decomposition: 𝒞' ← 𝒞
 3      For every cluster C ∈ 𝒞 that s' covers do
 4          Cover C: C' ← C ∪ {s'};  𝒞' ← SUBSTITUTE(C', C, 𝒞')
 5      Return a singleton set containing the new decomposition: {𝒞'}
 6  Else do
 7      Initialize the set of new decompositions: F ← ∅
 8      For every cluster C ∈ 𝒞 do
 9          If Causes(s) ∩ Causes(C) ≠ ∅ then
10              Restrict C: C' ← C ∪ {s'};  𝒞' ← SUBSTITUTE(C', C, 𝒞)
11              Ambiguate, disambiguate, and add the restricted cluster:
                    F ← F ∪ {DISAMBIGUATE(AMBIGUATE(C', 𝒞'))}
12      Adjoin s': C' ← (s');  𝒞' ← 𝒞 ∪ {C'}
13      Ambiguate, disambiguate, and add the adjoined cluster:
            F ← F ∪ {DISAMBIGUATE(AMBIGUATE(C', 𝒞'))}
14      For every cluster C ∈ 𝒞 do
15          For every admixable symptom s ∈ ADMIXABLE(s', C) do
16              Remove s from its previous cluster:
                    𝒞' ← SUBSTITUTE(C − {s}, C, 𝒞)
17              Admix s and s': C' ← (ss');  𝒞' ← 𝒞' ∪ {C'}
18              Ambiguate, disambiguate, and add the admixed cluster:
                    F ← F ∪ {DISAMBIGUATE(AMBIGUATE(C', 𝒞'))}
19      Return the set of new decompositions: F
```

Procedure SUBSTITUTE (New cluster C', Old cluster C, Decomposition $\mathcal{C}$)

```
20  Initialize a new decomposition: 𝒞' ← ∅
21  For each cluster C'' ∈ 𝒞 do
22      If the cluster is the old one [C'' = C] then
23          Add the new cluster: 𝒞' ← 𝒞' ∪ {C'}
24          Compute its common causes: Causes(C') ← ∪_{s∈C'} Causes(s)
25      Else do
26          Copy the cluster unmodified: 𝒞' ← 𝒞' ∪ {C''}
27  Return the new decomposition 𝒞'
```

Figure 4-4 Algorithm for symptom assignment. This algorithm contains executes the covering, restricting, adjoining, and admixing operators and applies ambiguation and disambiguation steps when appropriate.

If the possible causes for the new symptom s' have a nonempty intersection with the common causes for cluster C, then a new decomposition is created with s' added to C. The new decompositions are then ambiguated and disambiguated, using procedures that are discussed later in this chapter. Second, the adjoining operator creates a new cluster (s') and creates a new decomposition with (s') adjoined to the old decomposition. As with restricting, this decomposition is ambiguated and disambiguated. Finally, admixing finds every previously assigned symptom s that can be admixed with the new symptom. Admixing is possible when s restricts its current cluster and also restricts the singleton cluster (s'). When admixing is possible, a new decomposition is created, ambiguated, and disambiguated. The assignment procedure then returns the entire set of new decompositions obtained by restricting, adjoining, and admixing. This set of decompositions can be incorporated into the new frontier of decompositions in the search tree.

4.3 Ambiguation

A problem decomposition can assign a symptom to more than one cluster. This signifies that the assignment of the symptom is arbitrary and does not affect any commonality or disjointness constraints. We say that a symptom s is *ambiguous* with respect to a decomposition $\mathcal{C}$ if there exists more than one cluster $C \in \mathcal{C}$ such that s covers C. Note that ambiguity is defined using the common cause set and not the differential diagnosis. We use common cause sets because at this point the assignments of potentially ambiguous symptoms are still tentative, so differential diagnoses have not yet been formulated.

As we discussed in section 3.2.2, a decomposition with an ambiguous symptom is said to be ambiguous; otherwise, it is instantiated. Conceptually, an ambiguous decomposition represents a set of instantiated decompositions with the same commonality and disjointness constraints. An ambiguous decomposition thereby expresses symptom assignments at a higher level of abstraction, focusing on those symptoms that affect the differential diagnoses. Ambiguous symptoms do not affect the differentials or even the common cause sets because they have become too general compared to the existing clusters. The goal, then, is to ambiguate as much as possible, a notion that is called *maximal ambiguity*:

Definition 10 *A decomposition $\mathcal{C}$ is* maximally ambiguous *for a set P of*

Algorithm 4 (Ambiguation)

Procedure AMBIGUATE (Modified cluster C', Decomposition $\mathcal{C}$)

```
1  For all unmodified clusters C ≠ C' do
2      For all symptoms s in C do
3          If s covers the modified cluster C' then
4              Copy the symptom to the modified cluster: C' ← C' ∪ {s}
```

Figure 4-5 Algorithm for ambiguation. This algorithm checks all symptoms in the unmodified clusters and copies them to the modified cluster whenever they cover it.

positive symptoms if for every positive symptom s in P the following conditions hold:

1. *If s restricts its cluster, it belongs only to that cluster.*
2. *Otherwise, if s covers its cluster, it belongs to all clusters that it covers.*

Maximal ambiguity can be thought of as a canonical representation for sets of similar problem decompositions. Maximal ambiguity is like a "least-commitment" strategy [61, 62] that makes only critical symptom assignments and avoids arbitrary ones.

The opportunity for ambiguation occurs whenever a cluster receives a new symptom or when a new cluster is created. In either case, we refer to cluster with the new symptom as the *modified* cluster. There are two ways that ambiguation can occur: (1) The new symptom may cover more than one existing cluster; we call this *backward ambiguation*. (2) Previously assigned symptoms may cover the modified cluster; we call this *forward ambiguation*.

Since backward ambiguation applies only when the covering operator is executed, this step is performed in the symptom assignment procedure (lines 3–4 of Algorithm 3). The algorithm for forward ambiguation (called simply "ambiguation" when the context is clear), however, requires a separate procedure, which is presented in figure 4-5. This procedure takes two arguments: the modified cluster and the decomposition to be ambiguated. The procedure examines all symptoms in the unmodified clusters. Any symptom that covers the new cluster C' is copied to it, thereby ambiguating the symptom.

4.4 Disambiguation

After ambiguation, a decomposition may not meet the definition for maximal ambiguity. Therefore, we require a final step, called disambiguation, to correct such situations. Disambiguation "undoes" some assignments of ambiguous symptoms. The need for disambiguation arises from the forward ambiguation step. In that step, symptoms that cover the modified cluster are copied to that cluster. If these symptoms restrict their previous cluster, they need to be removed from that cluster in order to satisfy maximal ambiguity. Furthermore, if these symptoms are removed from their clusters, a secondary need for disambiguation may arise. This need arises because removing restricting symptoms from a cluster enlarges the common causes for that cluster. Hence, an ambiguous symptom that previously covered that cluster may now no longer cover it. The disambiguation step, presented in figure 4-6, takes care of both of these circumstances.

This procedure finds the ambiguous symptoms in a decomposition, making use of the DUPLICATES procedure, defined previously in Algorithm 1. It then removes these symptoms and recomputes the common causes for each cluster when necessary. Finally, for each previously ambiguous symptom, the procedure finds the set of clusters that it now covers and restores it to those clusters.

An example of disambiguation is shown in figure 4-7. The initial decomposition is (`Fever`,`Cough`), which has a differential of {`Tb`}. When symptom `Wheeze` is adjoined, a new decomposition (`Fever`,`Cough`) (`Wheeze`) is generated. The new cluster provides an opportunity for `Cough` to cover it, so the ambiguation procedure copies `Cough` to the new cluster. However, `Cough` still restricts its old cluster. Disambiguation detects this situation and removes `Cough` from its old cluster. The resulting decomposition is therefore (`Fever`) (`Wheeze`,`Cough`).

Note that disambiguation enlarges common cause sets and differentials. In the example above, `Cough` restricted its old cluster, yielding only `Tb` in its differential. After disambiguation, though, the old cluster had more disorders in its differential, namely, `Tb`, `Flu`, and `Mal`. Effectively, the symptom `Cough` was moved from a cluster that it restricted to a cluster that it covered. This process of moving symptoms between clusters can be interpreted as freeing unnecessary constraints. The combination of ambiguation and disambiguation frees constraints placed by unnecessarily restricting symptoms. As new symptoms are processed, new opportunities for freeing constraints may arise.

Algorithm 5 (Disambiguation)

Procedure DISAMBIGUATE (Decomposition $\mathcal{C}$)

```
 1  Find all ambiguous symptoms: Ambiguous(C) ← DUPLICATES(C)
 2  For all clusters C ∈ C do
 3      If C contains ambiguous symptoms [C ∩ Ambiguous(C) ≠ ∅] do
 4          Remove the ambiguous symptoms: C ← C − Ambiguous(C)
 5          Recompute its possible causes: Causes(C) ← ∩_{s∈C} Causes(s)
 6  For all ambiguous symptoms s ∈ Ambiguous(C) do
 7      Initialize its set of reassignments: Reassign ← ∅
 8      For all clusters C ∈ C do
 9          If s covers C then
10              Add the cluster to the set of reassignments:
                    Reassign ← Reassign ∪ {C}
11      If no reassignments are possible [Reassign = ∅] then
12          Return "degenerate"
13      Else do
14          For all reassigned clusters C ∈ Reassign do
15              Reassign the symptom to the cluster: C ← C ∪ {s}
16  Return the decomposition C
```

Figure 4-6 Algorithm for disambiguation. This procedure removes all ambiguous symptoms, recomputes common cause sets as necessary, and reassigns the symptoms to clusters they cover.

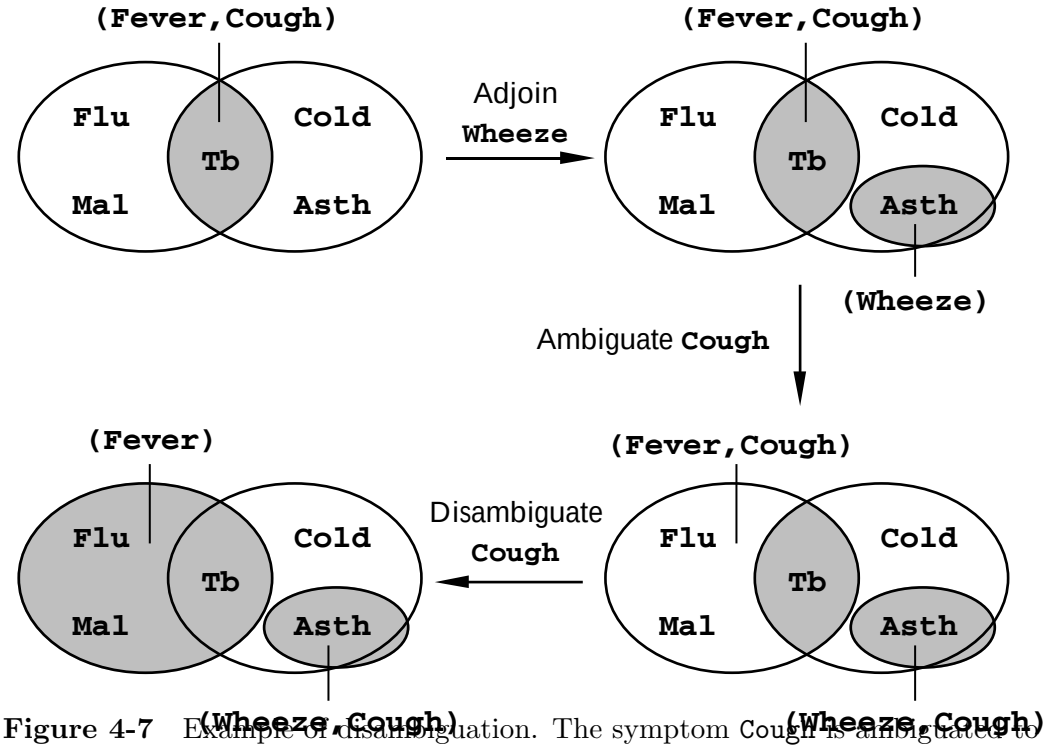

Figure 4-7 Example of disambiguation. The symptom `Cough` is ambiguated to the new cluster (`Wheeze`), but it restricts its previously assigned cluster. Disambiguation removes `Cough` from its previous cluster, thereby freeing constraints on its common causes and differential.

Disambiguation does not always work this well. Note that disambiguation reassigns each ambiguous symptom to fewer clusters than before. In the extreme, or *degenerate*, case, an ambiguous symptom may not assignable to any cluster. Degenerate cases can occur when an ambiguous symptom s exists and the disambiguation step removes a symptom from every cluster that s originally covered. Then, those clusters will have an enlarged set of common causes, and it is possible that s will no longer cover any of them. When a previously ambiguous symptom cannot be reassigned, the decomposition is said to be degenerate. Degeneracy indicates that the ambiguation step freed too many constraints. An ambiguous symptom must then be assigned arbitrarily to place another constraint. In such cases, our procedure simply discards the decomposition as degenerate, since other symptom assignments will yield the desired decompositions.

An example of a degenerate decomposition is shown in figure 4-8. The original decomposition is $(s_1 s_3 s_5)$ $(s_2 s_4 s_5)$. Symptom s_6 is then adjoined as a new cluster, (s_6). Since symptoms s_1 and s_2 cover the new cluster, the ambiguation procedure copies them to the new cluster. The decomposition at this point is $(s_1 s_3 s_5)$ $(s_1 s_6 s_2)$ $(s_2 s_4 s_5)$. The disambiguation step then removes the ambiguous symptoms, s_1, s_2, and s_5; recomputes the common cause sets; and reassigns the symptoms. Symptoms s_1 and s_2 are found to restrict their old clusters, so they are removed from those clusters. But symptom s_5 is found to restrict both of its old clusters, so it cannot be assigned to any cluster. Hence, the decomposition is degenerate.

Together, the symptom assignment, ambiguation, and disambiguation procedures produce a set of maximally ambiguous decompositions. This proposition is presented in the following theorem.

Theorem 8 *Algorithms 3, 4, and 5 terminate. Moreover, given a maximally ambiguous decomposition and a symptom, every decomposition produced by the algorithms is maximally ambiguous.*

Proof The algorithms loop over only finite sets; hence they terminate. For the rest of the theorem, we assume that all previously assigned symptoms s in decomposition $\mathcal{C}$ satisfy the definition for maximal ambiguity and consider a new symptom s', which results in a new decomposition $\mathcal{C}'$. If s' covers some cluster, then lines 3 and 4 of Algorithm 3 ensures that s' is maximally ambiguated. Since no common cause sets are changed, the maximal ambiguity of all symptoms s is preserved. Otherwise, s' restricts every cluster, so it

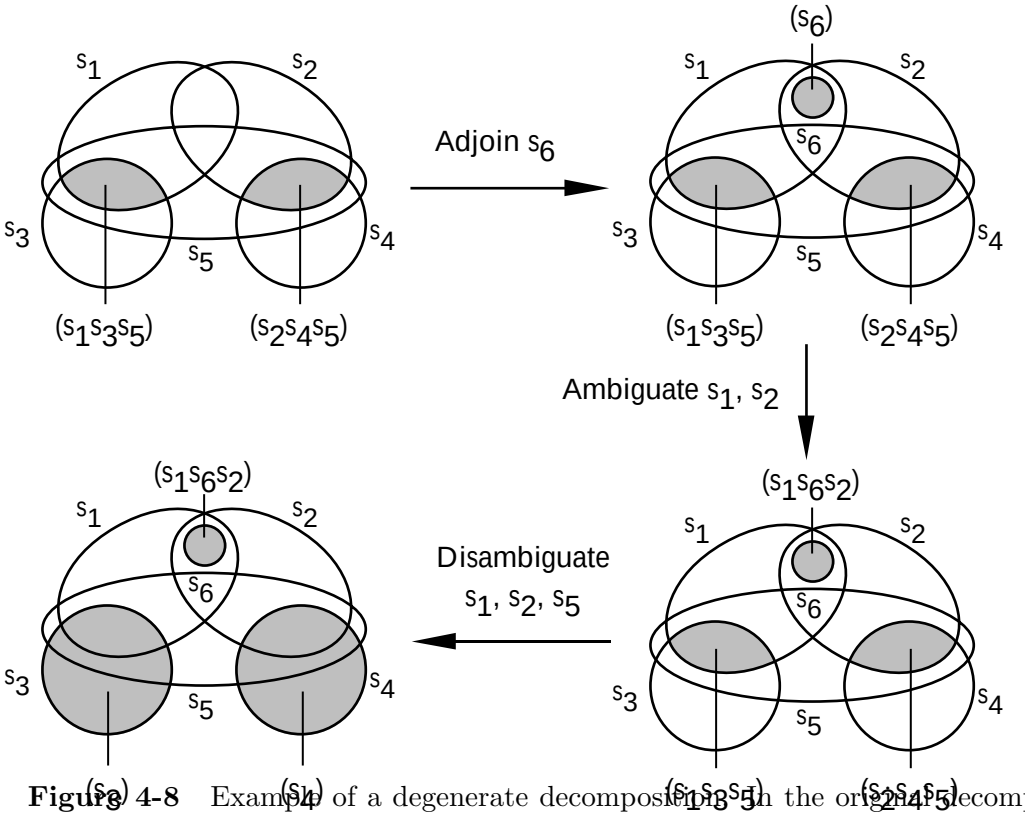

Figure 4-8 Example of a degenerate decomposition. In the original decomposition, symptom s_5 covers two clusters. Decomposition $(s_1 s_3 s_5)$ (s_6) $(s_2 s_4 s_5)$ is then created by adjoining, and symptoms s_1 and s_2 are ambiguated to the new cluster. Disambiguation removes s_1 and s_2 from the previous clusters, but s_5 no longer covers any cluster. The resulting decomposition is degenerate.

is assigned to only one cluster by the restricting, adjoining, and admixing operators, thereby also satisfying maximal ambiguity.

As for the previously assigned symptoms s, the only new opportunity for covering is the cluster C' containing s'. The other clusters $C \neq C'$ can only lose elements by Algorithm 5, so their common cause sets enlarge, creating no new opportunities for covering. Algorithm 4 ensures that if s covers the new or modified cluster it is placed in that cluster. Algorithm 5 ensures that all ambiguous symptoms satisfy the definition of maximal ambiguity. All symptoms not examined by Algorithm 5 covered no other clusters originally and continue to cover no other clusters, because the common causes of clusters C will have enlarged or remained the same; they do not cover C', otherwise Algorithm 4 would assign them to more than one cluster and they would have been examined by Algorithm 5. Hence the maximal ambiguity of all previously assigned symptoms is preserved. ∎

4.5 Search Strategy

The symptom assignment, ambiguation, and disambiguation procedures can explore the space of coherent decompositions when orchestrated by a suitable search strategy. There are several ways that search could be performed [52]. A breadth-first search process would explore the entire search space of plausible decompositions. Of course, depending on the particular application, we may not want to to do this. In such cases, we could use other strategies, such as depth-first search or beam search.

However, in this thesis, we are interested in comparing the efficiency of two algorithms. The breadth-first paradigm allows us to examine and compare the relative sizes of the two different search spaces. Hence, our implemented system, SYNOPSIS, is designed to use a breadth-first search strategy. For actual diagnostic problem solving, the system may be extended to use other search strategies.

In figure 4-9, we present a breadth-first search algorithm for computing all coherent decompositions for a set of positive symptoms. This algorithm generates problem decompositions using the symptom assignment procedure. That procedure, in turn, executes the covering, restricting, adjoining, and admixing operators, and calls the ambiguation and disambiguation procedures as necessary. At this point, duplicate decompositions may have been gener-

Algorithm 6 (Breadth-first Search)

Procedure DIAGNOSE (Positive symptoms P)

```
 1  Initialize the frontier: F ← ∅
 2  For every positive symptom s ∈ P do
 3      Expand the frontier by the symptom: F ← EXPAND(s, F)
 4  Return the frontier F
```

Procedure EXPAND (Positive symptom s, Frontier $\mathcal{F}$)

```
 5  Initialize a temporary frontier: F' ← ∅
 6  For every decomposition C in frontier F do
 7      Collect new decompositions: F' ← F' ∪ ASSIGN(s, C)
 8  Remove duplicate decompositions: F' ← REMOVE-DUPLICATES(F')
 9  Initialize the new frontier: F'' ← ∅
10  For every new decomposition C ∈ F' do
11      If C is not "degenerate" then
12          Formulate its differentials: C ← FORMULATE-DIFFERENTIALS(C)
13          If C is not "incoherent" then
14              Collect it: F'' ← F'' ∪ {C}
15  Return the new frontier F'
```

Figure 4-9 Algorithm for breadth-first decompositional search. The top-level procedure processes each symptom sequentially, resulting in a new frontier. The EXPAND procedure expands each decomposition in the old frontier, removes duplicate decompositions, and formulates the differentials of the remaining ones. Incoherent decompositions are pruned from the search tree.

ated and are therefore pruned. The disambiguation step may also discover degenerate decompositions, which are also pruned. The remaining decompositions are then submitted for formulation of their differentials. The process of differential formulation was presented in chapter 3. The incoherent decompositions are then pruned and the remaining, coherent decompositions are kept. This process repeats, yielding a frontier of coherent decompositions for each positive symptom in the given set.

The search algorithm uses the function REMOVE-DUPLICATES to remove duplicate elements from a set of sets. The actual implementation of this function depends how sets are represented in the diagnostic system. An efficient implementation of REMOVE-DUPLICATES for sets represented as bit vectors is presented in appendix A.

4.6 Incompleteness

The breadth-first search algorithm attempts to compute a complete set of coherent decompositions, and in practice, it generally succeeds. However, theoretically, the decompositional search algorithm is incomplete. That is, it may fail to generate a decomposition that is nevertheless coherent. An example of incompleteness is provided in the following example.

Example Consider the situation shown in figure 4-10. The desired decomposition is (s_3) (s_4) $(s_1 s_2 s_5)$, which is coherent. However, this decomposition cannot be generated by the algorithm when the symptoms are processed in the order: s_1, s_2, s_3, s_4, s_5. The only possible parent decomposition that could generate the desired decomposition is (s_3) (s_4) $(s_1 s_2)$, by using the restricting operator. However, this parent is not coherent, so it would have been pruned by the algorithm. Hence the desired decomposition cannot be generated. ∎

Interestingly, the desired clustering in this example can be generated if the symptoms are processed in another order. Incompleteness arises in part from a greedy solution of the first four symptoms. Then the algorithm cannot recover from this greediness when processing the fifth symptom. Normally, admixing helps the abductive decomposition algorithm recover from greediness, but even admixing is not sufficient in this case. Admixing combines the new symptom with only one previously assigned symptom. In this case,

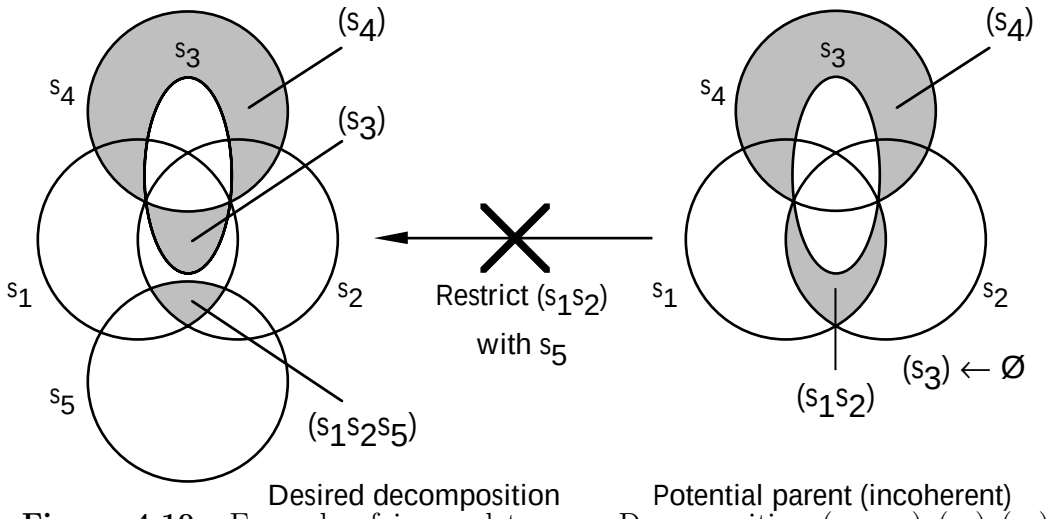

Figure 4-10 Example of incompleteness. Decomposition $(s_1 s_2 s_5)$ (s_3) (s_4), which is coherent, cannot be generated by the decompositional search algorithm when symptoms are processed in the order: s_1, s_2, s_3, s_4, s_5. The only potential parent decomposition, $(s_1 s_2)$ (s_3) (s_4) is incoherent because cluster (s_3) has a null differential.

a more powerful admixing operator, one in which two previously assigned symptoms are combined with the new symptom, could generate the desired decomposition. However, such an operator would probably be computationally expensive, because there are combinatorially many ways to select pairs or larger subsets of previously assigned symptoms.

Thus, the abductive decomposition is theoretically incomplete, although we have found that it almost always gives complete sets in practice. The algorithm could therefore be classified as a greedy or heuristic algorithm. But the algorithm is not terribly greedy, and the alternative pathways help make the algorithm robust. In any case, incompleteness should perhaps expected with algorithms that solve computationally intractable problems such as multidisorder diagnosis, since a complete algorithm for multidisorder diagnosis would be computationally expensive. Nevertheless, the conditions required for incompleteness are quite rare in practice, and as we shall see in the next chapter, decompositional search is fairly robust.

Chapter 5

Experimental Comparison

Nullins in verba (Don't take anyone's word for it).

— Motto of the Royal Society of London (1660)

In this chapter, we compare the decompositional search and candidate generation algorithms empirically. Although much of our motivation for decompositional search is based on intuitive ideas about implicit representation and causal equivalence, it is difficult to predict the actual computational behavior of an algorithm. Moreover, most theoretical analysis deals with worst-case performance and offers little to say about the average case. We therefore investigate the computational behavior of decompositional search through a series of four experiments. The first experiment compares the decompositional search and candidate generation algorithms on a wide variety of problems. The second experiment explores characteristics of the decompositional search algorithm. The third experiment studies issues of symptom presentation and ordering. The final experiment compares the algorithms on more complex problems.

5.1 Case Selection

Our experiments draw upon the QMR medical knowledge base [38]. We chose the medical domain because of its preponderance of complex, multidisorder problems. Multiple disorder problems occur often in medicine. According to a 1987 study by the National Center for Health Statistics [22], hospital patients have an average of 3.1 diagnoses at discharge. The complexity of the medical domain is reflected in the size of the QMR knowledge base, containing approximately 4000 symptoms and 600 diseases. QMR therefore covers 80 percent of the diseases encountered in general internal medicine. The QMR knowledge base constitutes a natural test of how well diagnostic algorithms can scale up to real-world domains.

We modified the QMR knowledge base by excluding symptoms that provide contextual evidence:

Age 16 to 25	Race Black	Sex Female
Age 26 to 55	Race Oriental	Sex Male
Age Greater than 55	Race White	

Contextual evidence does not represent effects of disease but rather causal predispositions to disease. Thus, we removed these anomalous symptoms

from the QMR knowledge base. Another important reason for removing contextual evidence was that each one is used in QMR as a placeholder for storing epidemiologic or probabilistic information about each disease, when relevant. Consequently, contextual symptoms were linked to almost the entire universe of 600 diseases in the QMR knowledge base as a possible "cause". Although decompositional search handles such large sets well, they would have handicapped the candidate generation algorithm severely.

The general strategy underlying our experiments is to generate cases at random and then diagnose them with both algorithms, comparing the time and space required to solve the problems. We generated cases using a stochastic model. In this model, we first selected one or more disorders called targets, which were assumed to be present, although of course this assumption was not known to the diagnostic algorithm. These target disorders were then used to generate a set of symptoms. Symptoms were selected based on their conditional likelihood of being caused by the target disorder, using the frequency values in the QMR knowledge base. These frequency values, which range from 1 to 5, have been shown to correspond most closely with link probabilities, with the following mapping [26]:

Frequency	Link probability
1	.03
2	.20
3	.50
4	.80
5	.97

A link probability specifies the probability that a symptom will be caused by a disorder, given that the disorder is present. For a symptom to be selected, its link probability had to exceed a random variable distributed uniformly between 0 and 1.

The decompositional search and candidate generation algorithms were implemented in ANSI Common Lisp [70] and compiled using Lucid Lisp. Compilation was optimized for execution speed. We executed the algorithms on a Sun SparcStation 2 with 48 megabytes of random access memory and 100 megabytes of virtual memory available to the Lisp process. For each run, we measured the number of nodes expanded, the number of nodes kept, and running time. Since the algorithms process symptoms sequentially, the order of the symptoms in a case was an important variable. So far, we have discussed cases as unordered sets of positive and negative symptoms. When

we consider the set of symptoms to be an ordered sequence, we call the sequence of symptoms a *case ordering*.

5.2 Single-Target Cases

For the first experiment, we generated a series of case orderings, each based on a single target disorder. For each ordering, we selected a disorder at random from the QMR knowledge base. Each disorder generated 7 symptoms in the following manner. A symptom s was tentatively selected at random from the possible effects of the disorder d. A random number uniformly distributed between 0 and 1 was also generated. If the link probability for d and s exceeded the random number, then symptom s was selected. Otherwise, another symptom and number were selected at random. This process was repeated until a set of 7 symptoms were selected. To prevent the search trees from expanding too rapidly, only symptoms with fewer than 100 possible causes were chosen.

A total of 100 single-target case orderings were generated in this fashion. The order of the symptoms in each ordering was random. These case orderings were diagnosed by the decompositional search and candidate generation algorithms. However, 14 of the 100 cases could not be solved by the candidate generation algorithm because they exceeded the memory allocated to the Lisp process (100 megabytes). For these cases, we therefore truncated the symptom lists at the point where the machine ran out of memory, resulting in the following case sizes:

	Symptoms				
Truncated after:	3	4	5	6	7
Number of cases:	1	8	2	3	86

Even though the cases were generated by only a single target disease, they constituted a test of multidisorder diagnosis. The single target disorder gave each case at least one single-disorder minimal candidate. But the majority of minimal candidates contained multiple disorders. By the standard of minimality, these multidisorder candidates were as plausible as the single-disorder candidates.

The algorithms were then compared on the set of 100 truncated and complete case orderings. The total running time, number of nodes expanded, and

number of nodes kept were recorded. In addition to these totals, the intermediate results were also recorded. After each symptom was processed, the number of intermediate nodes expanded and kept were recorded. Essentially, these numbers measure the width of the search tree for the two algorithms. The intermediate results for a typical ordering are shown below:

	Symptoms in Case Ordering								
Algorithm	s_1	s_2	s_3	s_4	s_5	s_6	s_7	Total nodes	
Cand.	337	1618	1306	1708	1848	2673	95	9625	expanded
	106	430	732	1012	405	27	73	2825	kept
Decomp.	2	3	7	12	12	4	6	47	expanded
	2	3	6	11	8	3	5	39	kept

As each symptom in the case ordering is processed, the algorithms create a frontier of intermediate nodes, each node representing a candidate or decomposition. The last column shows the total amount of work performed by the two algorithms. This particular search tree shows that decompositional search generates and keeps far fewer nodes at each frontier than candidate generation.

The results above are for only one case ordering. To summarize this run, we can compare the total number of nodes in the search tree, either expanded or kept, and the total amount of time required. These measures correspond to the space and time complexity of the two algorithms. Note that there are two ways to measure space complexity. We will use the number of nodes kept as our measure because it is more implementation-independent than the number of nodes expanded. A clever implementation of an algorithm might be able to predict which nodes not to expand, but any correct implementation of an algorithm cannot alter the number of nodes kept.

Because of the wide variation in magnitude for the different cases, the time and space complexity for the 100 case orderings are represented best on a log-log scatterplot. Scatterplots of the total nodes kept and the total running time are shown in figure 5-1.

The log-log scatterplots show a fairly high correlation between the two algorithms. The correlation coefficients for the space and time complexity graphs are 0.77 and 0.76, respectively. The high correlation indicates that computational complexity depends in large part upon the particular case. What is hard for one algorithm is also hard for the other one. The correlation also indicates the close relationship between candidate generation and

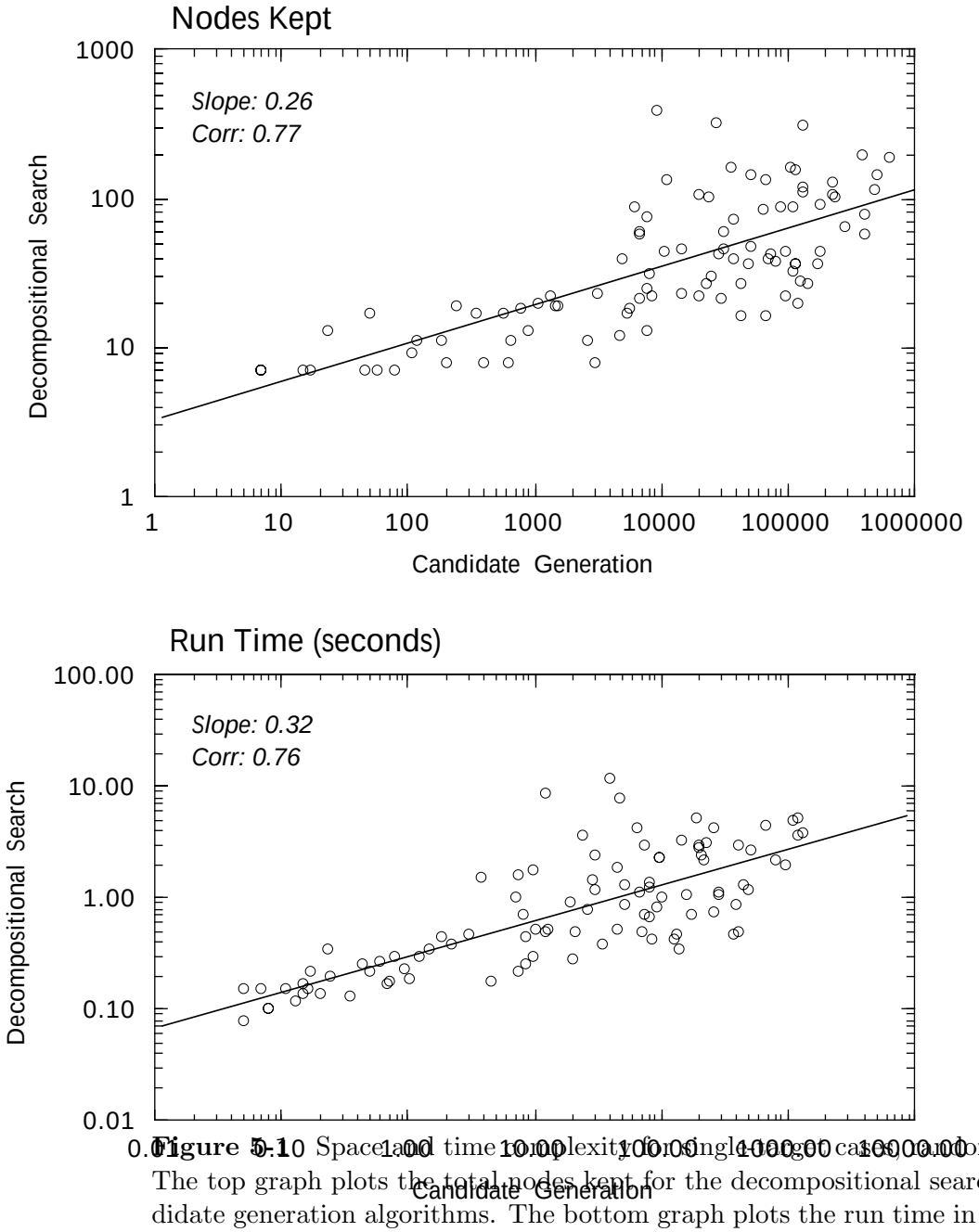

Figure 5.1 Space and time complexity for single target case with random ordering. The top graph plots the total nodes kept for the decompositional search and candidate generation algorithms. The bottom graph plots the run time in seconds for the two algorithms.

decompositional search. The scatterplots also fit the data to a linear form. These linear fits have the following slope-intercept form:

$$\begin{aligned} \ln(k_D) &= 1.17 + 0.26 \ln(k_C) \\ \ln(t_D) &= -1.20 + 0.32 \ln(t_C) \end{aligned}$$

where t_D and k_D are the running time and nodes kept for decompositional search, and t_C and k_C are the same quantities for candidate generation. These equations can be converted to a polynomial form:

$$\begin{aligned} k_D &= 3.22(k_C)^{0.26} \\ t_D &= 0.30(t_C)^{0.32} \end{aligned}$$

Therefore, the y-intercept of the lines changes complexity by only a multiplicative factor. However, the slope changes complexity by an exponential power. The log-log slope, then, best indicates the relationship between the two algorithms, and this slope is reported on each scatterplot. Another way to look at the data is to consider it from the standpoint of candidate generation. The inverse formulas for the above equations are:

$$\begin{aligned} k_C &= 0.0111(k_D)^{3.8} \\ t_C &= 43.2(t_D)^{3.1} \end{aligned}$$

Thus, the inverse slope is another measurement of the comparative efficiency of the decompositional search algorithm. Although we did not show the scatterplot for the total nodes expanded, it shows roughly the same relationship, with a log-log correlation coefficient of 0.81 and a log-log slope of $0.33 \approx 1/3.0$. To save space, we will report the slope for the scatterplot of total nodes expanded without illustrating the graph. To summarize, then, the complexity results are:

Single-target cases, random ordering

	Slope	Inverse slope
Nodes kept	0.26	3.8
Nodes expanded	0.33	3.0
Running time	0.32	3.1

The results above imply that, for the given cases, the space complexity for decompositional search is approximately the cube root or fourth root of that of candidate generation, depending on the criterion used. The time complexity

for decompositional search is approximately the cube-root of that of candidate generation. These relationships represent only polynomial reductions in complexity, so exponential complexity will still dominate in the worst case. But in the given problems the savings are nevertheless substantial, allowing real-world diagnostic problems to be solved in a reasonable amount of time. The savings are evident if we consider that decompositional search solved all of the problems in 10 seconds or less, while candidate generation often required up to 30 minutes.

5.3 Characterizing Decompositional Search

5.3.1 Accuracy

Decompositional search produces nonminimal candidates as well as minimal one in order to achieve a more compact representation. Hence, decompositional search can be considered an approximation of candidate generation, weakening the notion of minimality in return for a gain in efficiency. In order to assess this tradeoff, we now measure the degree of approximation that decompositional search achieves. For this test, we use candidate generation as the standard. There is no reason why decompositional search could not constitute its own standard. But in order to compare the efficiency of algorithms, we should determine that they compute approximately the same answer. The common denominator between the two algorithms is the candidate, since problem decompositions can be converted to candidates, but not vice versa.

To measure the accuracy of decompositional search, we computed the problem decompositions for each of the 100 cases generated above. We then expanded each problem decomposition $\mathcal{C}$ into its candidate set $\text{Cands}(\mathcal{C})$ by computing the Cartesian product of its differentials. The candidate sets for all problem decompositions were then compared with the set MinCands of minimal candidates produced by the candidate generation algorithm. Accuracy was then measured by the following quantities:

- Completeness: What proportion of minimal candidates are produced by decompositional search?

$$\text{Completeness} \quad = \quad \frac{|\cup_{\mathcal{C}} \text{Cands}(\mathcal{C}) \cap \text{MinCands}|}{|\text{MinCands}|}$$

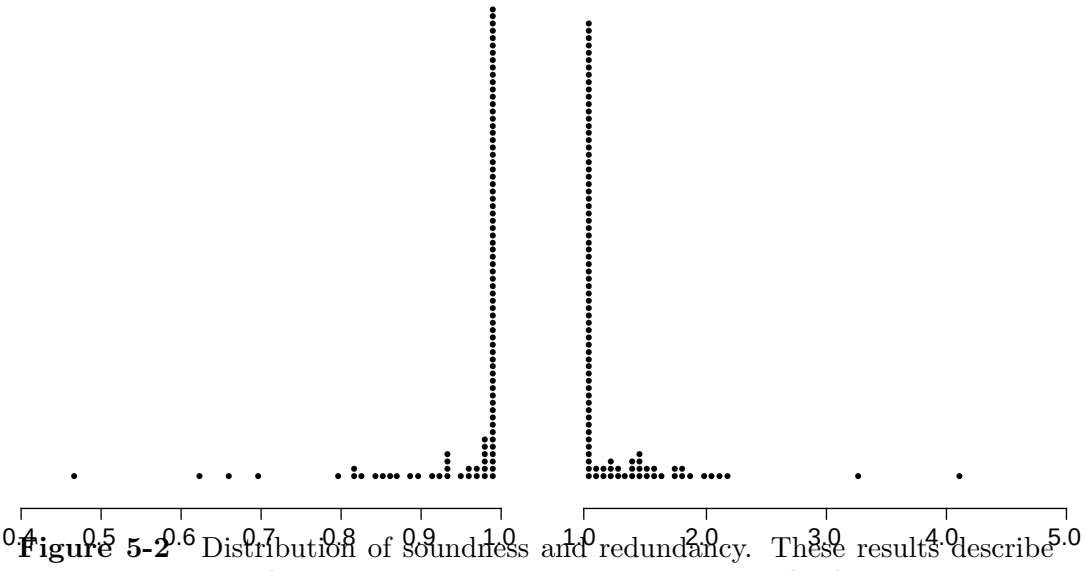

Figure 5-2 Distribution of soundness and redundancy. These results describe the results of the single-target cases with random ordering. The left graph shows the histogram for soundness; the right graph, for redundancy.

- Soundness: What proportion of candidates produced by decompositional search are minimal?

$$\text{Soundness} \quad = \quad \frac{|\cup_{\mathcal{C}} \text{Cands}(\mathcal{C}) \cap \text{MinCands}|}{|\cup_{\mathcal{C}} \text{Cands}(\mathcal{C})|}$$

- Redundancy: How many problem decompositions, on the average, contain a given candidate?

$$\text{Redundancy} \quad = \quad \frac{\sum_{\mathcal{C}} |\text{Cands}(\mathcal{C})|}{|\cup_{\mathcal{C}} \text{Cands}(\mathcal{C})|}$$

For all 100 cases, the decompositional search algorithm was complete from the standpoint of generating minimal candidates. In other words, for all cases, every minimal candidate was generated by at least one problem decomposition. Thus, in practice at least, decompositional search exhibits completeness in this respect.

However, the soundness and redundancy gave more variable results. As expected, decompositional search sometimes produced candidates that were nonminimal, and sometimes produced minimal candidates more than once. The histograms of soundness and redundancy are shown in figure 5-2. The

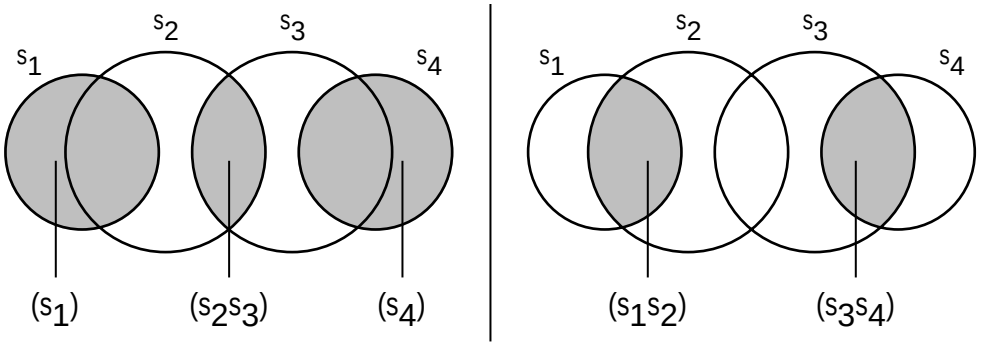

Figure 5-3 Example of nonminimality in decompositional search. The decomposition on the left entails nonminimal candidates. These are nonminimal because they contain the minimal candidates entailed by the decomposition on the right.

histogram on the left shows that, for the vast majority of cases, decompositional search was sound. In other words, almost every candidate in the Cartesian product of the differential was also minimal. But in rare cases, up to 52 percent of these candidates were nonminimal. The worst case occurred in a case of Lymphomatoid Granulomatosis. Decompositional search produced 133 coherent decompositions, which entailed 24172 unique candidates. Candidate generation produced 11527 minimal candidates, meaning that only 48 percent of the candidates produced by decompositional search were minimal. This result reflects the fact that decompositional search only approximates candidate generation, and that it includes nonminimal candidates when necessary to represent the minimal candidates compactly.

This case indicates that certain situations yield decompositions with several nonminimal disorders. One situation occurs when a combination of mutually restricting symptoms also overlap other clusters. This example is shown in figure 5-3. In this figure, the decomposition (s_1) $(s_2 s_3)$ (s_4) contains a restricted cluster $(s_2 s_3)$ as well as two peripheral clusters. The restricting symptoms s_2 and s_3 cover the peripheral clusters, (s_3) and (s_4), to a great extent. Consequently, many of the candidates entailed by the decomposition are nonminimal. Each of these nonminimal candidates contains a candidate entailed by the decomposition $(s_1 s_2)$ $(s_3 s_4)$, which is also shown in the figure. These candidates are all minimal, meaning that many of the candidates entailed by the former decomposition are nonminimal. Note however that the decomposition (s_1) $(s_2 s_3)$ (s_4) still meets the definition of coherency, apart from the minimality standard of the candidate generation approach.

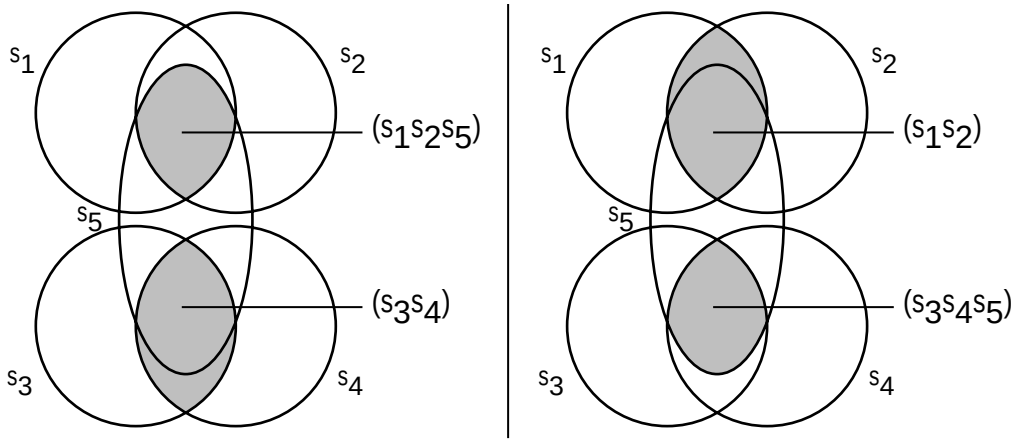

Figure 5-4 Example of redundancy in decompositional search. The two decompositions differ in their assignment of s_5 and overlap in their candidate sets.

The histogram on the right of figure 5-2 illustrates the amount of redundancy for decompositional search. This figure plots the average number of decompositions that entails each candidate. For most cases, the measure of redundancy is approximately 1.0, meaning that each candidate is contained in the candidate set of only one decomposition. However, two cases exhibited high levels of redundancy. The most redundancy came from a case of Chronic Thrombocytopenic Purpura that had 65 coherent decompositions in its final answer, entailing a total of 17770 candidates, of which only 4330 were unique. Of the 4330 unique candidates, 3999 were minimal. The redundancy for this case was therefore $17770/4330 \approx 4.1$. The second most redundancy came from the same highly unsound case of Lymphomatoid Granulomatosis discussed above that had 133 coherent decompositions, entailing a total of 78930 candidates, of which only 24172 were unique. Of the 24172 unique candidates, 11527 were minimal. The redundancy for this case was therefore $78930/24172 \approx 3.3$.

These cases indicate that certain conditions yield decompositions with a high degree of overlap in their candidate sets. Such cases apparently have a symptom that almost, but not quite, covers two or more different clusters, so this symptom cannot be ambiguated, even though it constrains those clusters very weakly. A model for high levels of redundancy is illustrated in figure 5-4. In this figure, assignment of symptom s_5 to either cluster results in two decompositions that are very similar, with entailed candidate sets that have a high degree of overlap.

5.3.2 Robustness

In chapter 4, we showed that decompositional search was theoretically incomplete. That is, it might not generate all coherent decompositions of a given set of positive symptoms. Problems with incoherency arise when symptoms are presented in different orderings. Given the enormous potential search space for decompositions, this is perhaps not surprising. However, in practice, we have found that decompositional search usually gives the same sets of decompositions, regardless of case ordering. Thus, the algorithm is fairly robust in practice.

Of course, the best test for robustness would compare decompositional search against a gold standard of all coherent decompositions for a given problem. Unfortunately, we lack such a gold standard, short of the computationally intractable method of generating every possible decomposition and testing it for coherency. Alternatively, we can test decompositional search to see if it gives the same answers for different permutations of the same set of symptoms.

We performed this test on the same single-target cases used in the first experiment. For each of the 100 cases, we created 10 random orderings and ran decompositional search on each ordering. We then determined whether the algorithm produced the same decompositions for every ordering.

The results showed that 94 of the 100 cases each gave the same set of coherent decompositions for all 10 case orderings. The remaining 6 cases, however, gave different results for different orderings. On these cases, coherent decompositions were missed on some case orderings. The non-robust cases are summarized in figure 5-5. This figure shows that up to 3 coherent decompositions sometimes failed to be generated.

Nevertheless, non-robust cases were in the minority. For the other 94 cases, decompositional search proved to be robust, showing that incompleteness occurs only occasionally in practice. Of course, one way to overcome incompleteness would be to try different symptom orderings and use the maximal set of coherent decompositions. The increased efficiency of decompositional search makes this computationally feasible. But the decompositions that decompositional search fails to generate have multiple interacting clusters and may not be worth generating anyway.

Total decompositions in solution	Missing decompositions			
	0	1	2	3
37	6	4		
40	6	4		
49	7	3		
51	4	3	0	3
65	6	4		
118	4	5	1	

Figure 5-5 Non-robust cases for decompositional search. Each line represents a case that was not robust with respect to symptom ordering. Each case lists the total number of coherent decompositions that should have been generated. It also lists the number of case orderings for which the algorithm missed a given number of decompositions.

5.4 Case Presentation and Ordering

The first experiment showed that space and time complexity can vary widely between different cases. We therefore decided to remove the variable of target disorder selection and focus on a particular target disorder. In this experiment, we compare decompositional search and candidate generation on a single target disorder, but with different presentations of that disorder and different orderings of each presentation.

By case presentation, we mean that a given disorder may reveal only a subset of its possible effects. A particular presentation occurs in part because causality in medicine is largely probabilistic: a disease may cause a certain symptom only some of the time, and the observed effects of a disease vary because of measurement error. Also, the information available to a clinician is limited. Not all test results are available at the outset and the presence of certain symptoms may be unknown to the patient or physician. The exact set of symptoms that occur for a given target disorder is called a case presentation. A case presentation is essentially the same as a case, but we use the term to emphasize that the cases are generated by the same target disorder.

By case ordering, we mean that evidence in a case may become available in a certain sequence. The first evidence in a case may be specific, suggesting only a few possible disorders, or it may be general, suggesting numerous

possibilities. Intuitively, we would expect that specific evidence makes cases easier to solve, while general evidence makes them harder. Diagnosis is difficult when the first signs of disease are general, as is usually the case. A patient may have only vague complaints of fatigue, suggesting numerous possible causes. Only later in a diagnostic workup does specific test information usually become available.

These two variables, case presentation and case ordering, are the subject of our third experiment. We selected a single target disorder, prerenal azotemia, as our experimental model. We chose prerenal azotemia because it had relatively few possible effects, thereby generating relatively small cases. The QMR knowledge base lists only 19 possible symptoms for prerenal azotemia, of which 5 are contextual symptoms dealing with age and sex. Removing these 5 contextual symptoms, as discussed above, left us with 14 symptoms for prerenal azotemia. This compares with an average of 80 possible causes per disorder in the entire QMR knowledge base.

We generated 10 cases by stochastically picking symptoms from this pool of 14 symptoms. In contrast with the first experiment, we did not set a limit on the number of symptoms that could be selected. Rather, each symptom for prerenal azotemia was considered sequentially, and if the link probability for that symptom exceeded a randomly generated number from 0 to 1, the symptom was selected. By not limiting the number of symptoms, we simulated the causation of symptoms more naturally. This selection process was performed a total of 10 times, giving 10 different sets of symptoms. The results of this generation process are shown in figure 5-6. This figure lists the 10 cases in order of increasing complexity, so that solution of case A yields the fewest minimal candidates, while solution of case J yields the most.

The figure shows that the stochastic selection process sometimes resulted in inconsistent combinations of symptoms. For instance, two cases had a serum urea nitrogen level of both 30–59 and 60–100, while four cases had a serum creatinine level of both 0–2.9 and 3–10 mg/dl. These are somewhat inconsistent, since we would expect only one range of values to hold for each test result. Unfortunately, this highlights one of the deficiencies of the diagnostic knowledge base: it does not record dependencies or relationships among symptoms. Nevertheless, we kept the cases with these dual values. Such cases might be interpreted as having values resulting from multiple tests. Thus, a plausible hypothesis would presumably need to explain the varying values.

	Symptom	Causes	Cases A	B	C	D	E	F	G	H	I	J
s_1	Azotemia of 2 wks or less duration	2	X	X	X	X	X	X	X	X		
s_2	Creatinine clearance decreased	40	X	X	X	X	X	X	X	X	X	X
s_3	Creatinine serum 3 to 10 mg/dl	62	X			X			X		X	
s_4	Creatinine serum 2.0 to 2.9 mg/dl	38	X	X	X	X	X		X	X	X	X
s_5	Dehydration	76			X	X			X	X		X
s_6	Mouth mucosa dry (Xerostomia)	28			X							X
s_7	Oliguria	18					X		X	X	X	
s_8	pH urine less than 6	7						X	X	X	X	
s_9	Sodium urine less than 20 mEq/day	7	X	X	X	X	X	X	X	X		X
s_{10}	Urea nitrogen serum 30 to 59	72	X	X	X				X	X		X
s_{11}	Urea nitrogen serum 60 to 100	38			X						X	X
s_{12}	Urine osmolality gtr than 320	7	X	X	X	X	X	X	X	X	X	X
s_{13}	Urine output less than 400 ml/day	29	X	X		X	X	X	X	X	X	X
s_{14}	Urine sp. gravity gtr than 1.020	10		X	X	X	X	X	X	X		X
	Total Symptoms:		8	8	10	9	8	7	12	11	8	10

Figure 5-6 Cases for prerenal azotemia subdomain. The columns list the symptoms contained in each stochastically generated case. One column also lists the total number of possible causes for each symptom.

Solution Class	Case	Algorithm	Distribution by Size 1	2	3	4	5	Total
1	A,B,C,D	Cand. Gen.	1	1	25			27
		Decomp. Search	1	1	1			3
2	E	Cand. Gen.	1	0	39	33		73
		Decomp. Search	1	0	3	1		5
3	F	Cand. Gen.	1	0	11	100		112
		Decomp. Search	1	0	2	1		4
4	G,H	Cand. Gen.	1	0	6	165	165	337
		Decomp. Search	1	0	1	3	1	6
5	I	Cand. Gen.	1	11	85	435	15	547
		Decomp. Search	1	2	2	1	1	7
6	J	Cand. Gen.	1	21	769	8985	29325	39101
		Decomp. Search	1	1	3	4	1	10

Figure 5-7 Distribution of solution sizes for prerenal azotemia cases. Different cases, such as G and H may have the same solutions, allowing them to be grouped into solution classes.

5.4.1 Case Presentation

Each case represents a different presentation of prerenal azotemia. We solved the 10 case presentations using the decompositional search and candidate generation algorithms. In the process, we discovered that some cases gave the same answers, even though they contained different sets of symptoms. This occurred because the symptoms that varied from one case from another may not have been specific or different enough to give a different set of answers. Thus, more than one case belonged to a given solution class. Altogether, the 10 cases yielded 6 solution classes. These solution classes and the distributions of the solutions for both algorithms are shown in figure 5-7. This figure shows that the particular case presentation, even for the same target disorder, greatly affects the set of answers. Cases A, B, C, and D resulted in a relatively few minimal candidates, while case J resulted in a solution that was several orders of magnitude larger. As figure 5-6 shows, cases C and J differ by only a single symptom: case C contains s_1, while case J contains s_{13} instead. The results show that the complexity of diagnosis depends critically on the particular evidence available, even for the same target disorder. One symptom can make a surprisingly large difference in the overall complexity.

5.4.2 Case Ordering

For each of the 10 case presentations, we generated 10 case orderings by randomly permuting the symptoms. For each case, we also created a "specific-first" ordering, where the symptoms were ordered in ascending order according to the number of their possible causes. In other words, the first symptom was the most specific, having the fewest possible causes, while the last symptom was the most general, having the most possible causes. We also created a "general-first" case, where the symptoms were ordered in descending order, from most general to most specific. Altogether, then, each problem class generated 12 cases, for a total of 120 cases.

We solved all case orderings using the decompositional search and candidate generation algorithms. As before, time and space complexity can be compared between the two algorithms. The results are shown in figure 5-8. Again, we see a linear correlation between the complexity of the two algorithms. Linear fits of the data give rise to the following results:

Prerenal azotemia cases, random ordering

	Slope	Inverse slope
Nodes kept	0.26	3.9
Nodes expanded	0.38	2.6
Running time	0.39	2.6

These formulas again show that the space required for symptom clustering algorithm is roughly the cube-root or fourth-root of that for candidate generation, depending on the measure used, while the time required for decompositional search is approximately the cube-root of that for candidate generation.

We can analyze the effects of case presentation and ordering separately, as shown in figure 5-9. The top graph in this figure shows the distribution of total run time for each case ordering, categorized by case presentation, using candidate generation, while the bottom graph shows the same data for decompositional search. The top point on each distribution shows the time required for general-first ordering, while the bottom point shows the time required for specific-first ordering. The remaining random orderings are as shown, with the median, 25th, and 75th percentiles marked. Although we do not show them, a similar graph would have resulted from the distributions of the nodes expanded or nodes kept for each algorithm.

This figure shows that symptom ordering has a large influence on the time required to solve a given problem. The difference between the best and

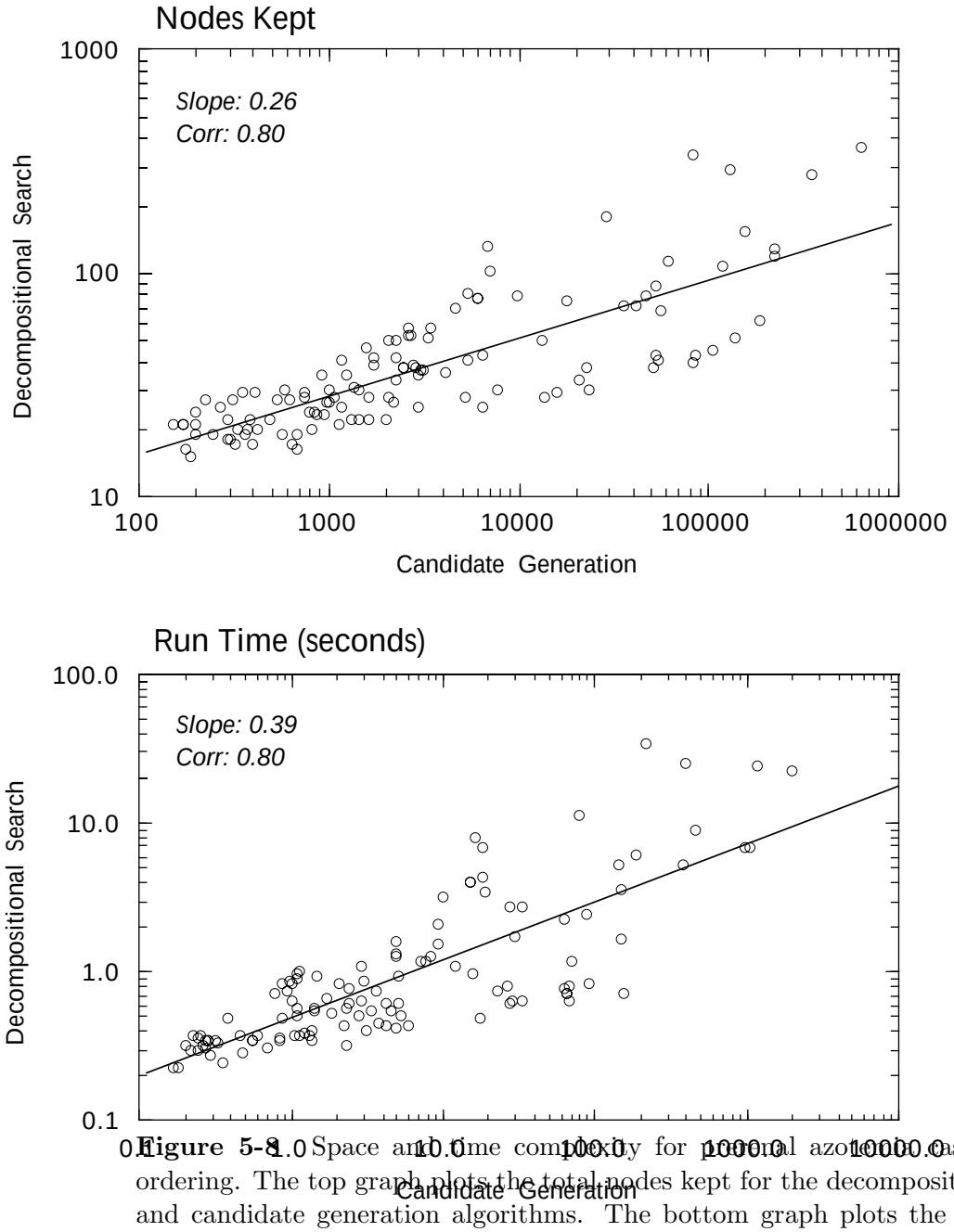

Figure 5-8 Space and time complexity for propositional azotomic cases, random ordering. The top graph plots the total nodes kept for the decompositional search and candidate generation algorithms. The bottom graph plots the run time in seconds for the two algorithms.

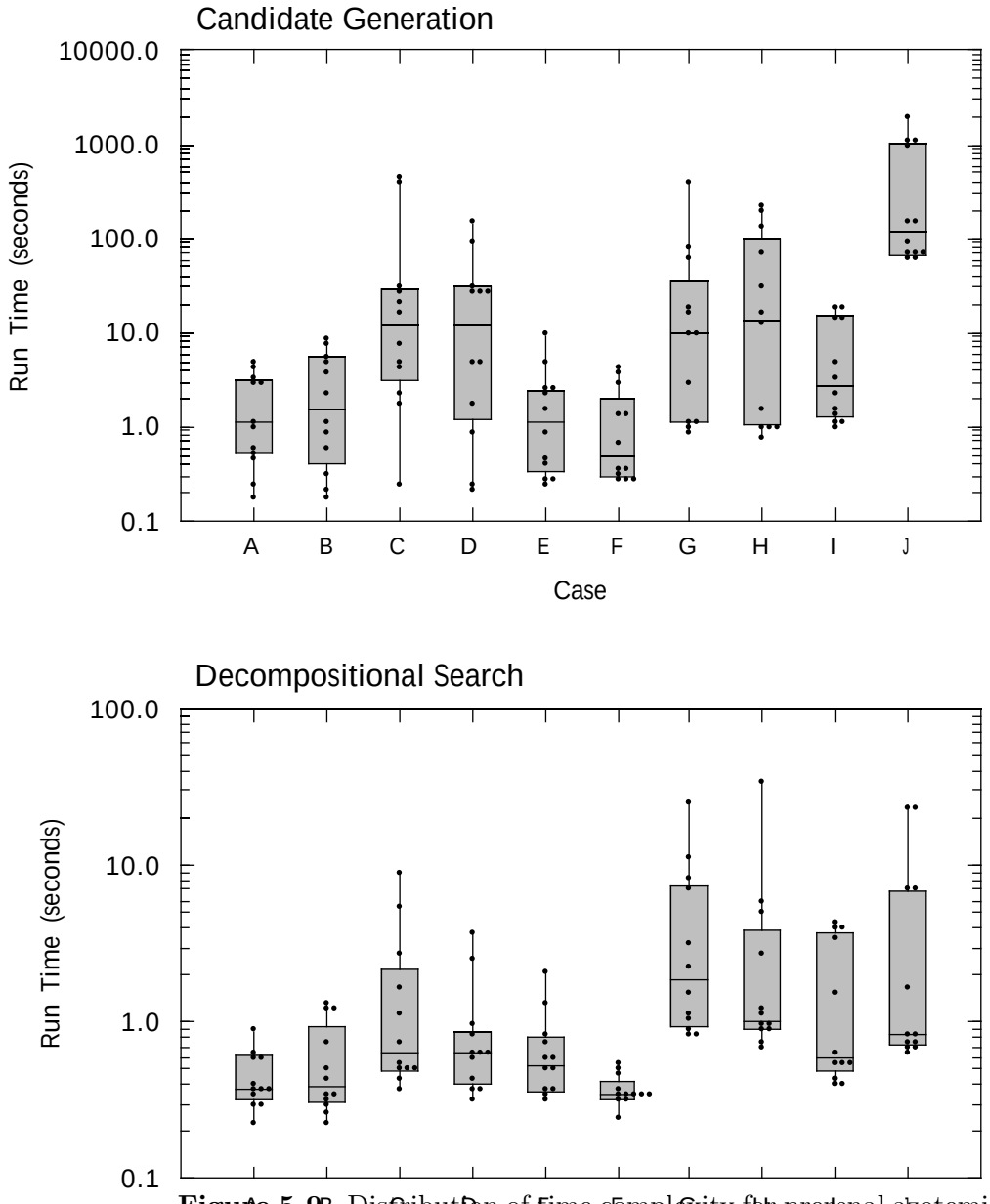

Figure 5-9 Distribution of time complexity for prerenal azotemia cases. The top graph shows the run time for candidate generation. Each dot represents a separate case ordering. The bottom graph shows the same analysis for decompositional search. The boxes mark the 25th, median, and 75th percentiles.

worst orderings can be two to three orders of magnitude. Therefore, given a set of symptoms, a diagnostic problem solver should order them so that the most specific symptoms are considered first. Of course, this strategy is only heuristic. Another ordering may in fact be faster than the specific-first ordering, perhaps because of certain structural interrelationships among the given symptoms. However, the heuristic strategy of ordering symptoms by specificity is easy to perform and probably near-optimal, as it was in the 10 cases shown here. This strategy makes intuitive sense, since the most specific symptoms give us the fewest possible choices at the outset.

5.5 Multiple-Target Cases

So far, our experimental cases have been generated by only one target disorder. As we have pointed out before, these cases still test multidisorder diagnosis, because most of the minimal candidates contain multiple disorders. Nevertheless, we can increase the complexity of diagnosis by considering cases generated by multiple target disorders. Such cases would provide a further test of the ability of the two algorithms to scale up to complex, real-world cases.

In this experiment, we generated a case by picking two target disorders at random from the QMR knowledge base. For the first disorder, we picked 7 symptoms stochastically as in the first experiment. The second disorder also generated 7 symptoms, with the proviso that each of these symptoms was distinct from those already generated for the first disorder. Thus, the two disorders gave rise to 14 different symptoms. Given our results from the previous experiment on symptom ordering, we ordered these symptoms in a specific-first order, to allow the algorithms to solve the cases as quickly as possible. A total of 100 double-target cases were generated in this fashion.

For comparison, we required a set of single-target cases. The cases generated in the first experiment were used for this purpose. As discussed above, 14 of those cases were truncated because they would otherwise have exceeded the available memory allocation. The remaining 86 cases contained 7 symptoms each. The only difference is that in the first experiment, cases were ordered randomly; in this experiment, they were arranged in specific-first order.

The results for the single-target cases are shown in figure 5-10. The log-log linear relationships gave the following results:

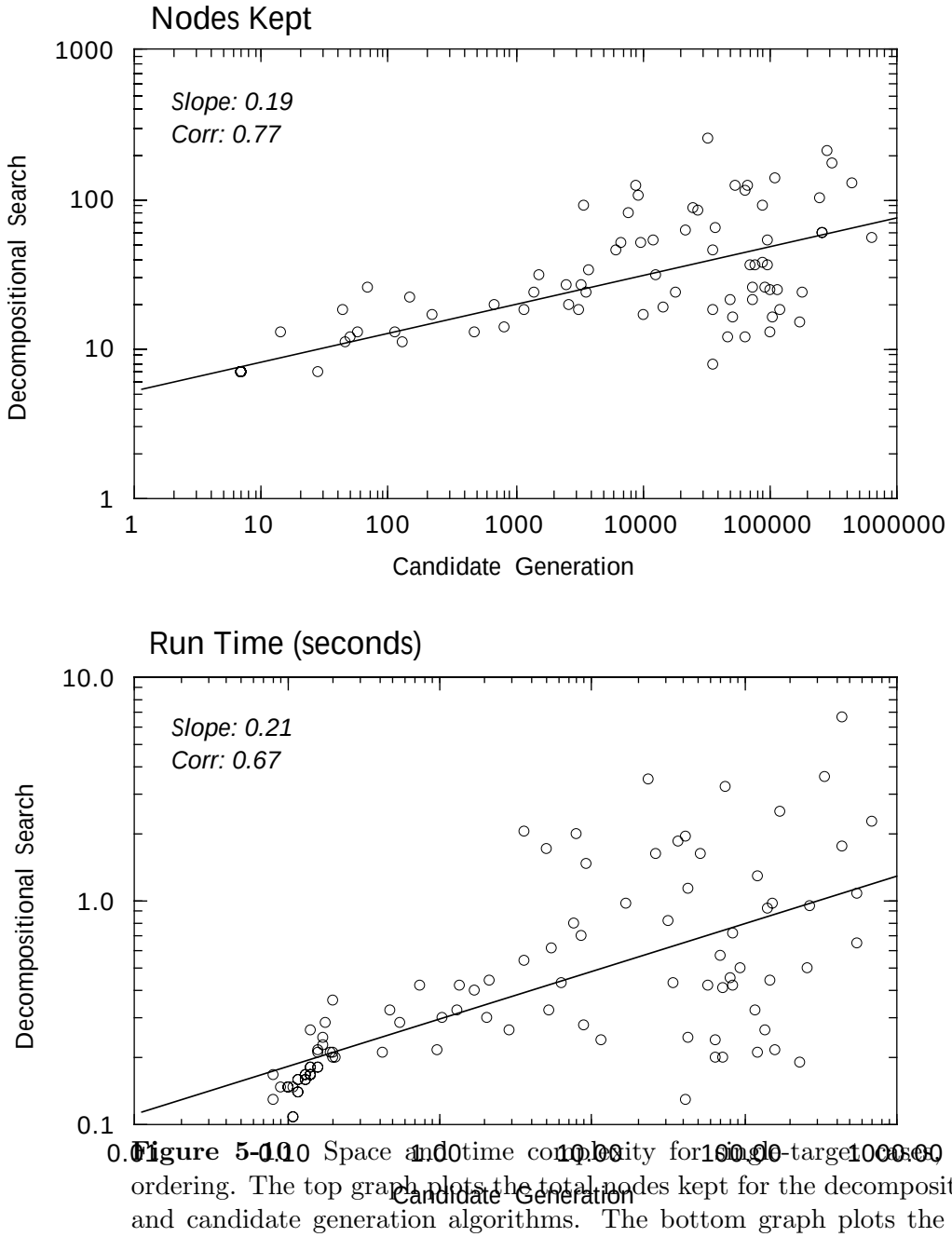

Figure 5-10 Space and time complexity for single-target case, specific-first ordering. The top graph plots the total nodes kept for the decompositional search and candidate generation algorithms. The bottom graph plots the run time in seconds for the two algorithms.

Single-target cases, specific-first ordering

	Slope	Inverse slope
Nodes kept	0.19	5.2
Nodes expanded	0.25	4.0
Running time	0.21	4.7

In other words, the space required for decompositional search is approximately the fourth-root or fifth-root of that required for candidate generation. The time required for decompositional search is approximately the fifth-root of that required for candidate generation. Note that the efficiency savings for decompositional search are even greater than in the first experiment, where the efficiency gain for nodes kept was a fourth-root savings, and for running time, a cube-root savings. This result suggests that decompositional search benefits proportionately more from a specific-first ordering strategy than candidate generation does.

We executed the decompositional search and candidate generation algorithms on the double-target cases. In the process, we found that 39 of the 100 cases could not be completed by the candidate generation algorithm because they exceeded the available memory space. As before, these cases were truncated at the point where they terminated and executed again. The resulting cases had the following characteristics:

	Symptoms									
Truncated after:	5	6	7	8	9	10	11	12	13	14
Number of cases:	1	6	4	8	5	4	5	4	2	61

The results for the double-target cases are shown in figure 5-11. The log-log linear relationships gave the following results:

Double-target cases, specific-first ordering

	Slope	Inverse slope
Nodes kept	0.18	5.4
Nodes expanded	0.24	4.1
Running time	0.20	5.0

In other words, the space required for decompositional search was approximately the fourth-root or fifth-root of that required for candidate generation. The time required for decompositional search was approximately the fifth-root of that required for candidate generation.

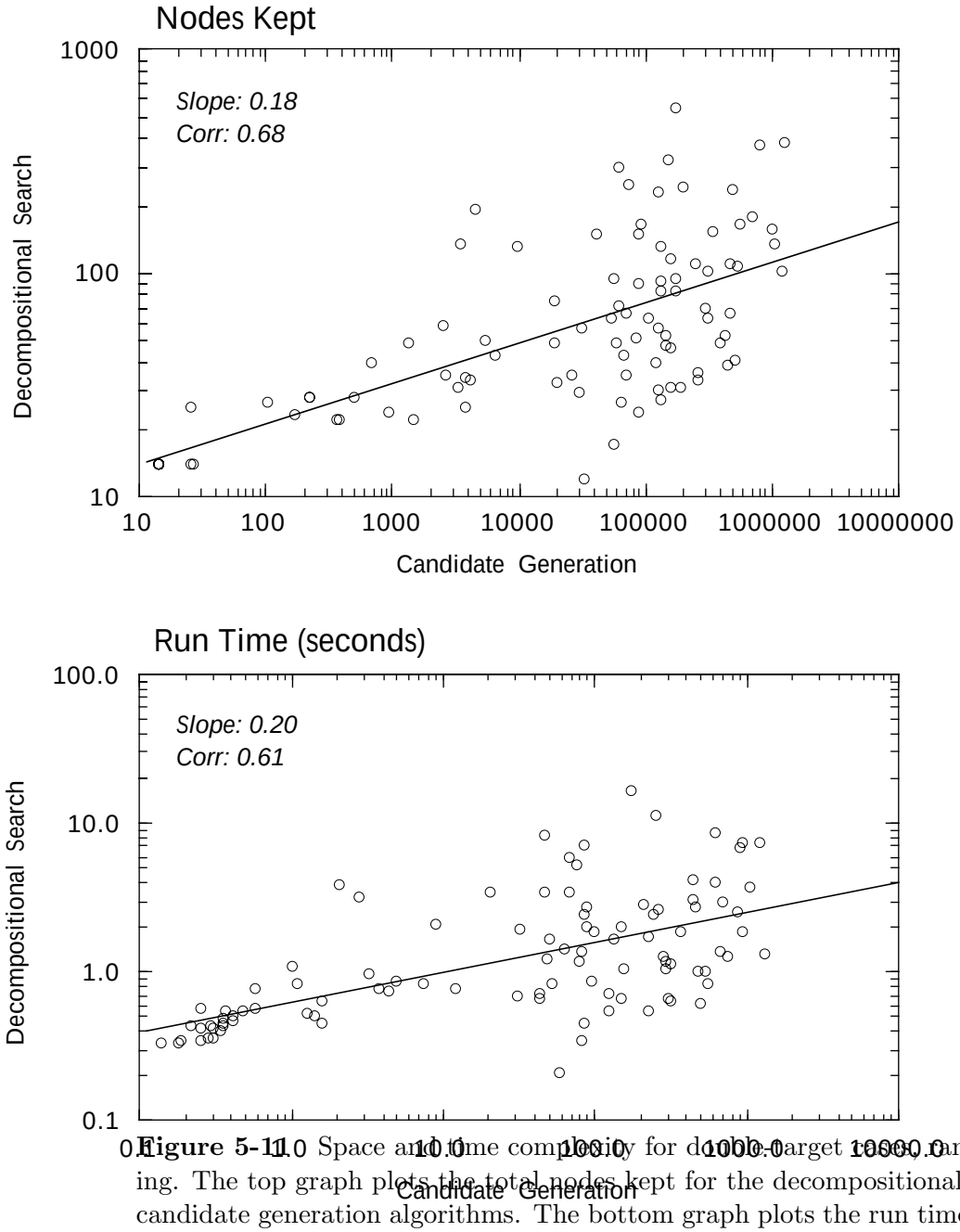

Figure 5-11 Space and time complexity for double-target cases, random ordering. The top graph plots the total nodes kept for the decompositional search and candidate generation algorithms. The bottom graph plots the run time in seconds for the two algorithms.

The savings for the double-target cases are approximately the same as for the single-target cases. This result indicates that the efficiency gains for decompositional search are relatively independent of the number of actual disorders present. In other words, the computational complexity increases multiplicatively with each additional target disorder, regardless of the algorithm used. This result follws because the solutions for each target disorder must be combined, even in decompositional search. Nevertheless, since decompositional search explains each target disorder more compactly, it has fewer decompositions to combine for each target disorder.

Chapter 6

Analysis

The fact . . . that many complex systems have a nearly decomposable, hierarchic structure is a major facilitating factor enabling us to understand, describe, and even "see" such systems and their parts.

— Herbert A. Simon, *The Sciences of the Artificial* (1969)

The experimental results in the previous chapter indicate that decompositional search is more efficient than candidate generation. In this chapter, we determine the reasons why. We explore this topic in several ways. First, we identify the major source of computational complexity, namely, the combinatorics of partial explanations. Then, we develop a theoretical model for the diagnostic process and compare the two algorithms using this model. Finally, we study the role of structure in problem solving through a series of experiments. Our analysis reveals that domain structure plays a large role in the efficiency of multidisorder diagnosis and that decompositional search exploits domain structure significantly.

6.1 Combinatorics of Partial Explanations

If we examine the solutions to the cases in the previous chapter, we discover that most of the complexity of candidate generation occurs in the largest solutions. For instance, consider the most difficult set of cases in the prerenal azotemia experiment, those arising from problem class J. The solutions to those cases have the following size distribution:

Solution Class	Case	Algorithm	Distribution by Size 1	2	3	4	5	Total
6	J	Cand. Gen.	1	21	769	8985	29325	39101
		Decomp. Search	1	1	3	4	1	10

As the table shows, candidate generation behaves most poorly on the largest solutions. Conversely, decompositional search represents the largest solutions most compactly. For solutions of size 5, one problem decomposition represents over 29,000 candidates. Thus, the efficiency of decompositional search derives mainly from the compact representation of large candidates. Ironically, the largest candidates are also usually the least likely because each disorder requires consideration of an additional prior probability. Therefore,

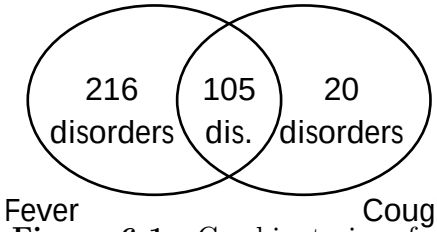

Figure 6-1 Combinatorics of partial explanations. This Venn diagram shows the number of possible causes for fever alone, cough alone, and both diseases, taken from the QMR knowledge base. The partial explanations are contained in the non-overlapping regions of diagram.

the allocation of effort by the candidate generation algorithm is counterproductive, spending most of its time on the least likely candidates.

The reason for the explosion of large minimal candidates is a matter of combinatorics. Consider the simple example of fever and cough from chapter 1. In that example, fever and cough each had three possible causes, with one cause explaining both symptoms. But consider a more realistic version of that example. In the QMR knowledge base, fever has 321 possible causes and cough has 125 possible causes, with 105 causes explaining both symptoms. The Venn diagram for this situation is shown in figure 6-1.

The 105 causes for both fever and cough are *complete explanations*, because each can explain all of the given symptoms. But the 216 and 20 causes in the non-overlapping regions of the Venn diagram are *partial explanations*, explaining only some of the given symptoms. If only single-disorder candidates are allowed, then only complete explanations need be considered. But if we allow multiple disorders, we must construct candidates from combinations of partial explanations.

Partial explanations fall into categories of *causal equivalence*, based on their ability to explain the symptoms in a given case. As the knowledge base scales up in size, the phenomenon of causal equivalence becomes increasingly important. In our example, the 216 causes for fever alone are causally equivalent, as are the 20 causes for cough alone and the 105 causes for both fever and cough. The phenomenon of causal equivalence explains why minimal candidates factor into Cartesian product representations. To form a two-disorder minimal candidate, we need to select one disorder from the 216 causes for fever alone and one disorder from the 20 causes for cough alone. Thus, combinations of partial explanations grow multiplicatively. In our example, the number of such combinations equals the product of the

partial explanations for each symptom: $(216)(20) = 4425$.

Thus, the compact representation of combinations of partial explanations accounts for much of the power of decompositional search. Rather than representing such combinations explicitly, decompositional search groups together disorders that are causally equivalent. Problem decompositions represent structures by which partial explanations combine. In our example, the 4425 two-disorder minimal candidates for fever and cough are represented by a single decomposition. The space required to represent these candidates is additive rather than multiplicative. The space required to represent a problem decomposition is $216 + 20 = 236$ disorders. This compact representation enables decompositional search to spend less space and time on generating and evaluating large candidates.

6.2 Theoretical Analysis

In this section, we analyze the computational complexity of diagnosis by developing a mathematical model. This model supplements the experimental results by yielding more insight into the properties of a domain that should influence computational complexity. However, our analysis is limited because, like most complexity analyses, it deals only with worst-case complexity. Moreover, since complex models are difficult to analyze, our model necessarily incorporates some strong simplifications.

The important parameters in our model are the number of possible causes for each symptom and their degree of correlation. Let c be the number of possible causes per symptom, and ρ be the correlation between the sets of possible causes, or simply the symptom correlation. We define the symptom correlation for two symptoms s_1 and s_2 to be

$$\rho(s_1, s_2) = \frac{|\text{Causes}(s_1) \cap \text{Causes}(s_2)|}{\min(|\text{Causes}(s_1)|, |\text{Causes}(s_2)|)} \tag{6.1}$$

Symptom correlation measures the degree of overlap between symptoms s_1 and s_2. It has a value between 0 and 1, with 0 signifying no overlap between possible causes, and 1 signifying complete overlap between possible causes. Our model assumes that c and ρ are constant for any symptom or pair of symptoms. By making these parameters constant, we are assuming that possible causes for symptoms are distributed randomly. In other words, our

simplified domain lacks any notion of structure between symptoms, except for a constant correlation between each pair.

Our model has some interesting properties. For instance, it allows an arbitrarily large number of symptoms in a case, but only a finite number of disorders to be triggered. Our model has this property because the total number of disorders grows asymptotically. The n^{th} symptom adds a non-overlapping area of size $(1-\rho)^{n-1}c$, which represents new disorders not in any other symptom so far. Thus the total number of disorders triggered by n symptoms is

$$c + (1-\rho)c + (1-\rho)^2 c + \ldots \;\approx\; c/\rho$$

using the formula for geometric sums. Thus, each new symptom triggers a diminishing number of new possible causes. This is because the quantity $(1-\rho)^{n-1}c$ decreases as more symptoms are added. This model simulates the generation of symptoms based on a single target disorder.

We assume, as the example of prerenal azotemia illustrates, that the complexity of candidate generation stems primarily from the largest candidates. These candidates derive from combining disorders that each explain only one of the given symptoms. Our analysis therefore will concentrate on the growth in the number of these large candidates. These candidates come from the non-overlapping regions of the symptom space, the parts of the possible cause sets that explain only a single symptom.

The n^{th} symptom removes a fraction of ρ from each non-overlapping region, meaning that $(1-\rho)$ of each region is retained after each new symptom. Thus, for n symptoms, there are n non-overlapping regions, each containing $(1-\rho)^{n-1}c$ disorders. Thus, the total number of candidates obtained by combining these partial explanations equals

$$|\text{Candidates}| \;=\; ((1-\rho)^{n-1}c)^n \;=\; (1-\rho)^{n(n-1)}c^n$$

The number of large minimal candidates will therefore begin with a rapid growth, controlled by the c^n factor. Then, as more symptoms are added, the number of large minimal candidates should decrease, as the $(1-\rho)^{n(n-1)}$ factor takes over. Since the total search complexity depends on the widest part of the search tree, we want to know when |Candidates| reaches a maximum. This is found by setting its derivative equal to zero:

$$\begin{aligned} 0 &= \frac{d}{dn}(1-\rho)^{n(n-1)}c^n \\ &= (1-\rho)^{n(n-1)}c^n\left[(2n-1)\ln(1-\rho) + \ln c\right] \end{aligned}$$

This implies that

$$\begin{aligned} 2n-1 &= \frac{-\ln c}{-\ln(1/(1-\rho))} \\ n &= 1+\frac{\log_r c}{2}, \quad r = 1/(1-\rho) \end{aligned}$$

Thus, the number of large minimal candidates is bounded by $O(c^{\log_r c})$. This function is plotted in the top graph of figure 6-2 for various values of c and ρ. The complexity is essentially exponential in the number of possible causes per symptom (c), with the point of takeoff depending on symptom correlation (ρ). As symptoms become less correlated, the complexity increases. This makes sense, because smaller symptom correlations mean proportionately larger non-overlapping regions, thereby increasing the number of partial explanations that can be combined.

For decompositional search, the analysis is a bit more difficult. Recall that a problem decomposition is a collection of subsets of P such that each cluster contains at least one unique symptom. However, not all decompositions can be generated by decompositional search simultaneously. This is because some decompositions are more ambiguous than others, and decompositional search generates only maximally ambiguous decompositions. Therefore, the maximum number of maximally ambiguous decompositions is bounded above by the number of partitions of P. If P contains n symptoms, this quantity equals the n^{th} Bell number, b_n [1]. The first few Bell numbers are:

n	1	2	3	4	5	6	7	8	9	10
b_n	1	2	5	15	52	203	877	4140	21147	115975

As these terms show, the number of possible decompositions grows very rapidly. The Bell sequence is bounded below by 2^n, so it grows at least exponentially. It is bounded above by the factorial sequence, so $b_n < n!$ for $n > 2$. However, our experimental evidence does not show such a rapid rate of increase. This is partly due to the plausibility criteria that sharply limit the number of decompositions. Thus, rather than the total number of maximally ambiguous decompositions, we want to estimate the number of coherent decompositions. Unfortunately, this is a difficult question and our analysis is somewhat speculative.

We conjecture that the number of decompositions grows until the explanation set for each cluster becomes small enough that it either becomes empty

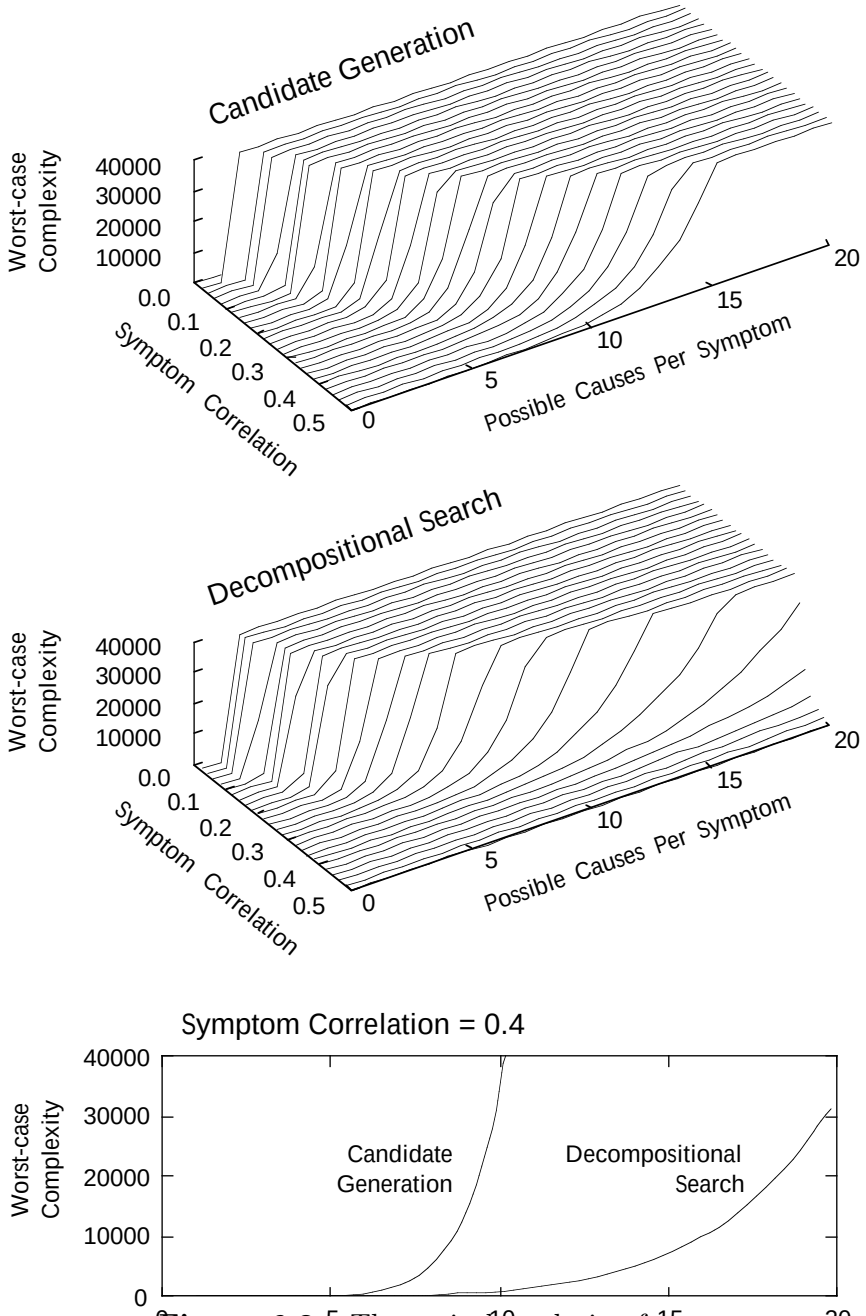

Figure 6-2 Theoretical analysis of worst-case complexity. (Top) Graphs of worst-case complexity for candidate generation, for various parameters of symptom correlation. (Middle) Graphs of worst-case complexity for decompositional search. (Bottom) Comparison of the worst-case complexity for the two algorithms for symptom correlation of 0.4.

or coverable by a new symptom. This certainly occurs when most regions in the symptom space contain only a single disorder. Then the next symptom will either contain the disorder—thereby covering it—or not contain it—resulting in a empty common causes set and an incoherent decomposition. In our model, a region that explains x symptoms out of n contains

$$\rho^{x-1}(1-\rho)^{n-x}c = \frac{(1-\rho)^n}{\rho}(\frac{\rho}{1-\rho})^x c$$

disorders. When the largest region contains only a single disorder, the growth of problem decompositions should certainly halt. Thus, we want to know the value of x that gives the largest region. If we assume that $\rho < 0.5$, the term $\rho/(1-\rho)$ is less than 1, so the largest region is achieved when x is minimized, or $x = 1$. These regions are again the non-overlapping regions that we considered in the analysis of candidate generation. The number of disorders in this region is

$$\begin{aligned}\frac{(1-\rho)^n}{\rho}(\frac{\rho}{1-\rho})c &= (1-\rho)^{n-1}c \\ &= r^{1-n}c\end{aligned}$$

where r is again defined to be $1/(1-\rho)$.

The largest region contains less than one disorder whenever

$$\begin{aligned}r^{1-n}c &< 1 \\ (1-n) &< -\log_r c \\ n &> 1+\log_r c\end{aligned}$$

So the number of decompositions should cease growing when n exceeds $\log_r c$. Thus, we expect the number of clusterings to be bounded as follows:

$$\begin{aligned}|\text{Decompositions}| &= O(n!) \\ &< O(n^n) \\ &< O((\log_r c)^{\log_r c}) \\ &= O(c^{\log_r \log_r c})\end{aligned}$$

In the last step, we used the fact that $a^{\log b} = b^{\log a}$. The worst-case complexity of decompositional search is plotted in the middle graph of figure 6-2. Again, as with candidate generation, the complexity grows exponentially

with the number of possible causes per symptom. The symptom correlation parameter also controls the point at which the exponential function takes off, with higher correlations resulting in slower growth functions. However, for the same symptom correlation, the complexity of decompositional search takes off at a higher value of c than with candidate generation and also grows at a slower rate. A direct comparison of candidate generation and decompositional search is shown in the bottom graph of figure 6-2 for a particular value of ρ, namely 0.4.

6.3 Domain and Problem Structure

The theoretical analysis presented above is limited by the fact that it is a worst-case analysis. It computes only upper bounds on the possible number of coherent decompositions or minimal candidates. However, in most cases, this theoretical maximal limit is not reached. Moreover, the theoretical model made the simplifying assumptions that symptoms have equal numbers of possible causes and that their pairwise correlations are equal. These assumptions, of course, do not hold in natural domains. Finally, the model does not incorporate any elements of domain structure.

In the remainder of this chapter, we examine empirically the role of domain structure in diagnostic complexity. It is clear that domain structure greatly affects the computational behavior of algorithms. But it is less clear how one should describe or quantify domain structure. We propose to measure structure by the distribution of a quantity called *explanatory power*. By manipulating this quantity, we can alter domain structure and observe the resulting effect on diagnostic complexity.

The domain knowledge, which is embodied in a bipartite knowledge base, can be characterized by two quantities: the distribution of possible causes for its symptoms and the distribution of possible effects for its disorders. For the QMR knowledge base, these distributions are presented in figure 6-3. These distributions reveal an asymmetry in the bipartite knowledge base: the possible effects are normally distributed in size, while the possible causes are exponentially distributed. We conjecture that these distributions are characteristic of most natural domains. The reason for this is that symptoms vary considerably in their generality and specificity. Indeed, symptoms are often designed to vary in terms of generality or specificity. General symptoms, such as fatigue, are useful for determining whether a problem exists; they gener-

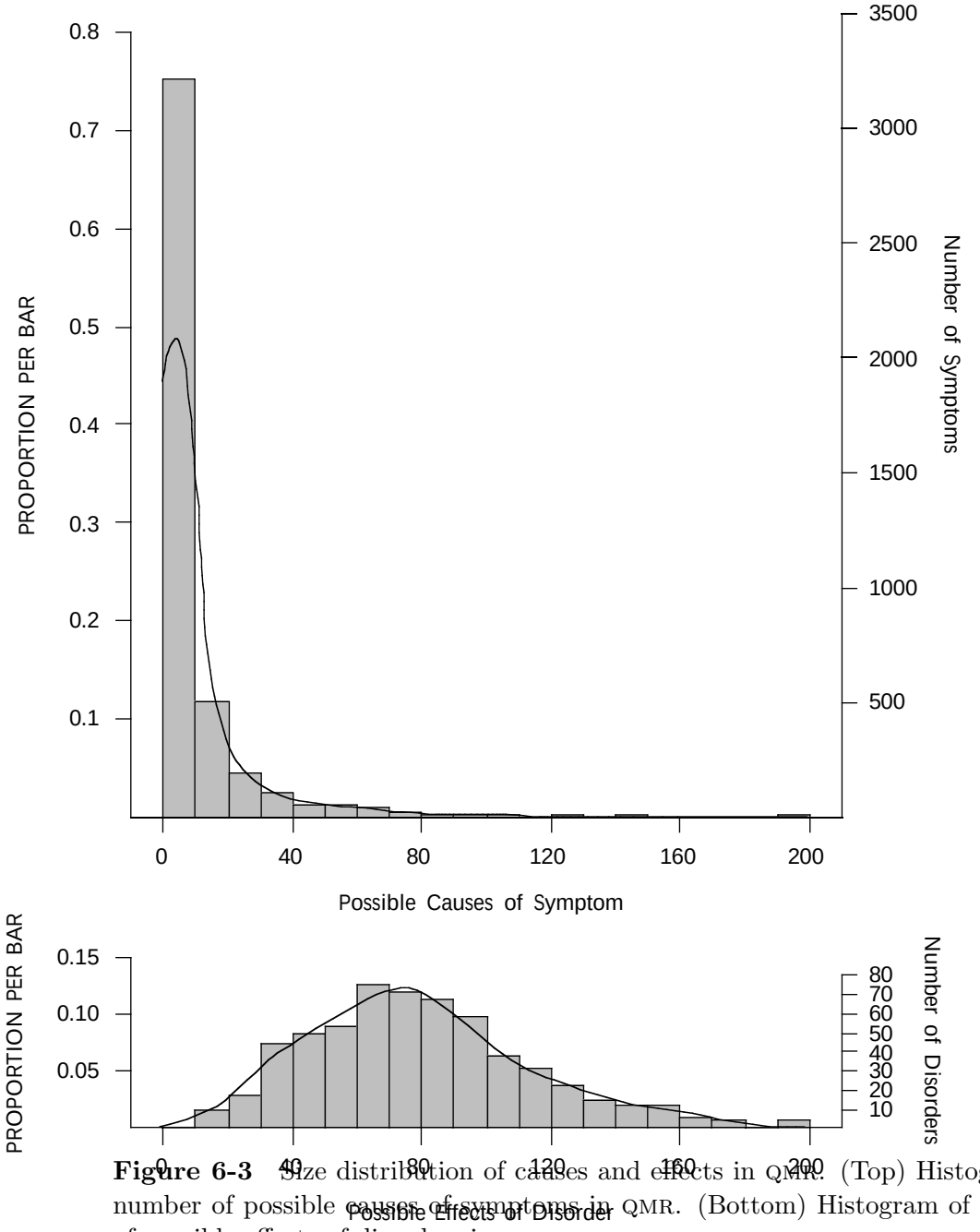

Figure 6-3 Size distribution of causes and effects in QMR. (Top) Histogram of number of possible causes of symptoms in QMR. (Bottom) Histogram of number of possible effects of disorders in QMR.

ally are sensitive indicators of a diagnostic problem. Specific symptoms, such as a urinary pH level, are useful for confirming or ruling out disorders. Specific tests have been developed for most disorders, so that most symptoms in the knowledge base are specific for a small set of disorders. This explains why possible causes for symptoms are distributed exponentially. On the other hand, each disorder has a relatively large number of both specific and general symptoms. This explains why possible effects for disorders are distributed normally.

But the size distributions for the entire domain are of secondary importance compared to the particular subdomain faced by the diagnostic system. The subdomain is defined by the symptoms in a case, which are generated by a set of target disorders, either by experimental stochastic simulation or by causal relationships in real life. A subdomain contains the set of symptoms present, along with the disorders that explain each symptom. These disorders are competitors of the initial target disorders. Of course, the number of target disorders and their identity is not known to the diagnostic problem solver.

An example of a subdomain is shown in figure 6-4. This is the subdomain for prerenal azotemia, where we assume that all possible effects of prerenal azotemia are in fact present. However, a different subdomain for prerenal azotemia might be defined using a subset of its possible effects. The subdomain also contains disorders linked to the symptoms. In figure 6-4, these disorders are arranged according to the number of symptoms in the subdomain that they explain. The disorders that explain the most symptoms are placed closest to the middle. The disorder that explains all of the symptoms, prerenal azotemia, is located in the center.

Thus, disorders can be ranked in terms of the fraction of positive symptoms in the case that they explain. We call this the *explanatory power* of a disorder. We define the explanatory power of disorder d for positive symptoms P as

$$\text{Power}(d) \quad = \quad \frac{|P \cap \text{Effects}(d)|}{|P|} \tag{6.2}$$

where $\text{Effects}(d)$ is the set of possible effects for d. The denominator serves to normalize the value of $\text{Power}(d)$ so that it lies between 0 and 1. This concept generalizes our earlier distinction between complete and partial explanations. A complete explanation is one whose explanatory power equals 1. A partial explanation has explanatory power less than 1. However, rather than having

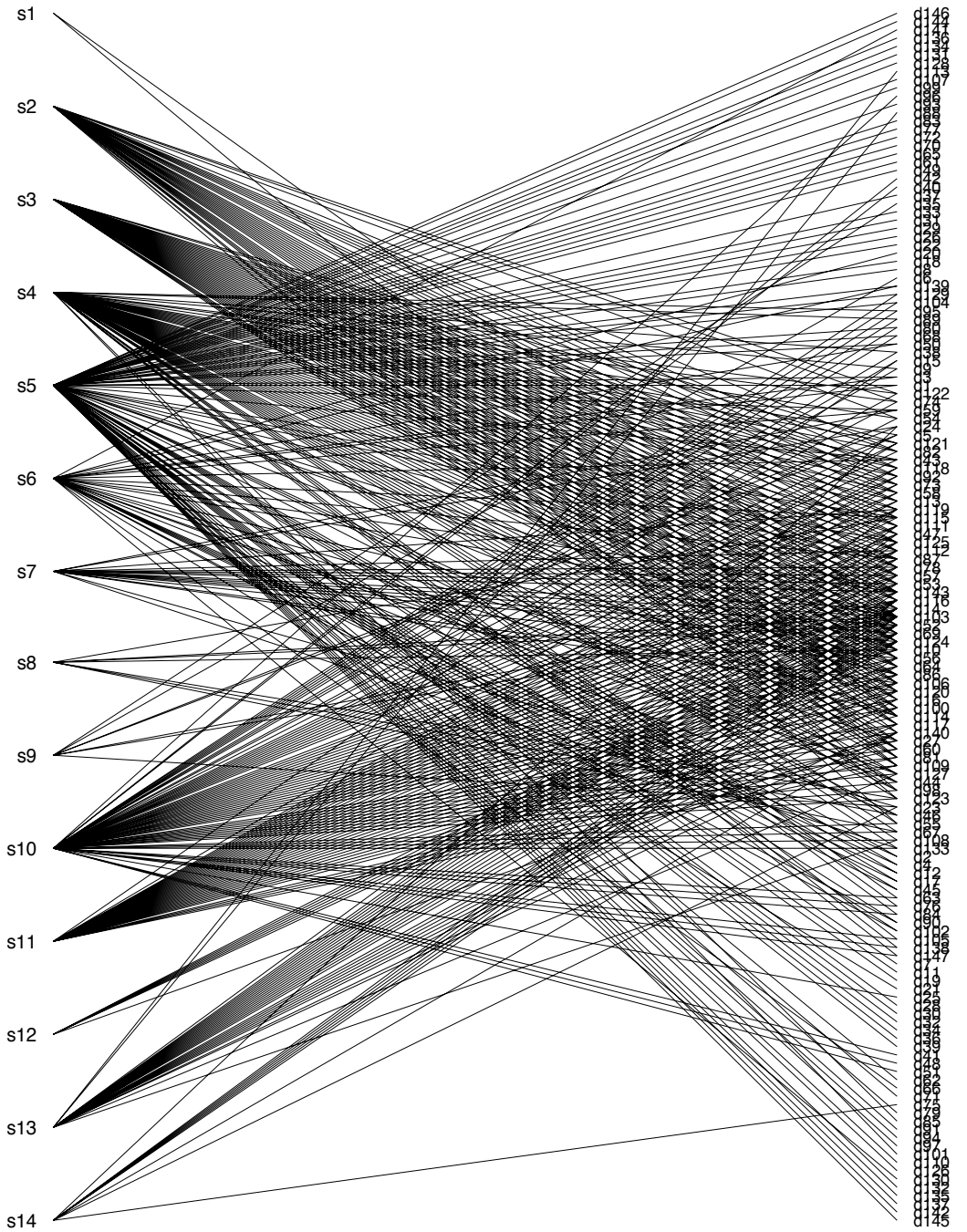

Figure 6-4 Prerenal azotemia subdomain. Symptom and disorder labels correspond to those listed in appendix D.

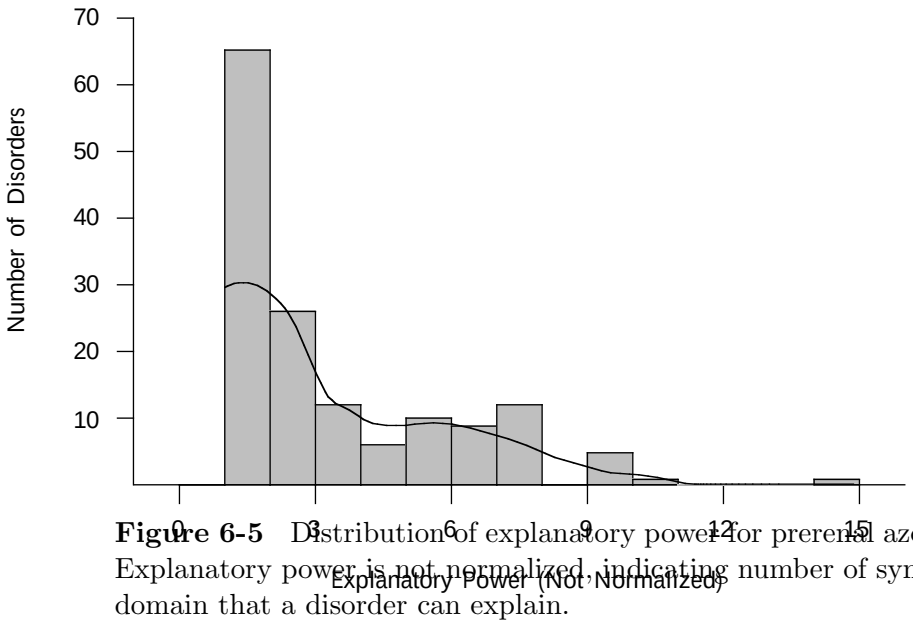

Figure 6-5 Distribution of explanatory power for prerenal azotemia subdomain. Explanatory power is not normalized, indicating number of symptoms in the subdomain that a disorder can explain.

only a dichotomy between complete and partial explanations, we now have gradations of explanatory power.

The graph of the prerenal azotemia subdomain shown in figure 6-4 shows how gradations of explanatory power are distributed, with a few disorders in the middle having much explanatory power, but most disorders in the periphery having little explanatory power. This distribution is shown more explicitly in figure 6-5. The explanatory power for this subdomain shows a distribution skewed towards the left, but with several disorders having an intermediate level of explanatory power. The single disorder having maximal explanatory power is the target disorder, prerenal azotemia, the only complete explanation. We describe this distribution as bimodal because it separates those disorders having essentially no explanatory power from those with intermediate explanatory power.

This bimodal distribution differs greatly from a normal distribution. Bimodality derives from the presence of decompositional structure in a domain. If a domain has decompositional structure, the target disorder will belong to a group of similar disorders. Groups of disorders induce a ranking among competing disorders. Some disorders will have similar possible effects compared to the target disorder, and thereby have high explanatory power. Other disorders will have largely different possible effects, resulting in relatively low explanatory power. The separation of similar and dissimilar

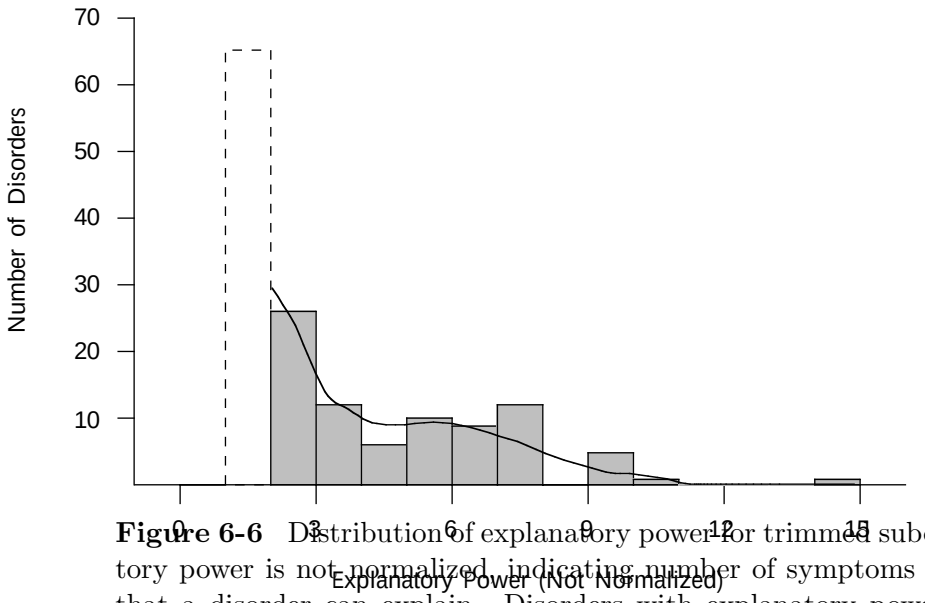

Figure 6-6 Distribution of explanatory power for trimmed subdomain. Explanatory power is not normalized, indicating number of symptoms in the subdomain that a disorder can explain. Disorders with explanatory power of 1 have been removed.

disorders depends on how complete the decompositional domain structure is. The distribution of explanatory power in figure 6-5 is evidence for near decomposability in the QMR knowledge base, at least in the neighborhood of prerenal azotemia.

With this notion of explanatory power, we can modify our original hypothesis about the combinatorics of partial explanations. We can now ascribe diagnostic complexity to the combinatorics of disorders with low explanatory power. We now conduct two experimental analyses to further assess the role explanatory power in diagnostic complexity.

6.4 Trimmed Subdomain

The first analysis tests the role played by disorders with low explanatory power. This experiment modifies the prerenal azotemia subdomain by removing disorders with low explanatory power and observing the effect on the two algorithms. Specifically, we remove those disorders in the prerenal azotemia subdomain that explain only a single symptom. The resulting subdomain is called a "trimmed" subdomain, since the least powerful disorders are trimmed. This modification removes the left-most side of the explanatory power distribution, as shown in figure 6-6. We then run the same experimen-

tal case orderings for the prerenal azotemia subdomain as before. Since we have presumably removed the disorders that account for most of the combinatorial explosion, we expect a decrease in the space and time required for the same cases.

We compare decompositional search and candidate generation as before, but on the trimmed subdomain. The results are shown in figure 6-7 and summarized below:

Prerenal azotemia cases, trimmed subdomain

	Slope	Inverse slope
Nodes kept	0.30	3.3
Nodes expanded	0.40	2.5
Running time	0.43	2.3

The results show that decompositional search is more efficient than candidate candidate, even on a simplified subdomain. However, the amount of efficiency gain is slightly smaller than for the original subdomain. Originally, as shown previously in figure 5-8, decompositional search had space and time advantages that were powers of 3.9 and 2.6, respectively. With the trimmed subdomain, these advantages lessened to powers of 3.3 and 2.3.

In addition, instead of comparing algorithms with each other, we can analyze each of them separately on the two subdomains. We can see how each algorithm responded to trimming of the subdomain. These comparisons for time complexity are shown in figure 6-8. The full results are summarized below:

Trimmed subdomain, candidate generation

	Slope	Inverse slope
Nodes kept	0.68	1.59
Nodes expanded	0.71	1.41
Running time	0.63	1.47

That is, adding the disorders with least explanatory power to the trimmed subdomain would have increased the time complexity by a power of approximately 1.5. Thus, the least powerful disorders accounted for a significant portion of the diagnostic complexity.

For decompositional search, the results were similar:

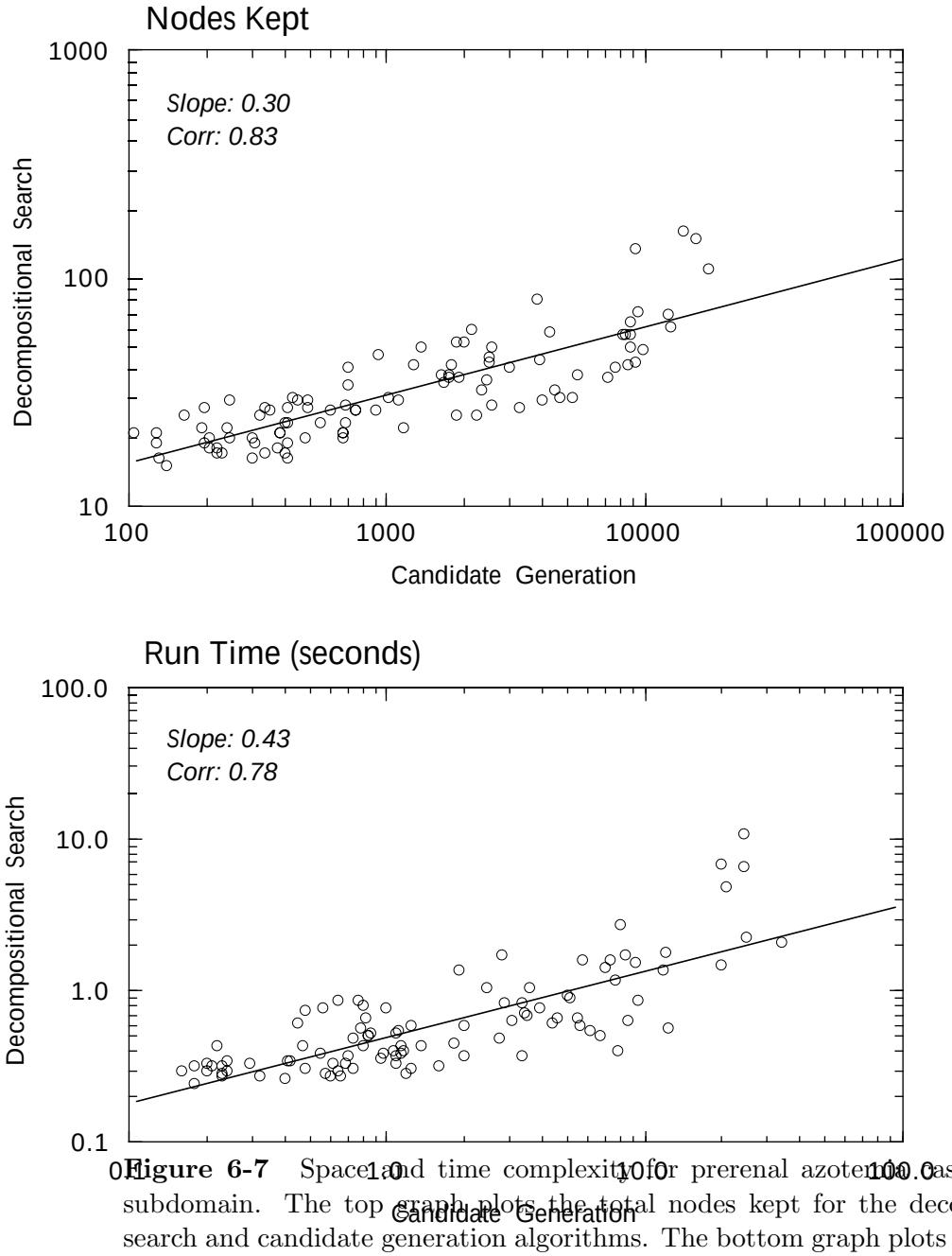

Figure 6-7 Space and time complexity for prerenal azotemia cases, trimmed subdomain. The top graph plots the total nodes kept for the decompositional search and candidate generation algorithms. The bottom graph plots the run time in seconds for the two algorithms.

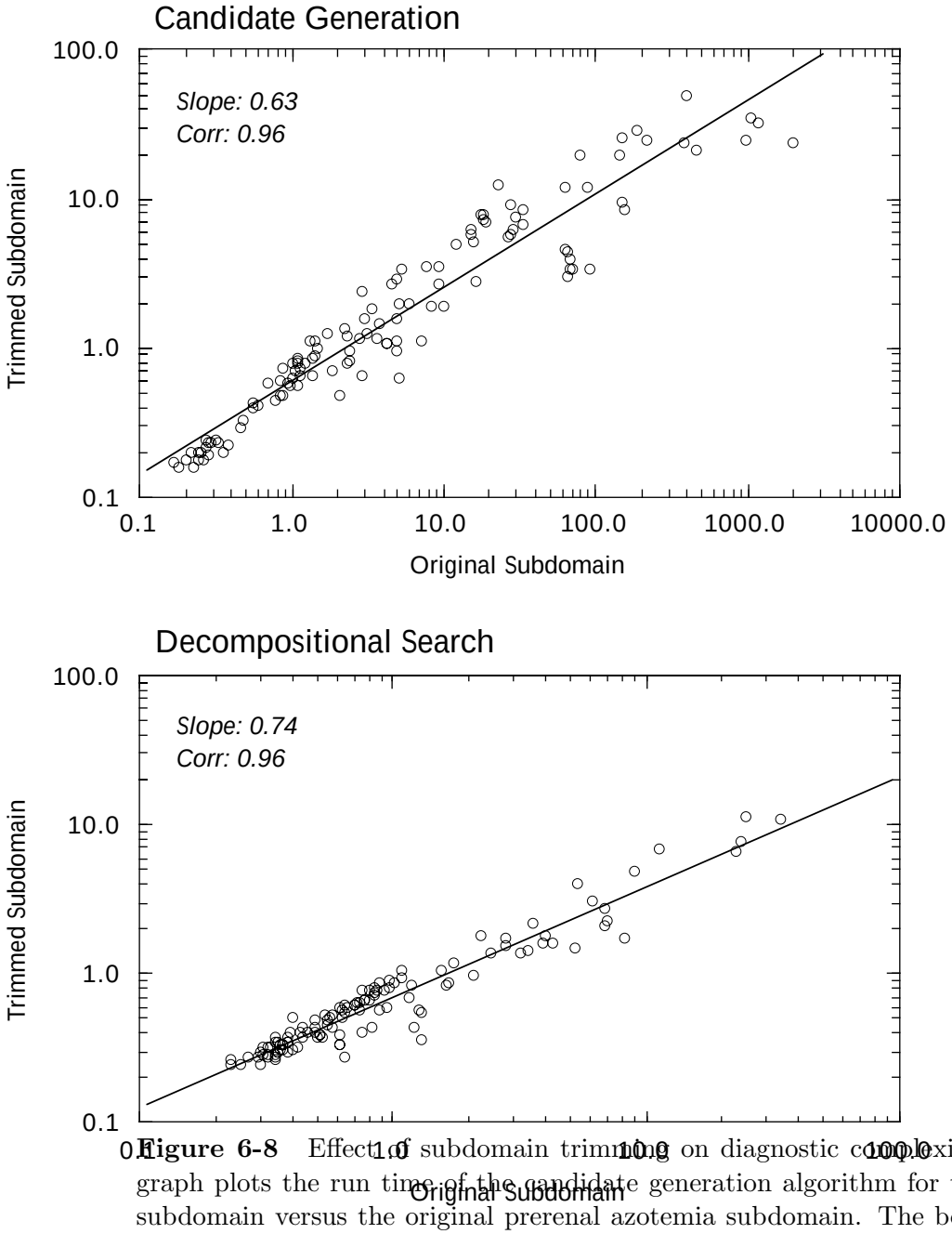

Figure 6-8 Effect of subdomain trimming on diagnostic complexity. The top graph plots the run time of the candidate generation algorithm for the trimmed subdomain versus the original prerenal azotemia subdomain. The bottom graph plots the same results for the decompositional search algorithm.

Trimmed subdomain, decompositional search

	Slope	Inverse slope
Nodes kept	0.77	1.30
Nodes expanded	0.76	1.32
Running time	0.74	1.37

The space and time complexity for the trimmed subdomain were powers of 1.30 and 1.37, respectively, compared to the original subdomain. Note that the effect of trimming on decompositional search was slightly less than that for candidate generation. That is, the least powerful disorders would have increased the complexity of the trimmed subdomain by a power of 1.5 for candidate generation but only 1.3 for decompositional search. This indicates that decompositional search handles the combinatorics of the least powerful disorders relatively better than candidate generation.

6.5 Redistributed Subdomain

Our second experimental analysis explores the role of domain structure in diagnostic complexity. We have argued that in a domain with decompositional domain structure, the explanatory power for diagnostic subdomains should have a non-normal, bimodal distribution. In such a distribution, disorders are separated to some extent into those with relatively low explanatory power and those with relatively high explanatory power. A bimodal distribution is evidence of decompositional structure. Decompositional structure occurs because the causal links in a knowledge base are not distributed randomly. Rather, there are clusters of symptoms and corresponding clusters of disorders, with relatively dense links within clusters and relatively sparse links between them. Competing disorders therefore have varying degrees of similarity to a given target disorder, depending on whether they belong to the same cluster.

We hypothesize that such decompositional structure is exploited by decompositional search and accounts for much of its efficiency. We test this hypothesis by experimentally removing structure from the prerenal azotemia subdomain and observing the result. Specifically, we remove structure by randomizing the possible causes for each symptom in the subdomain. Each symptom keeps the same number of possible causes as before, but the actual disorders that can cause each symptom are redistributed among the entire set of disorders in the subdomain. The only restriction is that each symptom

continues to contain the target disorder as a possible cause. This is done to maintain the semantics of the subdomain, so that all symptoms derive from the target disorder.

If we apply this procedure to the original prerenal azotemia subdomain, we get the redistributed subdomain shown in figure 6-9. As with the original subdomain, the disorders are arranged so that those with the greatest explanatory power are in the middle. In this figure, the disorders exhibit differences in explanatory power, resulting from the random redistribution process. Comparing the figure with the original subdomain, shown previously in figure 6-4, we may be able to discern that the distribution of explanatory power has become more uniform. To show this change more clearly, we graph the distribution of explanatory power in figure 6-10. This shows that the distribution of explanatory power has been changed from a bimodal distribution to a normal one.

We use this redistributed subdomain to run the same cases for prerenal azotemia as before, using the candidate generation and decompositional search algorithms. The two algorithms are compared in figure 6-11. The linear fits are shown below:

Prerenal azotemia cases, redistributed subdomain

	Slope	Inverse slope
Nodes kept	0.65	1.54
Nodes expanded	0.78	1.28
Running time	0.77	1.30

These compare to the original, structured subdomain, which had efficiency gains of 3.9, 2.6, and 2.6 for nodes kept, nodes expanded, and running time, respectively. Thus, domain structure apparently plays a large role in making decompositional search more efficient than candidate generation. When this structure is removed, the efficiency gain of decompositional search lessens substantially.

As with the trimmed subdomain experiment, we can analyze the effects of redistribution on each algorithm individually. The graphs are shown in figure 6-12. In these graphs, the time complexity for the redistributed subdomain is plotted on the abscissa, while the complexity for the structured subdomain is plotted on the ordinate. This allows us to analyze the effects of structure on reducing complexity. Again, only running time is plotted; the full results for candidate generation are shown below:

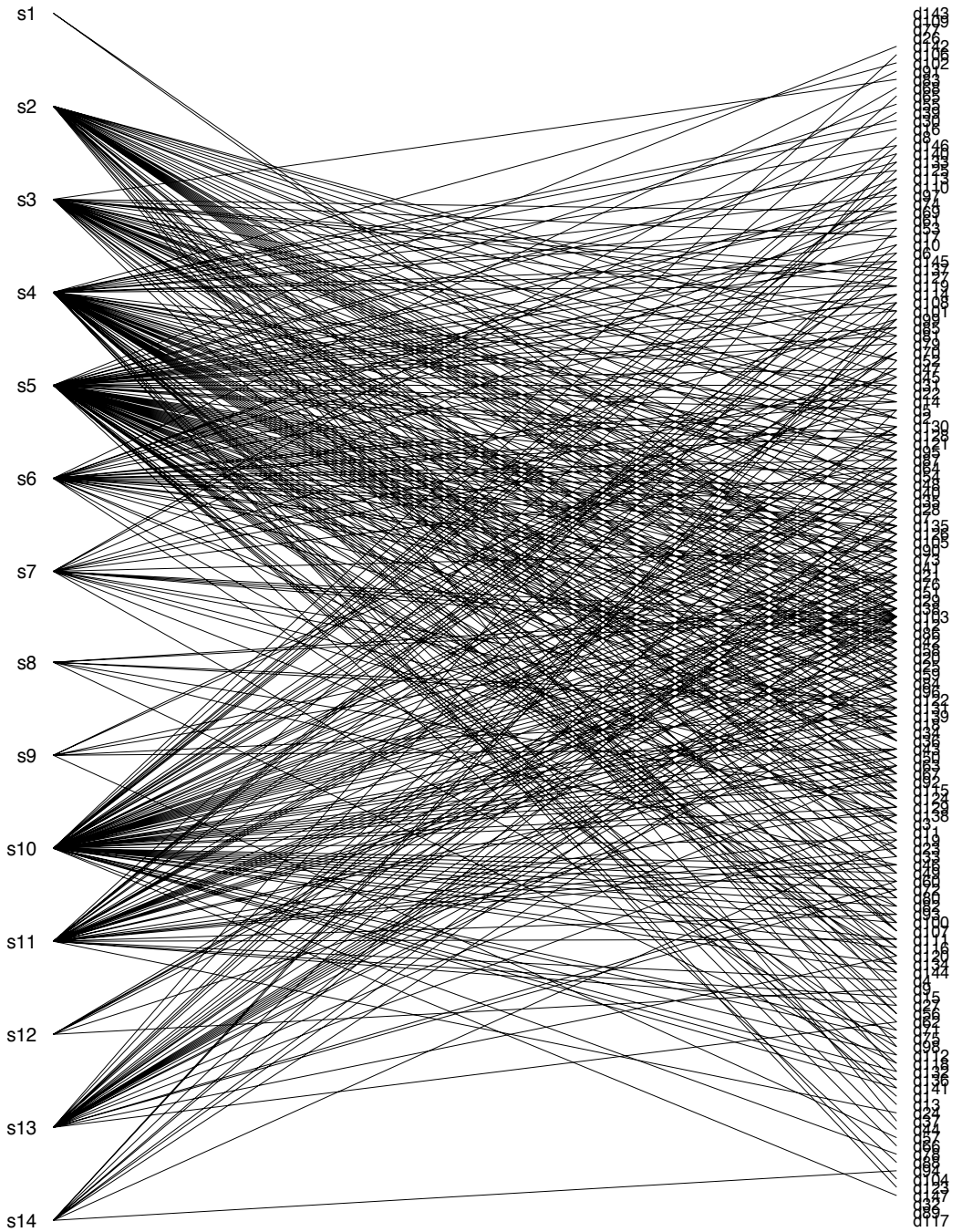

Figure 6-9 Redistributed prerenal azotemia subdomain. Symptom and disorder labels correspond to those listed in appendix D.

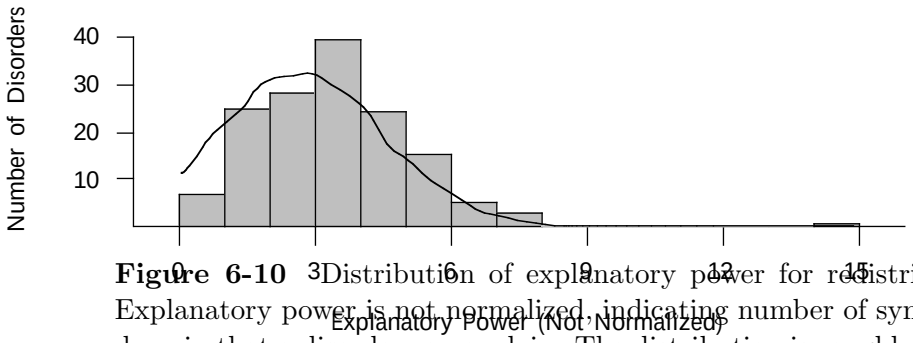

Figure 6-10 Distribution of explanatory power for redistributed subdomain. Explanatory power is not normalized, indicating number of symptoms in the subdomain that a disorder can explain. The distribution is roughly normal, compared with the bimodal distribution of the original subdomain.

Structured subdomain, candidate generation		
	Slope	Inverse slope
Nodes kept	0.63	1.59
Nodes expanded	0.59	1.70
Running time	0.52	1.92

These results show that the absence of domain structure increases the space and time required for candidate generation by powers of 1.59 and 1.92, respectively. Therefore, candidate generation also exploits domain structure.

However, as we have shown, decompositional search exploits structure relatively more. The effect of structure on the space and time efficiency of decompositional search is given below:

Structured subdomain, decompositional search		
	Slope	Inverse slope
Nodes kept	0.25	4.0
Nodes expanded	0.32	3.1
Running time	0.32	3.1

These values are substantially greater than those for candidate generation, indicating the relative extent to which decompositional search exploits domain structure.

6.6 Decomposition of a Subdomain

It is interesting that domain structure appears in such a specific example. We would expect structure at a high levels of structure, such as organ systems. But although the symptoms related to prerenal azotemia are in the

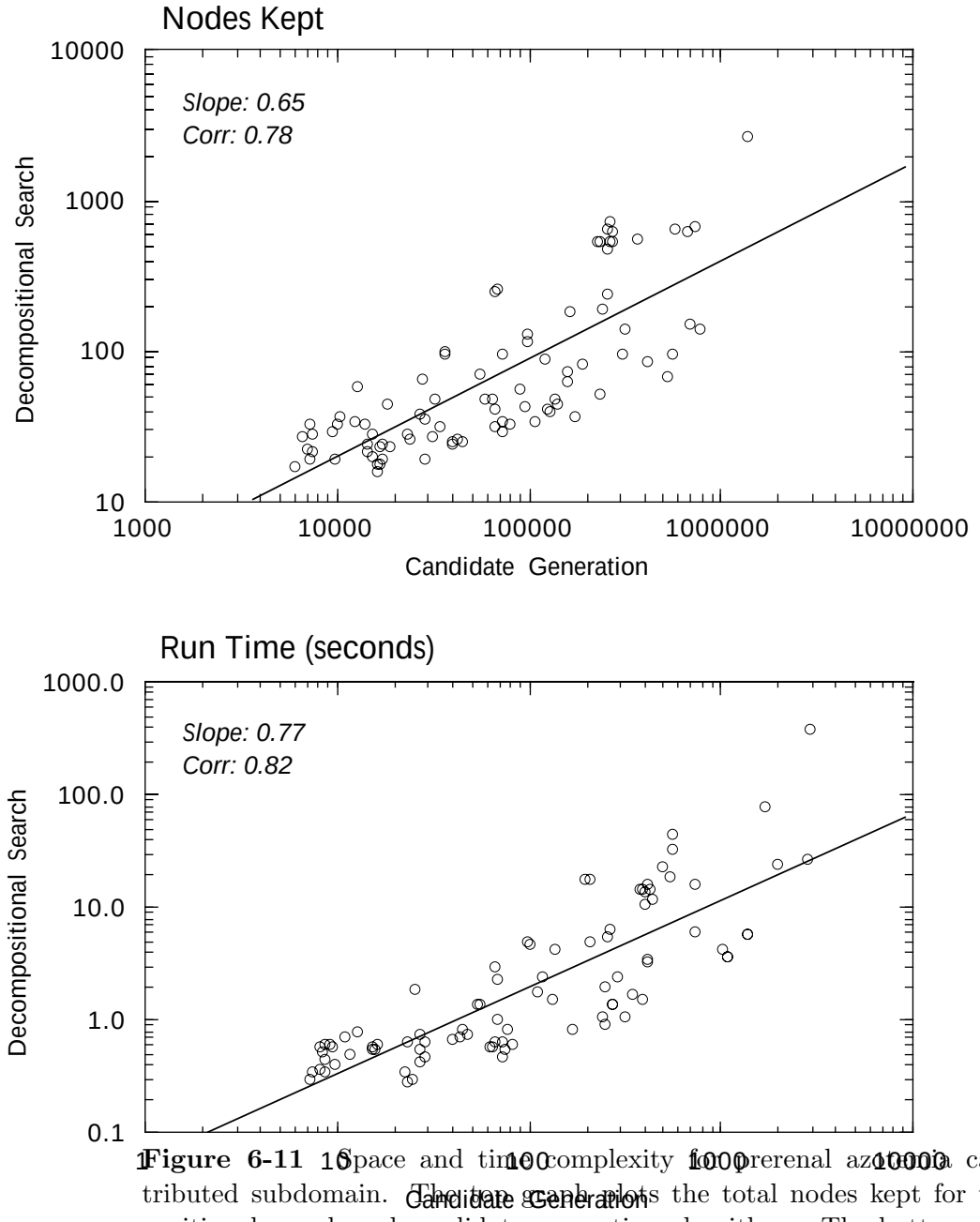

Figure 6-11 Space and time complexity for prerenal azotemia cases, redistributed subdomain. The top graph plots the total nodes kept for the decompositional search and candidate generation algorithms. The bottom graph plots the run time in seconds for the two algorithms.

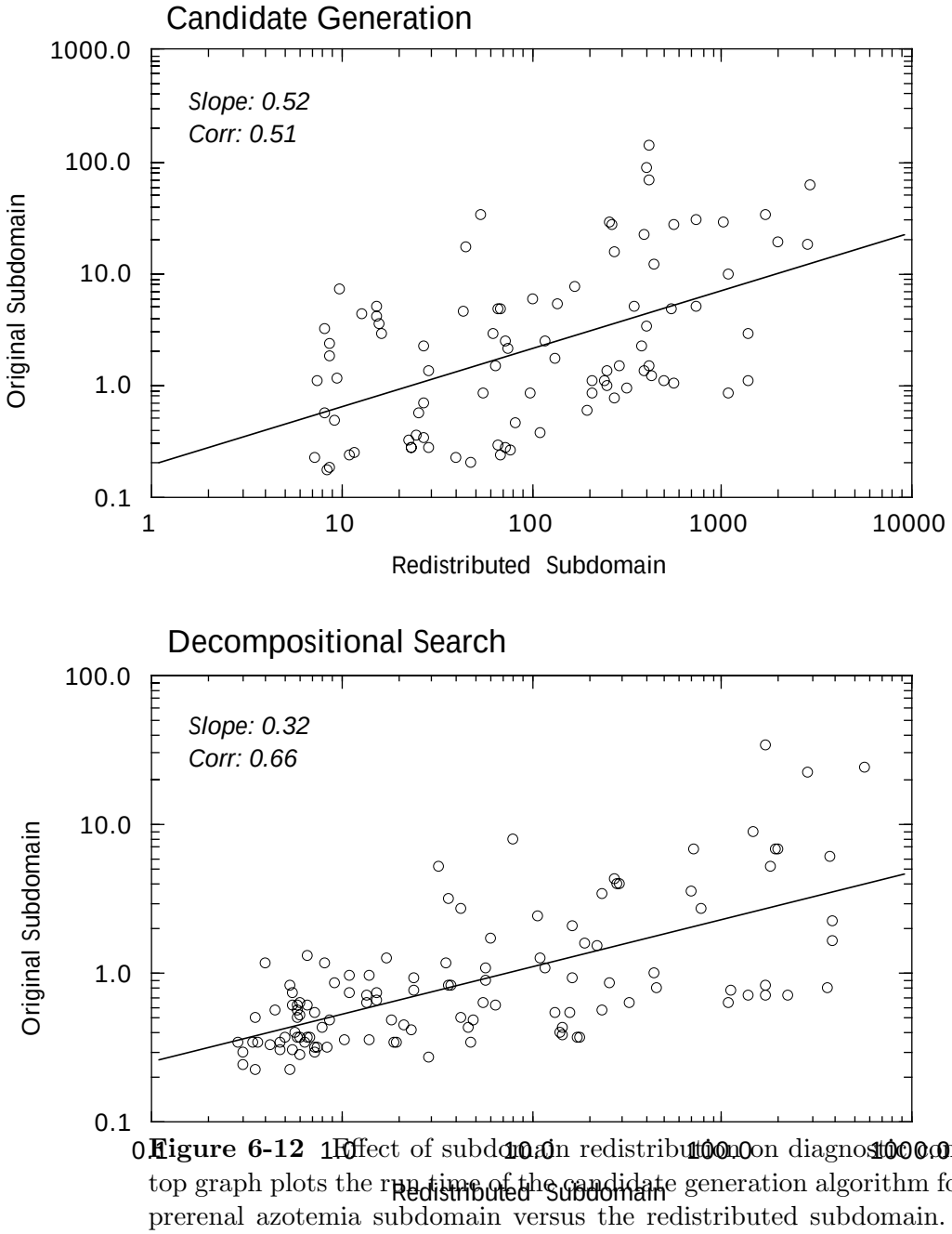

Figure 6-12 Effect of subdomain redistribution on diagnostic complexity. The top graph plots the run time of the candidate generation algorithm for the original prerenal azotemia subdomain versus the redistributed subdomain. The bottom graph plots the same results for the decompositional search algorithm.

same organ system, they still exhibit structure. To elicit this structure further, we now decompose the entire prerenal azotemia subdomain. This task contrasts with the experimental runs performed so far, where only subsets of the possible effects of prerenal azotemia have been decomposed.

Decomposition of all 14 possible effects of prerenal azotemia yields 6 coherent decompositions, listed in figure 6-13. These decompositions give some insight into the structure of the prerenal azotemia subdomain. The best decomposition places all symptoms into one cluster, with a differential containing only "Prerenal Azotemia". There are no coherent decompositions with two clusters, suggesting that the one-cluster decomposition is much more plausible than the others.

Nevertheless, the structure of the other decompositions is informative. For example, the cluster $(s_1 s_2 s_3 s_5 s_6 s_{11})$ recurs in all decompositions other than the first one. The symptoms in this cluster are:

s_1	Azotemia of two weeks duration or less
s_2	Creatinine clearance decreased
s_3	Creatinine serum 3 to 10 mg/dl
s_5	Dehydration
s_6	Mouth mucosa dry (Xerostomia)
s_{11}	Urea nitrogen serum 60 to 100

and its differential is {"Acute Renal Failure"}. This cluster indicates that acute renal failure constitutes a distinct syndrome for the symptoms associated with prerenal azotemia. In fact, prerenal azotemia and acute renal failure are causally related diseases, as *Harrison's Principles of Internal Medicine* attests [78, p. 1145]: "Prerenal azotemia causes 40 to 80 percent of cases of acute renal failure."

Another interesting syndrome appears in decomposition $\mathcal{C}_5$ as the cluster $(s_7 s_9 s_{14})$, which consists of the symptoms

s_7	Oliguria hx
s_9	Sodium urine less than 20 meq per day
s_{14}	Urine specific gravity gtr than 1.020

This cluster has the differential diagnosis {"Cardiac Failure Left Chronic Congestive", "Cardiac Failure Right Chronic Congestive"}, indicating that these two types of heart failure are similar. In heart failure, a complex sequence of hemodynamic and hormonal adjustments cause the kidneys to retain sodium and water, resulting in poor urinary output, low urinary sodium, and concentrated urine.

$\mathcal{C}_1 = (s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14})$
Differentials $\langle 1 \rangle$: {"Prerenal Azotemia"}

$\mathcal{C}_2 = (s_9)\ (s_1 s_2 s_3 s_5 s_6 s_{11})\ (s_2 s_3 s_4 s_7 s_8 s_{10} s_{11} s_{12} s_{13} s_{14})$
Differentials $\langle 6 \times 1 \times 1 \rangle$: {"Aldosteronism Primary" "Aldosteronism Secondary" "Cardiac Failure Left Chronic Congestive" "Cardiac Failure Right Congestive" "Constrictive Pericarditis" "Renal Failure Secondary To Liver Disease"} × {"Renal Failure Acute"} × {"Glomerulonephritis Acute"}

$\mathcal{C}_3 = (s_7)\ (s_8)\ (s_1 s_2 s_3 s_5 s_6 s_{11})\ (s_2 s_3 s_4 s_9 s_{10} s_{11} s_{12} s_{13} s_{14})$
Differentials $\langle 16 \times 5 \times 1 \times 1 \rangle$: {"Arteriolar Nephrosclerosis Malignant" "Cardiac Failure Left Chronic Congestive" "Cardiac Failure Right Congestive" "Glomerulonephritis Advanced Chronic" "Glomerulonephritis Rapidly Progressive" "Goodpasture Syndrome" "Heat Exhaustion" "Iga Nephropathy" "Lupus Nephritis" "Progressive Systemic Sclerosis Involving Kidneys" "Renal Artery Stenosis" "Renal Leptospirosis" "Renal Thrombotic Thrombocytopenic Purpura" "Renal Vasculitis" "Staphylococcal Scarlet Fever" "Tubular Necrosis Acute"} × {"Analgesic Nephropathy" "Diabetic Ketoacidosis" "Hypokalemic Nephropathy" "Renal Failure Chronic" "Renal Tubular Acidosis Proximal"} × {"Renal Failure Acute"} × {"Renal Failure Secondary To Liver Disease"}

$\mathcal{C}_4 = (s_8)\ (s_9)\ (s_1 s_2 s_3 s_5 s_6 s_{11})\ (s_2 s_3 s_4 s_7 s_{10} s_{11} s_{12} s_{13} s_{14})$
Differentials $\langle 5 \times 6 \times 1 \times 3 \rangle$: {"Analgesic Nephropathy" "Diabetic Ketoacidosis" "Hypokalemic Nephropathy" "Renal Failure Chronic" "Renal Tubular Acidosis Proximal"} × {"Aldosteronism Primary" "Aldosteronism Secondary" "Cardiac Failure Left Chronic Congestive" "Cardiac Failure Right Congestive" "Constrictive Pericarditis" "Renal Failure Secondary To Liver Disease"} × {"Renal Failure Acute"} × {"Arteriolar Nephrosclerosis Malignant" "Iga Nephropathy" "Renal Vasculitis"}

$\mathcal{C}_5 = (s_8)\ (s_1 s_2 s_3 s_5 s_6 s_{11})\ (s_7 s_9 s_{14})\ (s_2 s_4 s_{10} s_{12} s_{13} s_{14})$
Differentials $\langle 5 \times 1 \times 2 \times 5 \rangle$: {"Analgesic Nephropathy" "Diabetic Ketoacidosis" "Hypokalemic Nephropathy" "Renal Failure Chronic" "Renal Tubular Acidosis Proximal"} × {"Renal Failure Acute"} × {"Cardiac Failure Left Chronic Congestive" "Cardiac Failure Right Congestive"} × {"Arteriolar Nephrosclerosis Malignant" "Iga Nephropathy" "Renal Failure Secondary To Liver Disease" "Renal Vasculitis" "Toxemia Of Pregnancy"}

$\mathcal{C}_6 = (s_7)\ (s_8)\ (s_9)\ (s_1 s_2 s_3 s_5 s_6 s_{11})\ (s_2 s_4 s_{10} s_{12} s_{13} s_{14})$
Differentials $\langle 11 \times 5 \times 3 \times 1 \times 1 \rangle$: {"Glomerulonephritis Advanced Chronic" "Glomerulonephritis Rapidly Progressive" "Goodpasture Syndrome" "Heat Exhaustion" "Lupus Nephritis" "Progressive Systemic Sclerosis Involving Kidneys" "Renal Artery Stenosis" "Renal Leptospirosis" "Renal Thrombotic Thrombocytopenic Purpura" "Staphylococcal Scarlet Fever" "Tubular Necrosis Acute"} × {"Analgesic Nephropathy" "Diabetic Ketoacidosis" "Hypokalemic Nephropathy" "Renal Failure Chronic" "Renal Tubular Acidosis Proximal"} × {"Aldosteronism Primary" "Aldosteronism Secondary" "Constrictive Pericarditis"} × {"Renal Failure Acute"} × {"Toxemia Of Pregnancy"}

Figure 6-13 Coherent decompositions of the prerenal azotemia subdomain. The symptom labels correspond to those listed in appendix D. Differential diagnoses and their sizes are listed in sequence, matching that of the symptom clusters.

Finally, several differential diagnoses group together the disorders "Arteriolar Nephrosclerosis Malignant", "IgA Nephropathy", and "Renal Vasculitis". These diseases are similar in that they are vascular diseases of the kidney, causing renal damage by high blood pressures in the arteries supplying the kidney.

Thus, this experiment provides additional evidence for the presence of decompositional structure in diagnostic domains, even within a subdomain created by a single target disorder. The resulting "micro-syndromes" correspond to natural groupings also found in medical textbooks. Decompositional search found these symptom clusters and disease groupings solely from the individual patterns of causal links in the knowledge base. It did not rely on abstractions that were explicitly encoded in the knowledge base. This type of abstraction, called *dynamic abstraction*, allows decompositional search to use abstractions inherent in a knowledge base, even when such abstractions are not known beforehand. Dynamic abstraction also suggests that decompositional search might be applied to other domains or subdomains to discover underlying domain structure.

Chapter 7

Probabilistic Decompositional Search

. . . when you have eliminated the impossible, whatever remains,
however improbable, must be the truth.

— Arthur Conan Doyle, *The Sign of Four* (1890)

The decompositional search algorithm presented so far is categorical; it does not consider any information about the likelihood of disorders or symptoms. This chapter extends decompositional search to allow such probabilistic information. This is especially important in fields such as medicine where the probabilities of diseases and causal relationships range over several orders of magnitude.

This chapter derives a set of formulas for the probability of a candidate set, a candidate, and a task, conditioned on a set of positive and negative symptoms. We also compute the probability that the single-fault assumption is true. To derive these results, we first establish a precise semantics for these quantities. In particular, we clarify the probabilistic concepts of events, instances, link probabilities, causal probabilities, and non-causation probabilities.

7.1 Probabilistic Knowledge Bases

A probability is a function p that assigns a number between 0 and 1 to an event. An event is the outcome or state information for a set of objects in some universe. For instance, let a universe consist of a jar of n balls, labeled $b_1, \ldots, b_n$. Each ball can be red or white; these represent their possible states. An event might be the condition that ball b_i is red.

Intuitively, a probability measures the chance or likelihood of an event. However, probability theory depends only on the following axioms, where Ω is the certain event, one that is always true:

1. The probability of an event A is positive:

$$p(A) \geq 0$$

2. The probability of the certain event equals 1:

$$p(\Omega) = 1$$

3. If events A and B are mutually exclusive, that is, they cannot both occur simultaneously, then the probability that either event will occur is the sum of their individual probabilities:

$$p(A + B) = p(A) + p(B)$$

These axioms provide sufficient basis for computing probabilities.

7.1.1 Prior Probabilities

To bring probability theory into the realm of diagnosis, consider the universe U_D of all disorders d in the knowledge base. We assume that our knowledge base contains all possible disorders; if it is incomplete, we add a disorder called "Other" that stands for all disorders not in the knowledge base. Then the state of each disorder d is either positive or negative. We denote these events respectively as d^+ and d^-. The state of a disorder may also be left unspecified, which we denote by the disjunction "$d^+ \vee d^-$".

An event can specify the outcomes for a set D of disorders. The event that every disorder in D is positive is represented as D^+, and that every disorder in D is negative as D^-. The event that one or more disorder in D is present is represented simply as D. Note that the event D is the opposite of the event D^-. Like sets, events can be combined. Suppose A and B are events; then "$A + B$" means the disjunctive event $A \vee B$, and "AB" or "A, B" means the conjunctive event $A \wedge B$. For instance, the event where d_1 is present and d_2 is absent is denoted as $d_1^+ d_2^-$.

A probabilistic knowledge base requires two types of quantities: prior probabilities and link probabilities. The *prior probability* $p(d^+)$ for each disorder d is the probability that d is present, given no other information. We denote this quantity by $p(d^+)$. The probability that a disorder is absent is $p(d^-) = 1 - p(d^+)$. We will also use the following simplified notations for these quantities:

$$p_d^+ \equiv p(d^+) \tag{7.1}$$
$$p_d^- \equiv p(d^-) \tag{7.2}$$

We can combine prior probabilities to obtain the probabilities of sets of disorders:

$$p(D^+) = \prod_{d \in D} p_d^+ \tag{7.3}$$

$$p(D^-) = \prod_{d \in D} p_d^- \tag{7.4}$$

$$p(D) = 1 - \prod_{d \in D} p_d^- \tag{7.5}$$

In these equations, we assume that disorders occur independently. In other words, we assume that for any pair of disorders d_1 and d_2,

$$p(d_1^+ d_2^+) = p(d_1^+)p(d_2^+) \tag{7.6}$$

This is called the *disorder independence assumption*.

While most events specify only the state of some disorders in the universe, leaving the rest unspecified, other events specify the state of all disorders in the universe; we call such an event a *disorder instance*. For example, consider the disorder instance that all disorders in a given set D are present, while all other disorders in U_D are absent. We denote this disorder instance as $D^+(U_D - D)^-$. The probability of this disorder instance is

$$p(D^+(U_D - D)^-) = \prod_{d \in D} p_d^+ \prod_{d \in U_D - D} p_d^-$$

In addition to a universe U_D of disorders, a diagnostic knowledge base also contains a universe U_S of symptoms. A symptom can be either positive (present) or negative (absent). We denote the set of positive symptoms by P and the negative symptoms by N. Using the same notation as for disorders, we therefore denote a case by the event P^+N^-. Note that a case is not a symptom instance; it does not specify whether symptoms in $U_S - (P \cup N)$ are positive or negative.

7.1.2 Link Probabilities

In addition to having a prior probability attached to each disorder, a probabilistic knowledge base has a *link probability* attached to each link between a disorder and symptom. The meaning of a link probability is based on the concept of conditioning. Conditioning changes the probability of an event based on background information. Suppose we know that event A has occurred. This information changes the probability of B from $p(B)$ to $p(B \mid A)$, or the conditional probability of B given A. The conditional probability is defined as

$$p(B \mid A) = \frac{p(AB)}{p(A)} \tag{7.7}$$

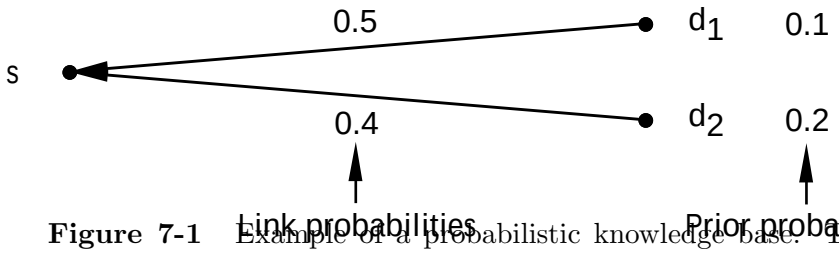

Figure 7-1 Example of a probabilistic knowledge base. This example corresponds semantically to two destroyers, d_1 and d_2, firing upon a ship s. Prior probabilities correspond to the probability of the destroyer being in the vicinity of the ship. Link probabilities correspond to the destroyers' firing accuracy.

A link probability results from conditioning on the presence of exactly one disorder. Suppose that some symptom s is positive and that some disorder d is present. We assume that s can only be caused by a disorder that is present; that is, it cannot be positive without cause. Then the probability that d causes s is

$$p(s^+ \mid d^+(U_D - \{d\})^-)$$

the link probability of s^+ given that only d is present. The conditioning event "$d^+(U_D - \{d\})^-$" effectively excludes all disorders other than d, thereby focusing on the contribution of d alone towards causing s. Link probabilities are given as primitive values by the domain expert, and to simplify the notation, we will use special symbols for them:

$$c_{ds} \equiv p(s^+ \mid d^+(U_D - \{d\})^-) \tag{7.8}$$

$$\begin{aligned} q_{ds} &\equiv p(s^- \mid d^+(U_D - \{d\})^-) \\ &= 1 - c_{ds} \end{aligned} \tag{7.9}$$

In a probabilistic knowledge base, each link is labeled with the value c_{ds}. If a link is not present between d and s, then $c_{ds} = 0$.

Since causal links are not directly observable in most domains, conditional probabilities typically assess whether symptoms are present, not whether they are caused by a particular disorder. Consequently, most conditional probabilities are different from link probabilities. To see this difference, consider the following example.

Example Consider the situation in figure 7-1, where a ship s becomes a target for destroyers d_1 and d_2. We are concerned about whether the ship sinks (represented by s^+) or stays afloat (s^-). In this problem, the destroyers

may or may not be present in the vicinity of the ship; suppose they have probabilities $p(d_1^+) = 0.1$ and $p(d_2^+) = 0.2$ of being present, respectively. The destroyers also differ in their firing accuracy. When present, destroyer 1 has a 0.5 probability of sinking the ship, while destroyer 2 has a 0.4 probability of sinking the ship. These are the link probabilities $p(s^+ \mid d_1^+ d_2^-)$ and $p(s^+ \mid d_2^+ d_1^-)$ of the ship sinking.

The ship sinks whenever a destroyer is present and that destroyer successfully torpedoes the ship. If both destroyers are present, the ship has the following chance of sinking:

$$p(s^+ \mid d_1^+ d_2^+) = 1 - (1 - 0.5)(1 - 0.4) = 0.7$$

This value was computed by finding the probability that both destroyers missed and subtracting that from 1. Now, by summing the cases where the other destroyer is present or absent, we can determine the conditional probabilities that the ship sinks:

$$\begin{aligned} p(s^+ \mid d_1^+) &= p(s^+ \mid d_1^+ d_2^+)p(d_2^+) + p(s^+ \mid d_1^+ d_2^-)p(d_2^-) \\ &= (0.7)(0.2) + (0.5)(0.8) = 0.54 \\ p(s^+ \mid d_2^+) &= p(s^+ \mid d_2^+ d_1^+)p(d_1^+) + p(s^+ \mid d_2^+ d_1^-)p(d_1^-) \\ &= (0.7)(0.1) + (0.4)(0.9) = 0.43 \end{aligned}$$

Compare these with the link probabilities:

$$\begin{aligned} p(s^+ \mid d_1^+ d_2^-) &= 0.5 \\ p(s^+ \mid d_2^+ d_1^-) &= 0.4 \end{aligned}$$

Note that the conditional probability of the ship sinking is higher than the link probability. This relationship holds because the conditional probability includes those events where the other destroyer sinks the ship. Also, note that the conditional probabilities are harder to compute. This is because link probabilities are given as primitives, while conditional probabilities have to computed over combinations of disorder instances. ∎

7.2 Causation and Probability

Given the prior and link probabilities in a diagnostic knowledge base, we now consider the use of probabilities in reasoning about particular diagnostic

cases. In a particular case, we usually cannot condition on the presence or absence of a disorder, since the point of diagnosis is to determine that information. On the other hand, we can condition on the presence and absence of symptoms, and in fact, the main use of symptomatic information is to change the likelihood of disorders in the universe.

Although we do not condition on disorders, we hypothesize about their presence and absence. Furthermore, in decompositional search, we hypothesize about how disorders cause symptoms. In this section, we consider how to compute the probability of hypothesized events of causation and non-causation.

7.2.1 Causal Probabilities

A *causal probability* is the probability that a given disorder or set of disorders causes a given symptom or set of symptoms. It measures the event that a particular link or set of links in the associative knowledge base "fire". Thus, a causal event means not only that a symptom is positive but that a particular disorder caused it to be positive. This event is denoted by $d \longrightarrow s$. (We do not need "+" superscripts, because the event already implies that both s and d are present.) For this causal event to occur, a disorder must both be present and successfully cause the symptom. Thus, we define causal probability as:

$$\begin{aligned} p(d \longrightarrow s) &\equiv p(s^+ \mid d^+(U_D - \{d\})^-)p(d^+) \\ &= p_d^+ c_{ds} \end{aligned} \tag{7.10}$$

We multiply by $p(d^+)$ instead of $p(d^+(U_D - \{d\})^-)$ because we include the possibility that other disorders may be present, although we still assume that d causes s. In other words, we make no assumptions about the state of other disorders or links. Note that causal probabilities differ from link probabilities, because link probabilities are conditioned on a disorder instance, while causal probabilities are not conditioned.

Example Consider again our example of the ship and two destroyers. We wish to know that probability that a destroyer sinks a ship, without knowing whether the destroyer is present or not. This event requires both that the destroyer be present and that its torpedo sinks the ship. In our example, we

can compute the probabilities that the ship is sunk by each destroyer.

$$\begin{aligned} p(d_1 \longrightarrow s) &= (0.1)(0.5) = 0.05 \\ p(d_2 \longrightarrow s) &= (0.2)(0.4) = 0.08 \end{aligned}$$ ∎

In addition, we will find the converse of a causal event useful. This event is expressed by $d \not\longrightarrow s$, which means that d does not cause s. A disorder d can fail to cause s if it is absent or if it is present and its link does not fire:

$$\begin{aligned} p(d \not\longrightarrow s) &= 1 - p(d \longrightarrow s) \\ &= p_d^- + p_d^+ q_{ds} \end{aligned} \tag{7.11}$$

Causal probabilities are especially useful in the decompositional search approach, because a problem decomposition essentially assigns a causal structure between differential diagnoses and symptom clusters. Causal probabilities help determine how feasible this structure is. In order to use causal probabilities for problem decompositions, we need to generalize them to sets of symptoms. This generalization requires an additional assumption, namely, that that each disorder causes each symptom independently. In our example, this is analogous to saying that each torpedo is independent of the others:

$$p(s_1^+ s_2^+ \mid d_1^+ d_2^-) = p(s_1^+ \mid d_1^+ d_2^-) p(s_2^+ \mid d_1^+ d_2^-) \tag{7.12}$$

This is the *causal independence assumption*. These assumptions plus the disorder independence assumption in 7.6 are made in most other probabilistic work. In particular, they are used by the belief networks community, where they are called the *noisy-or assumptions* [51].

With these assumptions, we can now compute the probability that a set of disorders disjunctively causes a set of symptoms. We denote this event by $D \stackrel{\vee}{\longrightarrow} S$, which is defined formally to mean:

$$D \stackrel{\vee}{\longrightarrow} S \equiv \exists d \in D.\ \forall s \in S.\ d \longrightarrow s$$

The probability of this event is:

$$p(D \stackrel{\vee}{\longrightarrow} S) = 1 - \prod_{d \in D} \left(1 - p_d^+ \prod_{s \in S} c_{ds}\right) \tag{7.13}$$

This probability holds because the causal event is false when every disorder fails to cause all symptoms. A disorder causes all symptoms when it is present and all causal links fire.

Causal probabilities can be conditioned on events. Conditioning introduces our background knowledge into the computation of a causal probabilities. For instance, we may want to compute the probability that d causes s, given that s is positive. This conditional causal probability can be computed as follows:

$$\begin{aligned} p(d \longrightarrow s \mid s^+) &= \frac{p(d \longrightarrow s,\ s^+)}{p(s^+)} \\ &= \frac{p(d \longrightarrow s)}{p(s^+)} \\ &= \frac{p_d^+ c_{ds}}{1 - \prod_{d \in U_D} \left(p_d^- + p_d^+ q_{ds}\right)} \end{aligned} \tag{7.14}$$

The second line follows because $d \longrightarrow s$ implies s^+. This combination of conditioning and causation foreshadows the results derived later in this chapter. These results generalize this calculation to sets of disorders and symptoms and conditions also on negative symptoms.

7.2.2 Non-Causation and Symptom Probabilities

In addition to computing the probability of causal events, we can compute the probability of non-causation, or a failure to cause a set of symptoms. This event $D \not\Longrightarrow S$ means that no element in D causes any element in S. In logical terms:

$$D \not\Longrightarrow S \ \equiv\ \forall d \in D.\ \forall s \in S.\ d \not\longrightarrow s$$

Note that this is different from the event $D \overset{\forall}{\not\longrightarrow} S$ which means that now every element in D causes all elements in S:

$$D \overset{\forall}{\not\longrightarrow} S \ \equiv\ \forall d \in D.\ \exists s \in S.\ d \not\longrightarrow s$$

The probability of the non-causation event is given below:

$$p(D \not\Longrightarrow S) = \prod_{d \in D} \left(p_d^- + p_d^+ \prod_{s \in S} q_{ds}\right) \tag{7.15}$$

This probability holds because each disorder can be considered independently. A disorder fails to cause a set of symptoms when it is absent or when it is present and each causal link fails.

Quantity	Notation
Prior probability	p_d^+, p_d^-
Link probability	c_{ds}, q_{ds}
Causal probability	$p(d \longrightarrow s)$, $p(D \xrightarrow{\vee} S)$
Non-causation probability	$p(D \dashv\!\!\!\vdash\!\!\!\rightarrow A)$

Figure 7-2 Summary of probabilistic notation.

The probability of non-causation is useful because it gives the probability that a set of symptoms is absent. A set of symptoms is absent whenever they are not caused by any disorder. We can therefore consider the non-causation of the entire knowledge base:

$$\begin{aligned} p(S^-) &= p(U_D \dashv\!\!\!\vdash\!\!\!\rightarrow S) \\ &= \prod_{d \in U_D} \left(p_d^- + p_d^+ \prod_{s \in S} q_{ds} \right) \end{aligned} \tag{7.16}$$

By subtracting this result from 1, we get the probability that at least one symptom from a set is present, given no other information:

$$p(S) = 1 - \prod_{d \in U_D} \left(p_d^- + p_d^+ \prod_{s \in S} q_{ds} \right) \tag{7.17}$$

So far, we have introduced several different types of events and computed their associated probabilities. These probabilistic quantities are summarized in figure 7-2. We will use these quantities in the sections that follow.

7.3 Case Probability

In diagnosis, we generate hypotheses to explain a set of evidence. Thus, probabilities in diagnosis are usually of the form $p(H \mid E)$, where H is a hypothesis and E is evidence. In diagnosis the evidence comes from a case that includes positive symptoms P and negative symptoms N. In computing the probability of a hypothesis, we often use Bayes's rule:

$$p(H \mid P^+N^-) = \frac{p(P^+N^- \mid H)p(H)}{p(P^+N^-)} \tag{7.18}$$

The denominator is the probability that a particular case will occur. Unfortunately, it is notoriously difficult to compute. The problem stems from

the positive symptoms, since there are many ways that a set of symptoms can be caused. Previously, we computed $p(S^-)$ and $p(S)$, but computing $p(S^+)$ is more difficult. Indeed, a symptom can be caused by any subset of its possible causes. Since the number of subsets grows exponentially with the number of elements, computing the denominator can be intractable.

Nevertheless, there is a clever technique that can make the computation of the denominator more efficient [27]. This works by trading the combinatorics of disorders for the combinatorics of symptoms. Specifically, the technique uses an inclusion-exclusion principle that turns a set of positive symptoms into alternating subsets of absent symptoms. This gives the case probability as:

$$\begin{aligned} p(P^+N^-) &= \sum_{P_\subset \in 2^P} (-1)^{|P_\subset|} p(P_\subset^- N^-) \\ &= \sum_{P_\subset \in 2^P} (-1)^{|P_\subset|} \prod_{d \in U_D} \left(p_d^- + p_d^+ \prod_{s \in P_\subset \cup N} q_{ds} \right) \end{aligned} \tag{7.19}$$

In this equation, U_D contains all disorders in the universe (or knowledge base), and 2^P denotes the power set of P, so $P_\subset$ represents each subset of positive symptoms P. The term $P_\subset^- N^-$ means the event that all symptoms in $(P_\subset \cup N)$ are absent. Essentially, we begin by making no assumptions on P, so that $P_\subset = \emptyset$, giving rise to the probability $p(N^-)$. Then, we consider all ways that each symptom in P could be absent, in addition to N, and subtract the corresponding probabilities. But this subtracts too much, since it subtracts the probability for each pair of positive symptoms twice. Thus, we add the probabilities that each pair of positive symptoms is absent, along with the negative symptoms N. Again, this overcompensates, so we then subtract the probabilities that each triplet of positive symptoms is absent, and so on.

Computing this denominator requires time on the order of

$$O(2^{\mathcal{P}}(\mathcal{P} + \mathcal{N})\mathcal{U}_D)$$

where $\mathcal{P}$ is the number of positive symptoms, $\mathcal{N}$ is the number of negative symptoms, and $\mathcal{U}_D$ is the number of disorders in the knowledge base. This is computationally expensive, because it is an exponential function of the number of positive symptoms. However, the number of positive symptoms is often relatively small for a particular case, so this complexity may be acceptable for some domains.

Nevertheless, in many situations, we only care about finding the best hypotheses, and not assessing their actual probabilities. In this these applications, the relative probability of a hypothesis is sufficient. The denominator is the same for all hypotheses, so we can simply ignore it in computing relative probabilities.

7.4 Candidate Sets

Previously, we have seen that a problem decomposition entails an initial and final candidate set. The final candidate set is entailed by a problem decomposition after the differential diagnoses are formulated. One way to extend probabilities to decompositional search is to compute the probability of these candidate sets. Recall that the candidate set is given by Cartesian product of differentials:

$$\text{Cands}(\mathcal{C}) \quad = \quad \bigtimes_{C \in \mathcal{C}} \text{Diff}(C)$$

The conditioning event is the set of positive symptoms P and negative symptoms N. The probability of a candidate set can be computed using the definition of conditional probability:

$$p(\text{Cands}(\mathcal{C}) \mid P^+N^-) \quad = \quad \frac{p(P^+N^-\text{Cands}(\mathcal{C}))}{p(P^+N^-)} \tag{7.20}$$

The denominator is the case probability discussed previously, so we need solve only the numerator. To compute the numerator, we use an inclusion-exclusion strategy, similar to that used in solving the case probability:

$$p(P^+N^-\text{Cands}(\mathcal{C})) \quad = \quad \sum_{P_\subset \in 2^P} (-1)^{|P_\subset|}\, p(P_\subset^- N^-\text{Cands}(\mathcal{C})) \tag{7.21}$$

To compute the probability $p(P_\subset^- N^-\text{Cands}(\mathcal{C}))$, we consider separately those disorders in a differential and those not in any differential. Consider a differential $\text{Diff}(C)$. If symptoms in $(P_\subset \cup N)$ are absent, then they cannot be caused by any disorder in $\text{Diff}(C)$. Yet, since $\text{Cands}(\mathcal{C})$ is true, at least one disorder in each differential must be present. The probability that at least one disorder in $\text{Diff}(C)$ is present but does not cause any symptoms in

$P_\subset$ or N is

$$p(\text{Diff}(C) \not\rightarrow P_\subset \cup N,\ \text{Diff}(C)) = \prod_{d \in \text{Diff}(C)} \left(p_d^+ \prod_{s \in P_\subset \cup N} q_{ds} + p_d^- \right) - \prod_{d \in \text{Diff}(C)} p_d^- \tag{7.22}$$

The next step is to consider symptoms not in any differential. If symptoms in $(P_\subset \cup N)$ are absent, then they cannot be caused by any disorder not in a differential. Let these other disorders be defined as

$$U_D^* = U_D - \bigcup_{C \in \mathcal{C}} \text{Diff}(C) \tag{7.23}$$

where U_D is the universe of disorders in the knowledge base. The probability that these disorders do not cause symptoms in $P_\subset$ or N is

$$p(U_D^* \not\rightarrow P_\subset \cup N) = \prod_{d \in U_D^*} \left(p_d^- + p_d^+ \prod_{s \in P_\subset \cup N} q_{ds} \right) \tag{7.24}$$

Finally, we can put the parts together. The probability of a candidate set is

$$p(\text{Cands}(\mathcal{C}) \mid P^+ N^-) = \frac{\sum_{P_\subset \in 2^P} (-1)^{|P_\subset|} \left[\prod_{C \in \mathcal{C}} p(\text{Diff}(C) \not\rightarrow P_\subset \cup N,\ \text{Diff}(C)) \right] p(U_D^* \not\rightarrow P_\subset \cup N)}{p(P^+ N^-)} \tag{7.25}$$

where terms are substituted from equations 7.22 and 7.24.

Computational Complexity The computational complexity of the above set of formulas is:

Numerator (equation 7.21):	$O(2^{\mathcal{P}} \mathcal{N} \mathcal{U}_D)$
Denominator (equation 7.19):	$O(2^{\mathcal{P}} (\mathcal{P} + \mathcal{N}) \mathcal{U}_D)$

The denominator term dominates, so the overall computational complexity is $O(2^{\mathcal{P}}(\mathcal{P} + \mathcal{N})\mathcal{U}_D)$, which is exponential in the number of positive symptoms but linear in the number of negative symptoms and size of the knowledge base. The numerator has roughly the same complexity as the denominator, so there is little advantage in computing relative probabilities between decompositions as opposed to absolute probabilities.

7.5 Candidates

The probability given in the previous section evaluates a *collection* of candidates, namely the Cartesian product of the differential diagnoses in a problem decomposition. In this section, we specialize this formula to obtain the probability of a *single* candidate D. We merely assume that the differential diagnoses $\text{Diff}(C)$ in some decomposition $\mathcal{C}$ are all singletons. Then, the candidate D is the only element in the Cartesian product:

$$\bigtimes_{C \in \mathcal{C}} \text{Diff}(C) \quad = \quad \{D\}$$

The result for this limiting case of singleton differentials is:

$$p(D^+ \mid P^+ N^-) = \frac{1}{p(P^+ N^-)} \sum_{P_\subset \in 2^P} (-1)^{|P_\subset|} \prod_{d \in D} \left(p_d^+ \prod_{s \in P_\subset \cup N} q_{ds} \right) \prod_{d \in U_D - D} \left(p_d^- + p_d^+ \prod_{s \in P_\subset \cup N} q_{ds} \right) \tag{7.26}$$

The formula for the denominator, as before, can be found in equation 7.19. In the absence of any evidence, where $P = N = \emptyset$, this expression reduces to the expected result for the prior probability of a candidate:

$$p(D^+) \quad = \quad \prod_{d \in D} p_d^+$$

7.6 Tasks

Recall that a task contains a symptom cluster C and its associated differential diagnosis $\text{Diff}(C)$. The meaning of a task is that each disorder in $\text{Diff}(C)$ is capable of explaining all symptoms in C. Thus, a task expresses a causal event: $\text{Diff}(C) \xrightarrow{\vee} C$. In computing the probability of a task, we assume that no other tasks are present. The case of multiple tasks complicates things considerably. To see why, let us try to compute the probability of a decomposition. Unfortunately, there is no guarantee that the tasks will be independent:

$$p\left(\bigwedge_{C \in \mathcal{C}} (\text{Diff}(C) \xrightarrow{\vee} C) \mid P^+ N^- \right) \quad \neq \quad \prod_{C \in \mathcal{C}} p(\text{Diff}(C) \xrightarrow{\vee} C \mid C^+ N^-)$$

In fact, knowledge about a symptom in P or N induces dependence on all of its possible causes, so that tasks may not be independent. Nevertheless, task independence may be a useful approximation. This is especially true since a problem decomposition attempts to separate symptoms into independent clusters. We can use a task independence assumption along with the results of this section to approximate the probability of a decomposition.

With this in mind, we proceed to compute the probability of a task, independent of other tasks. We apply the definition of conditional probability:

$$\begin{aligned} p(\text{Diff}(C) \xrightarrow{\vee} C \mid C^+N^-) &= \frac{p(\text{Diff}(C) \xrightarrow{\vee} C,\ C^+N^-)}{p(C^+N^-)} \\ &= \frac{p(\text{Diff}(C) \xrightarrow{\vee} C,\ N^-)}{p(C^+N^-)} \\ &= \frac{p(\text{Diff}(C) \xrightarrow{\vee} C \mid N^-)p(N^-)}{p(C^+N^-)} \end{aligned} \quad (7.27)$$

The second line is justified because $\text{Diff}(C) \xrightarrow{\vee} C$ implies C^+. The denominator is the usual case probability, discussed previously. We now consider the two factors in the numerator separately.

The second factor in the numerator of equation 7.27 is relatively simple. It is given by:

$$p(N^-) = \prod_{d \in U_D} \left(p_d^- + p_d^+ \prod_{s \in N} q_{ds} \right) \quad (7.28)$$

To compute the first factor in the numerator, we rely on the formula for the causal event $p(\text{Diff}(C) \xrightarrow{\vee} C)$, given in equation 7.13. However, the probability of each disorder is modified by the negative symptoms:

$$p(\text{Diff}(C) \xrightarrow{\vee} C \mid N^-) = 1 - \prod_{d \in \text{Diff}(C)} \left(1 - p(d^+|N^-) \prod_{s \in C} c_{ds} \right) \quad (7.29)$$

This equation requires the probability of a disorder conditioned on the negative symptoms, that is, $p(d^+|N^-)$:

$$p(d^+ \mid N^-) = \frac{p(d^+N^-)}{p(N^-)}$$

$$
\begin{aligned}
&= \frac{p_d^+ \prod_{s \in N} q_{ds} \prod_{d_i \in U_D - \{d\}} \left(p_{d_i}^- + p_{d_i}^+ \prod_{s \in N} q_{d_i s}\right)}{\prod_{d_i \in U_D} \left(p_{d_i}^- + p_{d_i}^+ \prod_{s \in N} q_{d_i s}\right)} \\
&= \frac{p_d^+ \prod_{s \in N} q_{ds}}{p_d^- + p_d^+ \prod_{s \in N} q_{ds}} \qquad (7.30)
\end{aligned}
$$

The final step results from canceling all terms in numerator and denominator except for the remaining case of d.

Putting the parts from equations 7.27, 7.28, 7.29, and 7.30 together yields the probability of a task:

$$
p(\text{Diff}(C) \xrightarrow{\vee} C \mid C^+ N^-) = \frac{\left[1 - \prod_{d \in \text{Diff}(C)} \left(1 - \frac{p_d^+ \prod_{s \in C} c_{ds} \prod_{s \in N} q_{ds}}{p_d^- + p_d^+ \prod_{s \in N} q_{ds}}\right)\right] \prod_{d \in U_D} \left(p_d^- + p_d^+ \prod_{s \in N} q_{ds}\right)}{p(C^+ N^-)} \qquad (7.31)
$$

where the denominator is the case probability in equation 7.19, applied to the cluster C.

Computational Complexity The computational complexity of the above equations is:

Numerator term 1 (equation 7.29):	$O(\mathcal{PNU}_D)$
Numerator term 2 (equation 7.28):	$O(\mathcal{NU}_D)$
Numerator:	$O(\mathcal{PNU}_D)$
Denominator:	$O(2^{\mathcal{P}}(\mathcal{P}+\mathcal{N})\mathcal{U}_D)$

Although computing the denominator requires time exponential with the number of present symptoms, the numerator requires only polynomial time. Thus, the relative likelihood of a task can be computed quickly.

7.7 Single-Fault Assumption

The single-fault assumption was made in early work in diagnosis [11]. This assumption simplified diagnosis by eliminating the combinatorial problems of multiple disorders. This assumption is reasonable in domains where faults are unlikely, since each fault usually lowers the overall probability of a candidate by an additional prior probability. It is also a good assumption when a single

fault can be spotted immediately, either because the device is constantly monitored or because the device fails completely or catastrophically when a fault does occur. When faults cannot accumulate, a single-fault assumption is more likely.

Interestingly, the likelihood of the single-fault assumption can be computed using the probability of a task derived in the previous section. The single-fault assumption postulates a single cause for all positive symptoms. In other words, the assumption hypothesizes a single-task decomposition with a cluster containing the positive symptoms and a differential containing the universe of disorders. Therefore, we can substitute P for C and U_D for Diff(C) in equation 7.31:

$$p(U_D \xrightarrow{\vee} P \mid P^+N^-) = \frac{\left[1 - \prod_{d \in U_D} \left(1 - \frac{p_d^+ \prod_{s \in P} c_{ds} \prod_{s \in N} q_{ds}}{p_d^- + p_d^+ \prod_{s \in N} q_{ds}}\right)\right] \prod_{d \in U_D} \left(p_d^- + p_d^+ \prod_{s \in N} q_{ds}\right)}{p(P^+N^-)} \tag{7.32}$$

This gives the probability that one disorder in the knowledge base explains all of the given positive symptoms. Note that if there is no disorder in the knowledge base that can possibly explain all the symptoms, then for each d in U_D there exists some s in P such that $c_{ds} = 0$. This would then reduce the value of the numerator to zero, and hence yield a zero probability for the single-fault assumption, as expected.

7.8 Relation to Other Work

7.8.1 Probabilistic Candidate Generation

Previous work on candidate generation also considers the probability of a candidate. But these results for the probability of a candidate are different from the formula derived here. Previous work suffers from various assumptions about the nature of candidates. For example, some researchers evaluate candidates not with conditional probabilities, but with prior probabilities. This is the approach taken by Hamscher in his XDE system [24]. He computes the prior probability of a candidate; moreover, he interprets a candidate as

a disorder instance rather than an event:

$$p(D^+(U_D - D)^-) \;=\; \prod_{d \in D} p_d^+ \prod_{d \in U_D - D} p_d^-$$

This probabilistic assessment is poor because it does not account for the role of symptomatic evidence in changing the probability of disorders.

Another analysis is made by deKleer and Williams in their GDE system [14]. They use the following formula for a conditional probability of a candidate:

$$p(D^+ \mid s^i) \;=\; \begin{cases} 0 & \text{if } D^+ \text{ predicts } s \neq s^i \\ p(D^+)/p(s^i) & \text{if } D^+ \text{ predicts } s = s^i \\ p(D^+)/mp(s^i) & \text{if } D^+ \text{ predicts nothing for } s \end{cases}$$

where a symptom can take states $s_1, \ldots, s_m$. Unfortunately, this equation does not apply well to diagnostic knowledge bases. In a diagnostic knowledge base, $m = 2$ and the states are s^+ and s^-. The noisy-or assumptions preclude the first case in the above equation, since we cannot predict the absence of a symptom s^-. That is, only the presence of a symptom can be predicted by a candidate; other disorders not in the candidate may still be present, so that the absence of symptom cannot be predicted categorically. Furthermore, the third case is inappropriate. It assumes that when no prediction can be made, every possible value of a measurement is equally likely, or

$$p(s^+ \mid D^+) = p(s^- \mid D^+) = 0.5$$

This seems to be a rather poor assumption. Most symptoms are unlikely, so we should not predict that on average half of the symptoms are positive. When D contains irrelevant knowledge, it should not affect our probability for s, so a better assumption is

$$p(s^+ \mid D^+) \;=\; p(s^+)$$

In other words, if symptom s is improbable to begin with, then it should still be improbable even after we know the irrelevant information in D^+. Conversely, if symptom s is very likely to begin with, then it should be still be likely after we know the irrelevant information in D^+. Thus, the correct equation for GDE, applied to our domain, should be

$$p(D^+ \mid s^+) \;=\; \begin{cases} p(D^+)/p(s^+) & \text{if } D^+ \text{ predicts } s^+ \\ p(D^+) & \text{otherwise} \end{cases}$$

Unfortunately, even when corrected, this equation has limited usefulness. In GDE's domain of circuit analysis, predictions are categorical. In our domain, though, predictions are probabilistic. It is never clear when a candidate predicts the presence of a symptom, so the above equation cannot be applied. On the other hand, the formulas developed in this chapter are applicable when predictions are both categorical and probabilistic. The categorical case is achieved simply by having link probabilities c_{ds} that are either 0 or 1. Moreover, the deKleer and Williams's formula only handles a single positive symptom. As we have seen, the probability of several positive symptoms is difficult to compute and requires exponential time.

Peng and Reggia [55] have considered the case where disorders cause symptoms with arbitrary causal strengths c_{ds}. However, their result simplifies the computation by introducing two assumptions. First, they construe candidates to be disorder instances. Second, they condition on the event that only the given positive symptoms are actually present. Their result is reproduced here:

$$p(D^+(U_D - D)^- \mid P^+(U_S - P)^-) = \frac{\prod_{s \in P} \left(1 - \prod_{d \in D} q_{ds}\right) \prod_{s \in U_S - P} \left(\prod_{d \in D} q_{ds}\right)}{\prod_{d \in U_D - D} p_d^-}$$

The assumptions made by Peng and Reggia may not be valid in some domains. For example, in medicine, people often have minor illnesses, such as the common cold. But the first assumption requires that the patient not have any diseases, however minor, except those that are included in the candidate. The second assumption is even less reasonable. It requires that we know the presence or absence of every symptom in the knowledge base. This is an infrequent situation in most domains. For example, this assumption forces us to decide, even before a chest X-ray has been performed, whether the result is going to be positive or negative. In most domains that deal with uncertainty, the status of most evidence is unknown.

In our equation for the probability of a candidate (7.26), on the other hand, both of these assumptions are relaxed. To relax the first assumption, we have defined a candidate to be present whenever all of its component disorders are present, regardless of whether other disorders are present or absent; the status of these other disorders is simply unknown. To relax the second assumption, we allow two sets of symptoms, positive and negative,

to be specified. Symptoms in the knowledge base not in either category are assumed to be unknown. This differs from Peng and Reggia's approach where only one set of symptoms, the positive ones, are specified, while all other symptoms are assumed to be negative.

7.8.2 Belief Networks

In addition to candidate generation, another approach to multidisorder diagnosis is based on belief networks [51]. A belief network is a network of nodes, where each node represents some proposition, such as a disorder, symptom, or intermediate state. Each proposition has an attached probability, which is updated locally and propagated to neighboring nodes as new evidence is received.

Belief networks perform a computation that is different from candidate generation. Whereas candidate generation deals with combinations of disorders, belief networks deal mainly with individual disorders. In applications to diagnosis, a belief network can assign a posterior probability to each disorder, given a case with positive and negative symptoms. But these probabilities are for individual disorders; they do not indicate what combinations of disorders (called interpretations in their terminology) are likely. Unfortunately, probabilities of individual disorders do not necessarily translate to plausible candidates in a straightforward way. For instance, if we merely assign "presence" to each disorder with a probability greater than 0.5 and "absence" to each disorder with a probability less than 0.5, the resulting interpretation may be highly unlikely. Or if we take the two or three most probable disorders, the result may also be highly unlikely.

Pearl has recognized this limitation of belief networks, and he has developed a method called belief revision to compute disorder combinations. However, belief revision is limited to finding only the best and second-best interpretations. In order to obtain a more complete list of disorder combinations, a search process such as candidate generation or decompositional search presently offers the only alternative.

7.9 Summary and Discussion

This chapter has derived equations for various probabilistic quantities. In summary, the equations and the computational complexity for their relative

probabilities are:

Quantity	Equations	Complexity
Candidate Set	7.25, 7.22, 7.24	$O(2^{\mathcal{P}}\mathcal{NU}_D)$
Candidate	7.26	$O(2^{\mathcal{P}}\mathcal{NU}_D)$
Task	7.31	$O(\mathcal{PNU}_D)$
Single-Fault Assumption	7.32	$O(\mathcal{PNU}_D)$

One of the major problems with set covering approaches is that they are essentially categorical. Perhaps, though, the results derived here may be used to change the set covering approach to a probabilistic one. For instance, evaluation of decomposition probabilities might be used to guide search. Unfortunately, the probability of a problem decomposition with more than one cluster is difficult to quantify, since the assumption of task independence is not valid.

However, the task independence assumption might closely approximate the probability of a problem decomposition. For many purposes, such as guiding search, approximate values would be sufficient. The result would be

$$p(\bigwedge_{C\in\mathcal{C}} (\text{Diff}(C) \xrightarrow{\vee} C) \mid P^+N^-) \approx \prod_{C\in\mathcal{C}} p(\text{Diff}(C) \xrightarrow{\vee} C \mid C^+N^-)$$

This would then provide a polynomial time method of assessing the probability of a problem decomposition. This assumption may be reasonable to make since the coherency criterion helps partition symptoms into separate causal groups. These operations help reduce the causal and probabilistic interactions between tasks, thereby supporting a task independence assumption.

In particular, combining probabilities with decompositional search should be computationally advantageous, because a problem decomposition entails a set of candidates. Knowing the probability of a decomposition would then allow one to explore large portions of the candidate space at a higher level of abstraction. A probabilistic decompositional search algorithm might then increase the efficiency gains achieved by the categorical algorithm.

Chapter 8

Conclusion

Everything's got a moral, if you can only find it.

— Lewis Carroll, *Alice's Adventures in Wonderland* (1865)

8.1 Summary

In this thesis, we have developed a new approach to diagnosis, based on finding plausible decompositions of a problem into subproblems. We have designed an algorithm called decompositional search; tested the efficiency of decompositional search against the non-decompositional candidate generation approach; and analyzed the features of the domain that account for the efficiency of decompositional search. In addition, we have developed a preliminary theory for including probabilistic considerations in decompositional search. Therefore, our major results can be summarized as follows:

Theoretical results

We developed a formal representation for problem decompositions as a set of clusters with associated differential diagnoses. Problem decompositions define a set of commonality and disjointness constraints that define the differential diagnosis for each cluster. Differentials can be formulated by using subsets of clusters as justifications for those clusters. We developed a plausibility criterion called coherency based on the satisfiability of the constraints placed by a decomposition.

This new representation for diagnosis is closely related to minimal candidates. A problem decomposition entails a candidate set, represented implicitly by the Cartesian product of its differential diagnoses. This candidate set contains all minimal candidates that could potentially be generated by the decomposition, but may also contain some nonminimal candidates required for the Cartesian product representation.

Coherent problem decompositions can be generated by a search process. This search process executes three steps: symptom assignment, ambiguation, and disambiguation. Symptoms may be assigned using covering, restricting, adjoining, and operators. The ambiguation and disambiguation steps reassign symptoms that are unnecessarily restricting. They allow the search process to ignore overly general symptoms and therefore focus on the assignments of symptoms that do place constraints.

Finally we demonstrated that decompositional search is theoretically incomplete with respect to generating the entire set of maximally ambiguous and coherent problem decompositions. However, in practice, decompositional search is robust and complete with respect to generating all minimal candidates for a given problem.

Experimental results

In general, decompositional search was more efficient than candidate generation by a power of 4 to 5 for space complexity and 3 to 5 for time complexity. These results held regardless of whether experimental cases were produced by one or two target disorders.

We demonstrated that both case presentation and case ordering significantly influence the complexity of problem solving, even for the same target disorder. A single symptom can greatly influence the complexity of a problem. Ordering symptoms according to their specificity appears to be a good heuristic strategy for reducing computational complexity.

Decompositional search was shown to be complete in practice with respect to generating minimal candidates. It was also fairly robust, giving the same set of problem decompositions regardless of symptom ordering. The algorithm was largely, but not completely, sound and irredundant with respect to generating minimal candidates. A few cases resulted in a set of problem decompositions that either entailed a relatively large proportion of nonminimal candidates or a large proportion of duplicated candidates among decompositions. We constructed simple models to explain these anomalous behaviors.

Analytical results

We conducted a worst-case theoretical analysis that suggests that candidate generation should have complexity $O(c^{\log_r c})$ and that decompositional search should have complexity $O(c^{\log_r \log_r c})$, where c is the number of possible causes for each symptom and r is a factor related to the correlation between the possible causes of each symptom. These formulas as symptom correlation decreases and as the knowledge base scales up in size.

Turning from a theoretical to an empirical analysis, we developed a theory for diagnostic complexity based on the combinatorics of partial explanations. We generalized this theory by defining the explanatory power for a given set

of disorders. Disorders with lower explanatory power combine to cause most of the complexity of diagnosis. Moreover, the distribution of explanatory power reflects underlying domain structure that can be exploited by decompositional search.

We tested this theory by manipulating the explanatory power of a subdomain, creating both a trimmed and a redistributed subdomain. The trimmed subdomain had disorders of least explanatory power removed, while the redistributed subdomain had the possible causes for each symptom randomized. Experimental runs on these modified subdomains suggest that, indeed, much of the efficiency of decompositional search derives from the exploitation of domain structure.

Finally, we considered one case in detail, by decomposing the entire prerenal azotemia subdomain. The resulting decompositions had clusters that often corresponded to meaningful groups in clinical medicine. This experiment suggests that decompositional structure is pervasive, existing not only at high levels of abstraction but also at the low level of single-disorder subdomains.

Probabilistic results

We modeled domain probabilistic knowledge by incorporating prior probabilities and link probabilities into a knowledge base. The noisy-or assumptions provide a semantics for such a probabilistic knowledge base. We defined the notion of causal probability and contrasted it with conditional probability. We also defined a probability of non-causation.

Using these probabilistic quantities, we derived a formula for the probability of a case that contains both positive and negative symptoms. We also derived a formula for the conditional probability of a candidate set, given a case. Based on this result, we derived a formula for the conditional probability of a candidate, given a case. This equation represents a probabilistic interpretation of a candidate that improves upon previous work, which makes several unwarranted assumptions. All of these quantities can be computed in time that is exponential in the number of positive symptoms in a case.

We derived a formula for the conditional probability of a task, given a case. This probability could be combined with a task independence assumption to give a polynomial-time algorithm for computing the probability of a decomposition. Based on this result, we also computed the conditional probability that the single-fault assumption is true, given a case. This probability

can also be computed in polynomial time.

8.2 Features of Decompositional Search

In this section, we present the major concepts underlying the decompositional search approach. These concepts complement the particular results listed above, because they provide broader principles of computation applicable to computer science in general.

8.2.1 Implicit and Explicit Representation

The first concept is that of implicit and explicit representation. A problem decomposition represents a set of candidates implicitly as a Cartesian product of its differential diagnoses. In contrast, candidate generation represents candidates explicitly. The implicit representation is more compact, requiring only space proportional to the sum of its differential sizes. However, the equivalent representation in terms of explicit candidates requires space proportional to the product of the differential sizes.

The implicit representation can be viewed as a *generator*, representing a set of candidates without necessarily computing them, except as needed. The idea of generators occurs frequently in computer science; for instance, in formal language theory, a grammar is an implicit generator of explicit strings in a language [30]. Generators can offer potentially large savings in efficiency if they can be manipulated and transformed directly, without having to convert them to the explicit form and back. In this thesis, we have shown that problem decomposition representations can indeed be transformed directly, without having to convert them to explicit sets of candidates.

The implicit and explicit representations are analogous to conjunctive and disjunctive normal forms in logic, respectively. Conjunctive normal form is a conjunction of disjunctive statements, while disjunctive normal form is a disjunction of conjunctive statements. Candidates have a conjunctive meaning, hypothesizing that every disorder in a particular set is present. On the other hand, differential diagnoses have a disjunctive meaning, hypothesizing that one or more disorders in a particular set is present. Furthermore, the collection of differential diagnoses in a problem decomposition have a conjunctive meaning, hypothesizing the presence of at least one disorder in each differential. In chapter 1, we showed how a set of differential diagnoses can

represent a set of candidates. If we use the conjunctive and disjunctive meanings above, we obtain the analogous result for disjunctive and conjunctive normal forms:

$$\begin{aligned}
&\{d_1 \vee d_2\} \wedge \{d_3 \vee d_4 \vee d_5\} \wedge \{d_6 \vee d_7 \vee d_8\} = \\
&\quad [d_1 \wedge d_3 \wedge d_6] \vee [d_1 \wedge d_3 \wedge d_7] \vee [d_1 \wedge d_3 \wedge d_8] \vee \\
&\quad [d_1 \wedge d_4 \wedge d_6] \vee [d_1 \wedge d_4 \wedge d_7] \vee [d_1 \wedge d_4 \wedge d_8] \vee \\
&\quad [d_1 \wedge d_5 \wedge d_6] \vee [d_1 \wedge d_5 \wedge d_7] \vee [d_1 \wedge d_5 \wedge d_8] \vee \\
&\quad [d_2 \wedge d_3 \wedge d_6] \vee [d_2 \wedge d_3 \wedge d_7] \vee [d_2 \wedge d_3 \wedge d_8] \vee \\
&\quad [d_2 \wedge d_4 \wedge d_6] \vee [d_2 \wedge d_4 \wedge d_7] \vee [d_2 \wedge d_4 \wedge d_8] \vee \\
&\quad [d_2 \wedge d_5 \wedge d_6] \vee [d_2 \wedge d_5 \wedge d_7] \vee [d_2 \wedge d_5 \wedge d_8]
\end{aligned}$$

This type of efficiency obtained by decompositional search is similar to Minsky's observation that dividing a problem into subproblems reduces the total complexity from the product of the individual search spaces to their sum [40]. In diagnosis, the idea of implicit representation was noted by Reggia [58], who observed that candidates could potentially be factored into "generators". However, he did not present an algorithm to perform the factorization. Implicit representations can also be applied to domains besides diagnosis. For instance, Hubbe and Freuder [31] have applied some of the ideas in this thesis to develop a cross product representation for constraint satisfaction problems. Their results also show that cross product representations substantially improve the performance of standard constraint satisfaction algorithms.

8.2.2 Convex Approximation

Although problem decompositions are closely related to candidate generation, their representation of minimal candidates is only approximate. In this thesis, we have established that a given problem decomposition is complete but not sound with respect to generating minimal candidates. In other words, the candidate set for a problem decomposition contains all minimal candidates that satisfy the commonality and disjointness assumptions for the decomposition, but it also contains some nonminimal candidates as well. One reason for unsoundness is that nonminimal candidates are sometimes needed to achieve a compact Cartesian product representation. Decompositional search is therefore an approximate algorithm, and this also accounts for some of its efficiency.

To better understand the nature of this approximation, let us consider an analogy. Suppose we wish to represent compactly all valid combinations for a two-cylinder combination lock, where each number to be dialed ranges between 0 and 9. A first approximation to achieving an implicit representation is therefore

$$\{0,1,2,3,4,5,6,7,8,9\} \quad \times \quad \{0,1,2,3,4,5,6,7,8,9\}$$

However, because of mechanical constraints, no combination may be contain digits. Therefore, the implicit representation above is not sound with respect to generating legal combinations. It generates some illegal combinations, such as [0,0] and [9,9]. However, if we were to remove these illegal combinations, a compact implicit representation would no longer be possible. The best possible representation would be

$$\begin{array}{rccc} & \{0\} & \times & \{1,2,3,4,5,6,7,8,9\} \\ \cup & \{1\} & \times & \{0,2,3,4,5,6,7,8,9\} \\ & \vdots & & \vdots \\ \cup & \{9\} & \times & \{0,1,2,3,4,5,6,7,8\} \end{array}$$

Thus, there is a tradeoff not only between the soundness and completeness of a decomposition but also between soundness and compactness. Problem decompositions can represent sets of minimal candidates compactly, but only at the cost of generating some nonminimal candidates as well. As we have seen experimentally, this cost is actually small in practice. The soundness of decompositional search is generally very close to perfect, meaning that almost all candidates entailed by a problem decomposition are minimal. In any case, there is little harm in generating a nonminimal candidate. In diagnostic reasoning, inferences are only intended to be plausible and not logically sound. In decompositional search, we opt for generating "nonplausible" candidates when necessary in order to achieve a compact representation.

8.2.3 Causal Structure

A problem decomposition not only produces differential diagnoses, but also links each of them with a symptom cluster. This mapping between differential diagnoses and symptom clusters constitutes a causal structure. Causal structures do not appear in the candidate generation approach. In that algorithm, candidates are generated without specifying which disorders cause which symptoms.

Causal structures are useful computationally because they group disorders according to their *causal equivalence*, that is, their ability to explain the given symptoms. An important feature of causal equivalence is that it is problem-specific. Two disorders may have largely different sets of possible effects, but in a particular problem, they may not be distinguishable. In the extreme case, when only one symptom is known, all of its possible causes are causally equivalent.

Causally equivalent disorders behave similarly in the candidate generation search tree, so many expansion and pruning steps in candidate generation are redundant. Indeed, this redundancy provides evidence of underlying domain structure. By representing a group of causally equivalent candidates, a problem decomposition enables search decisions to be made about an entire set of candidates. Thus, decompositional search avoids much of the redundant reasoning in candidate generation.

Aside from making decompositional search efficient, the causal structures inherent in problem decompositions may be useful end products in themselves. A user may not necessarily want to know the answer to a diagnostic problem, but might want to fit it into an understandable paradigm. In fact, the important question in diagnosis is often not to determine what disorder is present, but to determine what to do next. In many diagnostic problems, it may be premature to expect a definitive answer to the problem. Rather, the most useful analysis would be to help determine the next set of tests to perform.

The causal structures embodied in decompositional search may help provide this sort of analysis. Because it formulates a set of distinct subproblems, a decompositional search system can help a user focus on each subproblem individually. The symptom clusters can help the user focus on the subproblems that are most critical or of particular interest. Moreover, the differential diagnoses contain those disorders that compete directly against one another. The appropriate tests, then, would be those that discriminate among the disorders in a differential diagnosis.

8.2.4 Symptom-Based Diagnosis

Decompositional search differs from most diagnostic approaches not only in providing causal structure, but also in that it manipulates symptoms rather than disorders. In candidate generation, nodes in the search space are combinations of disorders, and these nodes are expanded by adding disorders.

On the other hand, in decompositional search, nodes in the search space are decompositions of symptoms, and these nodes are expanded by adding and rearranging symptoms.

Symptom-based diagnosis offers several advantages over disorder-based diagnosis. Diagnostic problems often have fewer symptoms than possible causes. This asymmetry appears to be a fundamental consequence of the nature of the diagnostic task. Since each symptom in a domain is linked to a set of possible causes and since each symptom could be caused separately, the total number of disorders under consideration generally exceeds the total number of symptoms under consideration. Consequently, the space of symptom combinations is potentially much smaller than the space of disorder combinations.

Nevertheless, in certain situations, the number of symptoms may exceed the number of disorders under consideration. For instance, a diagnostic case may contain numerous symptoms all linked to the same small set of disorders. But in this situation, the symptoms would all be ambiguous with respect to each other, and decompositional search would collapse the space of possible symptom combinations into only a few problem decompositions. The process of ambiguation exploits the fact that it is not the total number of symptoms that determines the size of a symptom-based search space, but rather the number of critical or "key" symptoms. In our theory of decompositional search, critical symptoms are identified by the notions of covering and restricting. Restricting symptoms define the common causes of a cluster and thereby constitute the critical symptoms that determine the plausible solutions to a diagnostic problem. Covering symptoms, on the other hand, are not critical since they often can be assigned to more than one cluster without changing the common cause sets of the decomposition.

8.2.5 Static and Dynamic Problem Decomposition

The final feature of decompositional search we will discuss is the difference between static and dynamic problem decomposition. Decompositional search is certainly not the first system to use abstraction to increase efficiency [60]. However, it differs from other diagnostic programs in the way it derives abstractions. In most knowledge-based algorithms, abstractions are derived by acquiring them from experts in the domain. These abstractions are then encoded explicitly into the knowledge base [6]. We call this type of abstraction *static*, because the abstractions are fixed in the knowledge base and used

repeatedly for each new problem.

On the other hand, decompositional search does not depend on predefined abstractions in the knowledge base. Rather than relying on static abstractions, decompositional search uses *dynamic abstraction*, by formulating an abstractions as needed for a particular case. Symptom clusters and differential diagnoses are created dynamically. In this type of abstraction, the formulation of abstractions is guided by a notion of plausibility, such as coherency. Dynamic abstraction is made possible by the domain structure, so that appropriate abstractions emerge from the collective set of links in the knowledge base.

By using dynamic abstraction, decompositional search can solve problems at an appropriate level of abstraction and can vary the level according to the particular evidence available. When the symptoms are general, the differential diagnoses for each cluster are relatively large. But as symptoms become more specific, the differential diagnoses become smaller. Thus, a problem decomposition is exactly as abstract as the symptoms warrant. Variable levels of abstraction not only help improve the efficiency of problem solving, but they might also help a user understand a problem at an appropriate level of detail. This contrasts with the candidate generation approach, which can only present solutions at the most primitive level of abstraction, the candidate itself.

The technique of dynamic abstraction may also help in areas other than diagnosis, such as knowledge acquisition and machine learning. In order to build a static abstraction, one must expend much effort to acquire knowledge from a domain expert. Moreover, such knowledge acquisition is likely to succeed primarily in domains where diagnostic expertise is highly developed and domain knowledge is structured clearly. But many fields do not have abstractions that are so well established. Even in a field like medicine, where syndromes are touted as useful problem solving tools [10, 15, 18, 43, 78], exact definitions of syndromes are difficult to pin down. Textbooks of medicine describe individual disorders in great detail but describe syndromes in only general terms, if at all. Moreover, the appropriate syndrome and level of detail vary according to the exact problem at hand, so that a static abstraction needs to contain not only the abstractions but also the conditions under which they are likely to be applicable or useful.

Finally, dynamic abstraction and static abstraction are not necessarily incompatible. A knowledge base might contain information about syndromes that could be used as heuristic guides to direct the decompositional search

process. These heuristics could help a diagnostic system focus on problem decompositions that are most likely to produce coherent descendants. But with an underlying dynamic abstraction process, such a system would also have the ability to create appropriate decompositions when necessary, tailored to the particular case at hand.

8.3 Relation to Other Work

We now place the decompositional search approach in context, comparing it with other related work. Comparisons with probabilistic methods, such as belief networks, were presented previously in chapter 7.

8.3.1 Diagnosis from First Principles

Decompositional search is closely related to work on diagnosis using first principles [11]. This field was motivated by the fact that most diagnostic programs, such as flowcharts and rule-based expert systems, had relied on heuristics and problem solving strategies derived from domain experts. Diagnosis from first principles attempts to solve problems directly from a description of the domain, rather than relying on expert-supplied heuristics or strategies. Another motivating factor is that flowcharts and rule-based expert systems handle multiple disorders poorly. This deficiency is in part due to the lack of heuristics for computing multiple disorders. Even if such heuristics were available, they would probably be inadequate for the large search spaces arising from multidisorder diagnosis. There have been two main approaches to diagnosing from first principles: model-based diagnosis [12] and set-covering diagnosis [58]. These two fields differ primarily in the way they model a domain.

Model-based diagnosis relies on two techniques: candidate generation and conflict recognition [14]. Conflict recognition is the process of generating "conflict sets" from a structural or functional model of the system being diagnosed. A conflict set is similar to the set of possible causes for a symptom, except that a conflict set may explain an observation in the context of a previous observation. An example of conflict recognition is provided in figure 8-1. In this example, the circuit should have outputs of F=12 and G=12, but has F=10 and G=12 instead. Since output F is incorrect, one of the components that compute F must be broken, namely, components A_1,

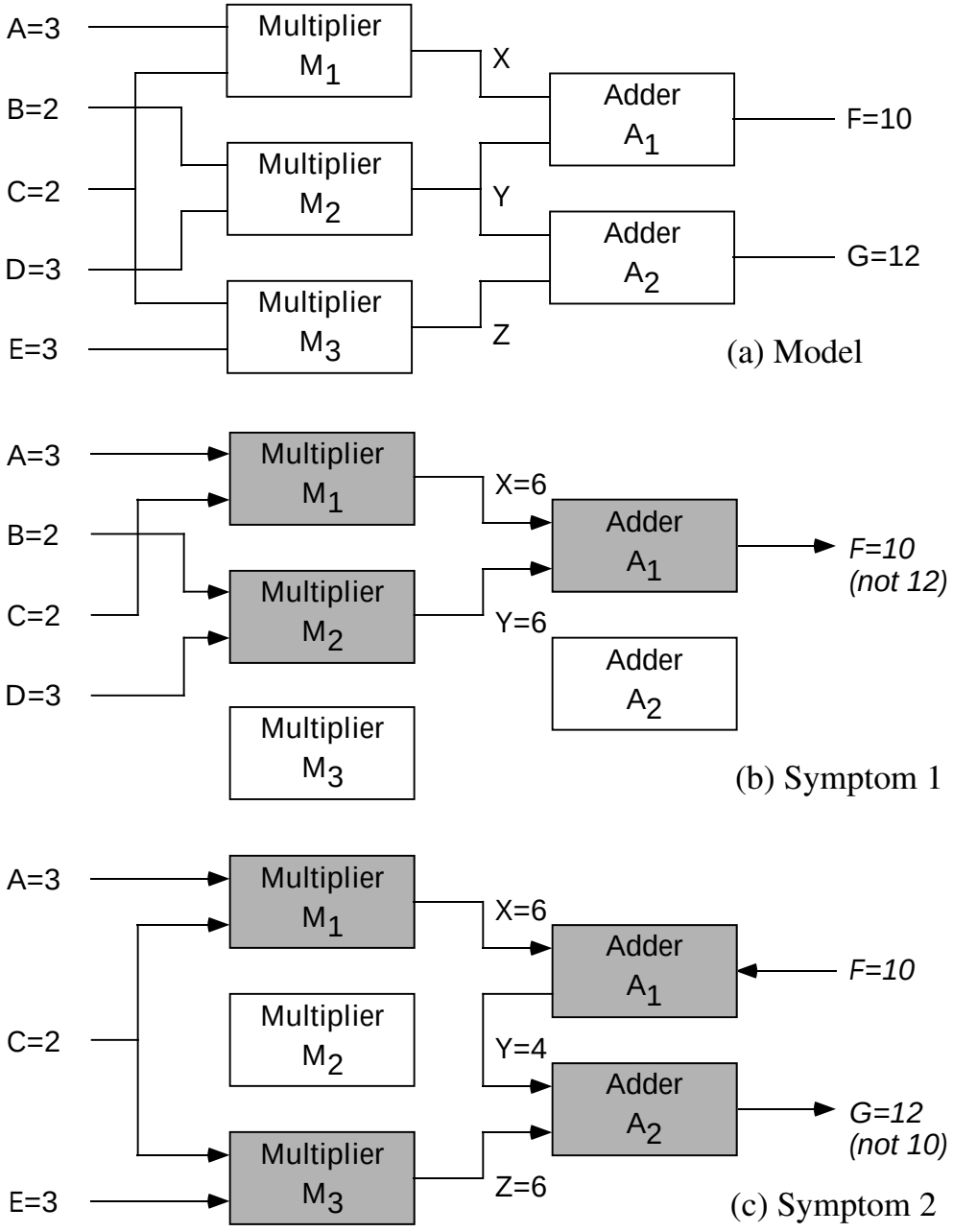

Figure 8-1 Conflict recognition. (a) Model of a circuit. (b) Conflict recognition for the symptom 1, resulting in the conflict set $\langle A_1, M_1, M_2 \rangle$. (c) Conflict recognition for the symptom 2 in the context of symptom 1, resulting in the conflict set $\langle A_1, A_2, M_1, M_3 \rangle$.

M_1, or M_2. These three components constitute the first conflict set. The second conflict set arises because if F is incorrect, G should be also be incorrect, assuming that components A_1, A_2, M_1, and M_3 are working. However, the output at G is correct, so one of these components is broken, thereby yielding a second conflict set. Note that the second conflict set explains the observation that G is correct in the context of the observation that F is incorrect.

Conflict recognition supplies input for candidate generation. The two conflict sets $\langle A_1, M_1, M_2 \rangle$ and $\langle A_1, A_2, M_1, M_3 \rangle$ result in the minimal candidates $[A_1]$, $[M_1]$, $[M_2, A_2]$, and $[M_2, M_3]$. Existing programs for model-based diagnosis process conflict sets using the candidate generation algorithm. Unfortunately, the computational complexity of candidate generation severely limits the applicability of model-based diagnosis. Decompositional search can help expand the usefulness of model-based diagnosis by providing a more efficient alternative to candidate generation. Decompositional search has no bearing on conflict recognition, but computing conflict sets is computationally much easier than generating candidates.

Whereas model-based diagnosis uses a structural or functional model of the domain, set-covering diagnosis uses a diagnostic knowledge base to represent the domain. As discussed previously, a diagnostic knowledge base consists of symptoms, disorders, and the causal relationships between them. This type of knowledge base is especially appropriate for domains where structural or functional models are unavailable or difficult to construct. For instance, we do not yet have the ability to construct a model of the human body with enough detail and accuracy for model-based diagnosis. However, we do have much knowledge about associations between diseases and symptoms. On the other hand, we can easily construct diagrams for circuits, so much work in model-based diagnosis has been applied to circuit troubleshooting. Nevertheless, existing set-covering methods also rely predominantly on the candidate generation algorithm and hence would benefit from the decompositional search algorithm.

8.3.2 Medical Diagnostic Systems

Medicine is a particularly appropriate domain for multidisorder diagnosis because the domain is large, probabilities play an important role, and multiple coexisting diseases are common [7, 47, 63, 73, 74]. Here we focus on the evolution of systems that attempt to solve the problem of multidisorder di-

agnosis. This excludes many early systems, such as MYCIN [65] and PIP [49], that solve problems where only one disorder is assumed to be present.

The problem of multiple disorders was recognized by Gorry in 1968 [21]. Interestingly, Gorry's program used a "pattern-sorting function" that essentially clusters symptoms (or "attributes", as he terms them):

> The program processes the attributes through the *pattern-sorting function*. This function makes decisions about the relevance of attributes to the current diagnostic problem. If, for example, the initial problem definition had included the attribute "sore ankle," the pattern-sorting function might have decided that "sore ankle" and "persistent coughing" were manifestations of different medical problems and should be considered separately. The output of the pattern-sorting function is a set of attributes that it believes should be considered as a group by the program.

This pattern-sorting function essentially found symptom clusters using the commonality constraint. That is, a symptom cluster is valid only if there exists a disorder that can explain all symptoms in that cluster. The program maintained a list of valid symptom clusters and decided on the basis of probability and utility which cluster to pursue. The program then selected a test to discriminate among the disorders that could explain the cluster. Thus, Gorry's program can be thought of as solving the multidisorder problem sequentially, by solving one cluster at a time, with the capability of switching between clusters.

Another system that used a sequential approach was the INTERNIST program [39]. This program, now available as QMR [38], meaning Quick Medical Reference, contains diagnostic knowledge in the form of disease profiles, which is essentially equivalent to the bipartite knowledge base we have been using in this thesis. Each disease-symptom pair has a link probability indicating how likely the disease is to cause the symptom, given that the disease is present. This probability appears as a frequency value between 1 and 5. Each disease-symptom pair also has an evoking strength attached, indicating the importance of considering a disease given a symptom. INTERNIST uses these frequency values and evoking strengths to identify the highest scoring disease. It then builds a differential diagnosis around this disease, by finding those diseases that are competing explanations for the same symptoms. It scores the diseases in the differential diagnosis and asks the user a question to discriminate among them. Once the top-ranked disease scores high enough

relative to the rest of the differential, INTERNIST concludes that differential and forms another one, based on the unexplained symptoms. Thus, INTERNIST uses a score-based "partitioning heuristic" to decompose a problem sequentially.

The sequential strategy in INTERNIST addresses the problem of multiple disorders but often gives poor results. One problem with sequential differential formulation is that it is not robust. Each differential must be finished before the next differential is considered. If INTERNIST formulates the first differential incorrectly, it will misinterpret the rest of the problem. Another issue in sequential differential formulation is how to assign symptoms to each disorder. After concluding a disease, INTERNIST finds all symptoms that it explains and removes them from further consideration. But this may remove too many symptoms, since symptoms potentially caused by one differential may actually be caused by another one. Thus, INTERNIST suffers because diagnostic problems are only nearly decomposable and not completely so. The sequential technique works well on problems that are completely decomposable. But near decomposability complicates matters because of interactions between subproblems and the consequent need to explore alternative decompositions.

To remedy these deficiencies in INTERNIST, the CADUCEUS program was developed by Pople [57]. He emphasized the importance of "task formulation" in multidisorder diagnosis, and thereby reiterated the need to identify the correct decomposition for a given problem. CADUCEUS finds tasks by using a combined hierarchical-causal network. The hierarchies contain preformulated differential diagnoses that are triggered by causal links. Causal links are orthogonal to hierarchical links and connect physiologically related disease categories in different hierarchies. CADUCEUS diagnoses a problem by first triggering multiple differential diagnoses in the hierarchies and then following various subsumption and causal relationships to derive a globally consistent diagnostic picture.

CADUCEUS addresses the problems of sequential differential formulation and lack of causal integration found in INTERNIST. However, it suffers from other problems. First, it relies on hierarchically structured differential diagnoses. Unfortunately, differentials cannot be organized so cleanly. To remedy this, differential diagnoses are triggered using a existential relationship, meaning that some diseases under a node may not explain a given symptom. But this substantially weakens the inference possible from that node. Second, CADUCEUS requires a substantial amount of physiological information

to create a hierarchical-causal network. Many areas of medicine lack such a detailed causal understanding of disease processes.

A more rigorous approach to multiple disease interactions appears in the ABEL program by Patil [45, 48]. It structures physiological knowledge about acids, bases, and electrolytes into several levels of abstraction. It can therefore reason about additive and antagonistic interactions between diseases. But at the same time, it requires domains where such detailed information is available and where close interactions between disorders occur. This makes the approach well suited to narrow domains with well understood physiology, like acid-base and electrolyte therapy, but ill suited to broad domains such as internal medicine, at least until a detailed pathophysiological model of the human body becomes available.

8.3.3 Conceptual Clustering

Decompositional search shares a number of similarities with a technique in machine learning called conceptual clustering [17, 37]. In conceptual clustering, one is given a set of objects, and the task is to assign them to clusters such that the set of clusters scores well on a clustering quality function. The clustering quality function is essentially a plausibility criterion, and can include such measures as the simplicity, commonality, disjointness, and discrimination of a cluster.

Although the task of conceptual clustering appears superficially identical to that of decompositional search, it is fundamentally different in detail. In conceptual clustering, the objects are described by pairs of variables and their associated values. Clusters of objects are therefore described by pairs of variables and the range of values they can take. For example, consider the two objects:

$$\begin{aligned} &\text{Object 1:} \quad (\text{Color} = \text{blue}) \wedge (\text{Size} = \text{large}) \wedge (\text{Shape} = \text{round}) \\ &\text{Object 2:} \quad (\text{Color} = \text{red}) \wedge (\text{Size} = \text{medium}) \wedge (\text{Shape} = \text{round}) \end{aligned}$$

A cluster containing these two objects might have the following description:

$$[\text{Color} = \text{blue} \vee \text{red}] \wedge [\text{Size} \geq \text{medium}] \wedge [\text{Shape} = \text{round}]$$

Thus, the goal of conceptual clustering is not only to make plausible clusters but also to describe them in a way that makes sense. The task of finding sensible descriptions for a set of objects is a type of inductive inference called concept learning, a central topic in machine learning [41, 79]

In contrast, whereas objects in conceptual clustering are not causally related, decompositional search is closely tied to causal relationships. Decompositional search relies upon causal relationships between symptoms and disorders to derive its commonality and disjointness criteria. The plausibility of a decomposition is determined by trying to satisfy these criteria by formulating differential diagnoses. Thus, a cluster is essentially "described" by its differential, and differential formulation can be seen as an analogue of concept learning for diagnosis.

Decompositional search also differs from conceptual clustering in that it can assign symptoms to more than one cluster. In most work on conceptual clustering, objects can belong to only one cluster. Decompositional search assigns symptoms to multiple clusters when they cover more than one cluster. Therefore, decompositional search distinguishes between "critical" symptom assignments that define a cluster and "noncritical" symptom assignments that place no constraints. This distinction is missing in conceptual clustering, where all objects in a cluster are treated equally.

8.3.4 Problem Reduction Techniques

Decompositional search shares some similarity with problem reduction methods in artificial intelligence. Problem reduction takes a problem and decomposes it into smaller subproblems. Problem reduction originated with Gelernter's geometry theorem-proving machine in 1959 [20], which was the first program that could handle conjunctive subgoals [44, p. 138]. The technique was also used in Slagle's symbolic integration program SAINT [69].

Problem reduction can best be described using AND/OR trees [69]. The nodes in an AND/OR tree alternate at each level between decomposing a problem into subproblems (AND nodes) and providing alternative solutions to those subproblems (OR nodes). An example of an AND/OR tree is provided in figure 8-2. In this travel problem, the goal is to travel from Cambridge, Massachusetts to Palo Alto, California. This goal can be decomposed into three subgoals: getting from Cambridge to the airport in Boston, taking a flight to the airport in San Jose, and getting to Palo Alto. This decomposition constitutes an AND node because all three subgoals must be solved. Each subgoal may then be solved in several ways. For example, getting to the Boston airport can be accomplished by driving, riding the subway, or taking a taxi. Solving a subgoal constitutes an OR node because only one solution is necessary. This alternating process of decomposing and solving subgoals

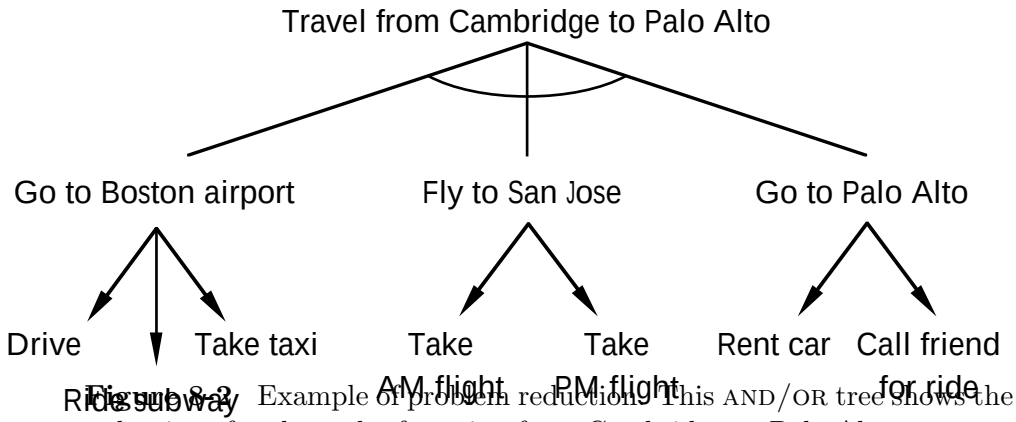

Figure 8.2: Example of problem reductions. This AND/OR tree shows the possible reductions for the task of moving from Cambridge to Palo Alto.

can continue recursively.

Since Gelernter's and Slagle's work, problem reduction techniques have been used in various applications, including theorem proving, automated planning, and expert systems. Nevertheless, although problem reduction is a technique for decomposing problems, it still does not explore the space of alternative decompositions. Problem reduction offers only one way to decompose a problem. There is only one AND node available to decompose a given subgoal. The OR nodes offer a choice, but they present only alternative solutions for a subgoal, not alternative decompositions.

On the other hand, decompositional search explores alternative decompositions of a problem. Decompositions must be synthesized by clustering symptoms and cannot be chosen from a predefined selection of possibilities. Furthermore, problem decompositions are not complete problem reductions, since a problem reduction may have an arbitrary number of subgoal levels. A problem decomposition, however, is only a single-level problem reduction. It is like an AND/OR tree with only one AND node and one level of OR nodes. This limitation arises because of the nature of diagnosis, where each symptom is fully explained by a single disorder. In problem reduction, though, subgoals can be solved by a conjunction of subgoals, hence its recursive form.

8.4 Further Work

Decompositional search constitutes a new approach to diagnostic problem solving. Because the work here has been largely exploratory, further work would help develop the ideas and techniques in this thesis.

One area of further research is to adapt heuristic knowledge to the decom-

positional search algorithm. A major theme of work in artificial intelligence is the power of domain knowledge [36, 50]. Decompositional search exploits domain structure that is inherent in the domain. However, it makes no use of domain knowledge about structure per se. This type of heuristic knowledge might be called structural heuristics, and could guide plausible structuring of evidence or hypotheses. Structural heuristics, such as expert knowledge about common syndromes, could be encoded statically in a knowledge base and could then guide a decompositional search algorithm in assigning symptoms to clusters.

In addition to domain-specific knowledge about structure, there is often case-specific information that can help to cluster symptoms. For instance, some symptoms may occur at the same time, suggesting a common cause. Thus, temporal clustering information might be used to group symptoms together. Similarly, spatial relationships might be useful. For instance, two symptoms that are anatomically close may suggest a common cause. This type of case-specific information represents structure often available in a diagnostic problem, but not represented in a set or sequence of symptoms.

Another area of research is to explore different search strategies. We have used breadth-first search to compare and analyze the space and time complexity of two algorithms. However, given an appropriate measure of plausibility, decompositions might differ greatly in their level of plausibility. Thus, for actual diagnostic applications, other search strategies would be more appropriate than breadth-first search. For example, best-first search, especially in conjunction with a probabilistic evaluation function, could provide a more efficient algorithm.

Many compromises between best-first and breadth-first search are also possible. A beam-first search would keep a fixed number of the best intermediate solutions. The efficiency of the search process could then be controlled by the width of the beam. Of course, plausibility is not the only criterion for good diagnosis. For instance, it is important in medicine to rule out diseases that are rare but have a high degree of morbidity or mortality. Therefore, notions of decision making, such as utility, might also be used to guide search.

The underlying diagnostic knowledge base provides another opportunity for further work. The decompositional search algorithm in this thesis applies to bipartite diagnostic knowledge bases that contain only symptoms and disorders. However, domain knowledge can often be represented as a network [46, 57, 77]. For instance, a medical knowledge base might contain intermediate concepts between symptoms and disorders to represent patho-

physiological states. Further research might extend problem decompositions to represent possible decompositions of a network.

Another topic for further research deals with test ordering or evidence gathering. Diagnosis is often conducted in a hypothetico-deductive paradigm, in a cycle of evidence gathering and hypothesis formation [16]. In this paradigm, an initial set of evidence would yield a set of problem decompositions. These decompositions might then suggest additional tests to order. The decompositional search approach might lead to novel test ordering strategies, since a problem decomposition divides problems into subproblems. These subproblems provide an extra level of detail with which to focus evidence gathering.

Finally, the techniques presented here might be extended to other problem tasks. Many tasks in planning [33, 76] and problem solving [71, 72] seek solutions with multiple components. As we have mentioned, researchers in machine learning have investigated clustering representations, and the ideas in this thesis have found application in solving constraint-satisfaction problems [31]. The ubiquity of decompositional techniques suggests both that structuring problems is an important task and that decomposition provides a powerful tool for organizing and abstracting data. This thesis suggests how structure in diagnostic problems may be discovered and exploited by decompositional search and how problem decompositions can facilitate the solution and understanding of ill-structured diagnostic problems.

Appendix A

Implementation of Decompositional Search Algorithm

The following is the SYNOPSIS implementation of the decompositional search algorithm written in ANSI Common Lisp [70]. The implementation includes procedural details that help make the algorithm run efficiently. We present the program in "bottom-up" order, starting with the primitive implementation and working up to the higher-order functions.

A.1 Sets

SYNOPSIS makes extensive use of sets and set operations. We implement sets as integers that represent bit vectors. These functions are written as macros to avoid the overhead of an extra function call. The following functions implement the standard operations for sets:

```
(defmacro make-empty-set () `0)

(defmacro make-singleton-set (index) `(ash 1 ,index))

(defmacro make-full-set (size) `(1- (ash 1 ,size)))

(defmacro empty-set? (set) `(zerop ,set))

(defmacro set-equal? (set1 set2) `(= ,set1 ,set2))

(defmacro superset? (set1 set2) `(empty-set? (logandc1 ,set1 ,set2)))

(defmacro subset? (set1 set2) `(empty-set? (logandc2 ,set1 ,set2)))

(defmacro set-member? (index set) `(logbitp ,index ,set))

(defmacro set-insert (singleton set) `(logior ,singleton ,set))

(defmacro set-remove (singleton set) `(logandc2 ,set ,singleton))

(defmacro cardinality (set) `(logcount ,set))

(defmacro intersect (set1 set2) `(logand ,set1 ,set2))

(defmacro intersect-sets (sets) `(apply #'logand ,sets))

(defmacro set-union (set1 set2) `(logior ,set1 ,set2))
```

```
(defmacro set-union-sets (sets) `(apply #'logior ,sets))

(defmacro difference (set1 set2) `(logandc2 ,set1 ,set2))
```

The following operations are used to sort sets in lexicographic order. These functions are useful in sorting decompositions.

```
(defmacro set>? (set1 set2) `(> ,set1 ,set2))

(defmacro set>=? (set1 set2) `(> ,set1 ,set2))

(defmacro set<? (set1 set2) `(< ,set1 ,set2))

(defmacro set<=? (set1 set2) `(<= ,set1 ,set2))
```

Sometimes, we need to convert a set from a bit vector representation to a list representation. The following function helps to perform the conversion.

```
(defmethod indices-of (set)
  (loop for bit-vector = set
        then (difference bit-vector (make-singleton-set index))
        until (zerop bit-vector)
        for index = (1- (integer-length bit-vector))
        collect index))
```

In decompositional search, a frequent operation is to find duplicates among a collection of sets. The following procedure performs this function using operations on bit vectors:

```
(defmethod duplicates (sets)
  (loop with gtr-than-one = (make-empty-set)
        with gtr-than-zero = (make-empty-set)
        for set in sets
        do (setf gtr-than-one
                 (logior gtr-than-one (logand gtr-than-zero set)))
           (setf gtr-than-zero (logior gtr-than-zero set))
        finally (return gtr-than-one)))
```

A.2 Primitive Classes

We now define the classes and operations dealing with the primitive elements of a diagnostic knowledge base, namely, its symptoms and disorders. Primitive classes have a name and two indices, a universal index and a case index. The universal index is a permanent, unique number assigned to each symptom and to each disorder, so that no two symptoms have the same index and no two disorders have the same index. The case index is a temporary number assigned to a symptom or disorder as it becomes relevant to a particular case. The case indices are therefore smaller than the universal indices, meaning that set representations using case indices are also much smaller. When a symptom or disorder acquires a case index, the singleton set containing only that element is also stored to increase efficiency.

```
(defconstant start-index 0)
(defconstant not-indexed (1- start-index))

(defclass primitive ()
  ((name :reader name :initarg :name)
   (univ-index :reader univ-index :initarg :univ-index)
   (case-index :accessor case-index :initform not-indexed)
   (singleton :accessor singleton :initform (make-empty-set))))

(defmethod indexed? ((p primitive))
  (/= (case-index p) not-indexed))
```

Given the definition of a primitive class, we can now apply the set definitions to them. Since primitives have singletons attached to them, we can simply use the stored value, instead of having to compute it each time.

```
(defmethod insert-elt ((p primitive) set)
  (set-insert (singleton p) set))

(defmethod remove-elt ((p primitive) set)
  (set-remove (singleton p) set))

(defmethod make-set (primitives)
  (set-union-sets (map 'list #'singleton primitives)))
```

Each symptom has a set of possible causes, and each disorder has a set of possible effects. Disorders also have a prior probability attached to them. When a symptom or disorder is created, only the universal index can be assigned, since the case index holds only for a particular diagnostic case.

```
(defclass symptom (primitive)
  ((univ-causes :reader univ-causes :initform (make-empty-set)
                :initarg :univ-causes)
   (causes :accessor causes :initform (make-empty-set))))

(defmethod make-symptom (name univ-index)
  (make-instance 'symptom :name name :univ-index univ-index))

(defclass disorder (primitive)
  ((univ-effects :reader univ-effects :initform (make-empty-set)
                 :initarg :univ-effects)
   (effects :accessor effects :initform (make-empty-set))
   (prior-prob :reader prior-prob :initarg :prior-prob)))

(defmethod make-disorder (name univ-index prior-prob)
  (make-instance 'disorder :name name :univ-index univ-index
                           :prior-prob prior-prob))
```

To generate the indices needed for symptoms and disorders, we define the following abstraction for index generators:

```
(defmacro make-index-generator () `(1- start-index))

(defmacro reset-index-generator (generator)
  `(setf ,generator (1- start-index)))

(defmacro generate-index (generator) `(incf ,generator))
```

This abstraction is instantiated in the following procedures to generate case indices as needed.

```
(defvar *symptom-case-index* (make-index-generator))
(defvar *disorder-case-index* (make-index-generator))

(defmethod gen-symptom-case-index () (incf *symptom-case-index*))
(defmethod gen-disorder-case-index () (incf *disorder-case-index*))
```

```
(defmethod reset-symptom-case-index ()
  (setf *symptom-case-index* (1- start-index)))
(defmethod reset-disorder-case-index ()
  (setf *disorder-case-index* (1- start-index)))
```

Since sets are represented as bit vectors, which are essentially arrays of indices, we need to be able to lookup an element by its index. We implement this functionality by hash tables, using the following abstraction:

```
(defmacro make-table (test) `(make-hash-table :test ,test))

(defmacro reset-table (table) `(clrhash ,table))

(defmacro lookup-table (key table) `(gethash ,key ,table))

(defmacro enter-table (key val table)
  `(setf (gethash ,key ,table) ,val))
```

With these operations, we can define lookup tables for the universal and case indices of both symptoms and disorders:

```
(defvar *symptom-univ-indextable* (make-table #'eql))
(defvar *disorder-univ-indextable* (make-table #'eql))
(defvar *symptom-case-indextable* (make-table #'eql))
(defvar *disorder-case-indextable* (make-table #'eql))

(defmethod lookup-symptom-univ-index (index)
  (lookup-table index *symptom-univ-indextable*))

(defmethod lookup-disorder-univ-index (index)
  (lookup-table index *disorder-univ-indextable*))

(defmethod lookup-symptom-case-index (index)
  (lookup-table index *symptom-case-indextable*))

(defmethod lookup-disorder-case-index (index)
  (lookup-table index *disorder-case-indextable*))
```

As we mentioned above, we often need to deal with lists of primitives, rather than bit vectors. The following functions convert a set from its usual bit vector representation to a list of primitives:

```
(defmethod univ-symptoms-of (set)
  (map 'list #'lookup-symptom-univ-index (indices-of set)))

(defmethod symptoms-of (set)
  (map 'list #'lookup-symptom-case-index (indices-of set)))

(defmethod univ-disorders-of (set)
  (map 'list #'lookup-disorder-univ-index (indices-of set)))

(defmethod disorders-of (set)
  (map 'list #'lookup-disorder-case-index (indices-of set)))
```

When a symptom is entered in a particular case, it needs to be assigned a case index. The possible causes of the symptom are now relevant to the diagnostic case, and so they also require case indices. The following function logs in the required case indices.

```
(defmethod enter-symptom-in-case ((s symptom))
  (when (not (indexed? s))
    (setf (case-index s) (gen-symptom-case-index))
    (enter-case-index s)
    (add-to-symptom-list s)
    (setf (singleton s) (make-singleton-set (case-index s)))
    (let ((disorders (univ-disorders-of (univ-causes s))))
      (loop for d in disorders
            when (not (indexed? d))
            do (setf (case-index d) (gen-disorder-case-index))
            (enter-case-index d)
            (setf (singleton d) (make-singleton-set (case-index d))))
      (setf (causes s) (make-set disorders)))))

(defmethod enter-univ-index ((s symptom))
  (enter-table (univ-index s) s *symptom-univ-indextable*))

(defmethod enter-univ-index ((d disorder))
  (enter-table (univ-index d) d *disorder-univ-indextable*))

(defmethod enter-case-index ((s symptom))
  (enter-table (case-index s) s *symptom-case-indextable*))

(defmethod enter-case-index ((d disorder))
  (enter-table (case-index d) d *disorder-case-indextable*))
```

```
(defmethod reset-case-indextable (case-indextable)
  (maphash #'(lambda (case-index p)
               (declare (ignore case-index))
               (setf (case-index p) not-indexed))
           case-indextable))

(defmethod reset-case ()
  (reset-case-indextable *symptom-case-indextable*)
  (reset-case-indextable *disorder-case-indextable*)
  (reset-symptom-case-index)
  (reset-disorder-case-index)
  (reset-symptom-list))
```

The symptoms in a case are stored in a list, so that they can be accessed as necessary to expand a decomposition at a given level. The following procedures implement this functionality:

```
(defvar *symptom-list* '())

(defmethod reset-symptom-list () (setf *symptom-list* '()))

(defmethod add-to-symptom-list ((s symptom))
  (setf *symptom-list* (append *symptom-list* (list s))))

(defmethod get-next-symptom (level) (elt *symptom-list* level))
```

A.3 Tasks

A problem decomposition contains a set of clusters, each of which has an associated differential diagnosis. In order to keep each cluster together with its differential, we define an object called a task. A task is essentially a subproblem, and it consists of a cluster and a differential:

```
(defclass task ()
  ((cluster :accessor cluster :initarg :cluster)
   (common-causes :accessor causes :initarg :causes)
   (justification-set :accessor just-set :initform (make-empty-set))
   (exclusion-set :accessor excl-set :initform (make-empty-set))
   (differential :accessor diff :initform (make-empty-set))))
```

```
(defmethod make-task ((s symptom))
  (make-instance 'task :cluster (singleton s) :causes (causes s)))

(defmethod copy-task ((task task))
  (make-instance 'task :cluster (cluster task) :causes (causes task)))
```

The following operations are useful in sorting tasks:

```
(defmethod equal-tasks? ((task1 task) (task2 task))
  (set-equal? (cluster task1) (cluster task2)))

(defmethod task<? ((task1 task) (task2 task))
  (set<? (cluster task1) (cluster task2)))
```

The following operations insert and delete symptoms from clusters. When a symptom is inserted or removed from a cluster, the common causes for the cluster are recomputed, except when the symptom is known to cover the cluster, for which a separate procedure exists.

```
(defmethod causes (cluster)
  (intersect-sets (map 'list #'causes (symptoms-of cluster))))

(defmethod add-symptom! ((s symptom) (task task))
  (setf (cluster task) (insert-elt s (cluster task)))
  (setf (causes task) (intersect (causes s) (causes task)))
  task)

(defmethod add-covering-symptom! ((s symptom) (task task))
  (setf (cluster task) (insert-elt s (cluster task)))
  task)

(defmethod remove-symptom! ((s symptom) (task task))
  (setf (cluster task) (remove-elt s (cluster task)))
  (setf (causes task) (causes (cluster task)))
  task)

(defmethod remove-symptoms! (set (task task))
  (setf (cluster task) (difference (cluster task) set))
  (setf (causes task) (causes (cluster task)))
  task)
```

Two predicates that are used extensively are covering and restricting, which are opposites. Two methods are given for each predicate: one for clusters in existing tasks and one for clusters constructed on the fly.

```
(defmethod covers? ((s symptom) (task task))
  (superset? (causes s) (causes task)))

(defmethod covers? ((s symptom) cluster)
  (superset? (causes s) (causes cluster)))

(defmethod restricts? ((s symptom) (task task))
  (not (superset? (causes s) (causes task))))

(defmethod restricts? ((s symptom) cluster)
  (not (superset? (causes s) (causes cluster))))
```

A.4 Sets of Tasks

Since a decomposition contains multiple clusters, it will be represented by multiple tasks. We represent a set of tasks as a list. Sets of tasks can be altered using the following functions:

```
(defmethod add-task ((newtask task) task-set)
  (cons newtask (map 'list #'copy-task task-set)))

(defmethod substitute-task ((oldtask task) (newtask task) task-set)
  (cons newtask (map 'list #'copy-task (remove oldtask task-set))))
```

In addition, we will find it necessary to put sets of tasks in a particular order, so that they can be compared. The following functions sort and compare sets of tasks.

```
(defmethod sort-task-set (task-set) (sort task-set #'task<?))

(defmethod equal-task-sets? (task-set-1 task-set-2)
  (and (= (length task-set-1) (length task-set-2))
       (every #'equal-tasks? task-set-1 task-set-2)))
```

```
(defmethod task-set<? (task-set-1 task-set-2)
  (cond ((< (length task-set-1) (length task-set-2)) t)
        ((> (length task-set-1) (length task-set-2)) nil)
        (t (loop for task1 in task-set-1
                 for task2 in task-set-2
                 when (task<? task1 task2) do (return t)
                 else when (task<? task2 task1) do (return nil)
                 finally (return nil)))))
```

A.5 Decompositions

We now define the class for problem decompositions. A problem decomposition consists of a set of tasks. We also store the unifying disorders for the decomposition in a slot. The `compute-diff?` slot is a flag that indicates whether the differentials for the decomposition should be computed or recomputed.

```
(defclass decomp ()
  ((tasks :accessor tasks :initarg :tasks)
   (unifying :accessor unifying :initform (make-empty-set))
   (compute-diff? :accessor compute-diff? :initarg :compute-diff?)))

(defmethod make-decomp (tasks &optional compute-diff?)
  (make-instance 'decomp :tasks tasks :compute-diff? compute-diff?))
```

Decompositions can often be generated in duplicate. A standard representation for decompositions helps to identify duplicates for removal. The standard representation lists the tasks in lexicographic order according to their case index:

```
(defmethod standardize ((C decomp))
  (setf (tasks C) (sort-task-set (tasks C)))
  C)
```

The following predicates compare decompositions, to be used for sorting purposes:

```
(defmethod equal-decomps? ((C1 decomp) (C2 decomp))
  (equal-task-sets? (tasks C1) (tasks C2)))

(defmethod decomp<? ((C1 decomp) (C2 decomp))
  (task-set<? (tasks C1) (tasks C2)))
```

A.6 Differential Formulation

Differential formulation computes the differentials for a given decomposition. The differentials are computed by an iterative process that recomputes the justification set, exclusion set, unifying disorders, and differentials. When the justification sets remain unchanged, the process terminates. During any step, if a justification set or differential is found to be null, the process terminates, and the result "incoherent" is thrown to the top-level procedure.

```
(defmethod formulate-differentials ((C decomp))
  (loop for task in (tasks C)
        do (setf (just-set task) (cluster task)
                 (excl-set task) (causes task)
                 (diff task) (causes task))
        finally (return (formulate-differentials-aux C))))

(defmethod formulate-differentials-aux ((C decomp))
  (block exit
    (loop initially
          (setf (unifying C)
                (duplicates (map 'list #'excl-set (tasks C))))
          for task in (tasks C)
          for new-diff = (difference (diff task) (unifying C))
          when (empty-set? new-diff) do (return-from exit :incoherent)
          else do (setf (diff task) new-diff))

    (loop with altered? = nil
          for task in (tasks C)
          for new-just = (compute-justification task C)
          when (empty-set? new-just) do (return-from exit :incoherent)
          else when (not (set-equal? new-just (just-set task)))
          do (setf (just-set task) new-just
                   (excl-set task) (causes (just-set task))
                   altered? t)
          finally (if altered?
                      (return (formulate-differentials-aux C))
                      (return C)))))

(defmethod compute-justification ((task task) (C decomp))
  (make-set (remove-if-not #'(lambda (s) (justification? s task C))
                           (symptoms-of (just-set task)))))
```

```
(defmethod justification? ((s symptom) (task task) (C decomp))
  (every #'(lambda (task-i)
             (not (superset? (causes s) (diff task-i))))
         (remove task (tasks C))))
```

A.7 Ambiguation and Disambiguation

Ambiguation is performed after a new task is created, since that task may be coverable by previously assigned symptoms. The procedure loops through each symptom in each existing cluster to see if it covers the new cluster. If so, it is added to the new cluster.

```
(defmethod ambiguate ((newtask task) (C decomp))
  (loop for task in (remove newtask (tasks C))
        for cluster = (cluster task) do
        (loop for s in (symptoms-of cluster)
              when (covers? s newtask)
              do (add-symptom! s newtask))
        finally (return C)))
```

Disambiguation is performed after ambiguation to remove ambiguous assignments that no longer hold. Ambiguous symptoms are found by finding duplicates within the decomposition. The procedure removes these symptoms, recomputes the common causes for each cluster, and then reassigns the ambiguous symptoms to the clusters that they do cover.

```
(defmethod disambiguate ((C decomp))
  (let ((amb-symptoms (duplicates (map 'list #'cluster (tasks C)))))
    (restore-ambiguous amb-symptoms
                       (remove-ambiguous amb-symptoms C))))

(defmethod remove-ambiguous (amb-symptoms (C decomp))
  (loop for task in (tasks C)
        do (remove-symptoms! amb-symptoms task)
        finally (return C)))

(defmethod restore-ambiguous (amb-symptoms (C decomp))
  (let ((symptoms (symptoms-of amb-symptoms)))
    (loop for s in symptoms
```

```
          for reassign = (find-reassignable s C)
          when (null reassign) do (return :degenerate)
          else do (loop for task in reassign
                        do (add-covering-symptom! s task))
          finally (return C))))

(defmethod find-reassignable ((s symptom) (C decomp))
  (loop for task in (tasks C)
        when (covers? s task)
        collect task))
```

A.8 Symptom Assignment

A.8.1 Covering

The covering operator determines whether some cluster can be covered by the new symptom. If so, it adds the symptom to each cluster that it covers.

```
(defmethod cover-op ((C decomp) (s symptom))
  (when (coverable? C s) (list (cover-aux C s))))

(defmethod coverable? ((C decomp) (s symptom))
  (some #'(lambda (task) (covers? s task)) (tasks C)))

(defmethod cover-aux ((C decomp) (s symptom))
  (loop for task in (tasks C)
        when (covers? s task)
        collect (add-covering-symptom! s (copy-task task)) into tasks
        else collect (copy-task task) into tasks
        finally (return (make-decomp tasks))))
```

A.8.2 Restricting

The restricting operator results in several decompositions, one for each cluster that a symptom restricts.

```
(defmethod restrict-op ((C decomp) (s symptom))
  (loop for task in (tasks C)
        when (restricts? s task)
        collect (restrict-task task C s)))
```

```
(defmethod restrict-task ((task task) (C decomp) (s symptom))
  (let* ((newtask (add-symptom! s (copy-task task))))
    (disambiguate
     (ambiguate
      newtask
      (make-decomp (substitute-task task newtask (tasks C)) t)))))
```

A.8.3 Adjoining

The adjoining operator simply creates a decomposition with the new symptom appended as a cluster by itself.

```
(defmethod adjoin-op ((C decomp) (s symptom))
  (let ((newtask (make-task s)))
    (list
     (disambiguate
      (ambiguate
       newtask
       (make-decomp (add-task newtask (tasks C)) t))))))
```

A.8.4 Admixing

The admixing operator results in several decompositions, one for each previously assigned symptom that can admix with the new symptom. The procedure loops through all admixable symptoms in all previous clusters and collects the resulting decompositions.

```
(defmethod admix-op ((C decomp) (s symptom))
  (loop for task in (tasks C)
        append (admix-task task C s)))

(defmethod admix-task ((task task) (C decomp) (s symptom))
  (loop for admix-s in (admixable-symptoms task s)
        collect (admix-symptom admix-s task C s)))

(defmethod admix-symptom
    ((admix-s symptom) (task task) (C decomp) (s symptom))
  (let ((result-task (add-symptom! admix-s (make-task s)))
        (newtask (remove-symptom! admix-s (copy-task task))))
    (disambiguate
```

```
     (ambiguate
      result-task
      (make-decomp
       (add-task result-task
                 (substitute-task task newtask (tasks C)))
       t)))))

(defmethod admixable-symptoms ((task task) (s symptom))
  (unless (= (cardinality (cluster task)) 1)
    (loop with new-set = (make-set (list s))
          for admix-s in (symptoms-of (cluster task))
          when (and (restricts? admix-s
                                (remove-elt admix-s (cluster task)))
                    (restricts? admix-s new-set))
          collect admix-s)))
```

A.9 Nodes

Nodes are the components of the search tree. Each node contains a decomposition and is related to a parent node and children nodes. For debugging and explanation purposes, each node has an operator, which explains how its decomposition was derived from its parent decomposition.

```
(defclass search-class ()
  ((level :accessor level :initarg :level)
   (probability :accessor prob :initform 0.0)))

(defclass node (search-class)
  ((decomp :accessor decomp :initarg :decomp)
   (parent :accessor parent :initarg :parent)
   (operator :accessor operator :initarg :operator)
   (children :accessor children :initform '())))

(defmethod make-node ((C decomp) (parent node) operator)
  (let ((node (make-instance 'node
                             :level (1+ (level parent))
                             :decomp C
                             :parent parent
                             :operator operator)))
    (push node (children parent))
    node))
```

```
(defmethod make-root-node ()
  (make-instance 'node
                 :decomp (make-decomp '())
                 :level 0
                 :parent :none
                 :operator :root))

(defmethod make-nodes (decomps (parent node) operator)
  (map 'list #'(lambda (C) (make-node C parent operator))
       (map 'list #'standardize (remove :degenerate decomps))))
```

The following procedures are used for comparing and sorting nodes, based on their decompositions.

```
(defmethod equal-nodes? ((node1 node) (node2 node))
  (equal-decomps? (decomp node1) (decomp node2)))

(defmethod node<? ((node1 node) (node2 node))
  (decomp<? (decomp node1) (decomp node2)))
```

Frontiers are simply the nodes generated after an iteration of the search process. In this program, a breadth-first strategy is used, so that each frontier is simply one level of the search tree.

```
(defclass frontier ()
  ((nodes :accessor nodes :initarg :nodes)))

(defmethod make-frontier (nodes)
  (make-instance 'frontier :nodes nodes))

(defmethod make-initial-frontier ()
  (reset-case)
  (make-instance 'frontier
                 :nodes (list (make-root-node))))
```

A.10 Search

The following function is the top-level procedure for decompositional search. It takes a set of symptoms and produces a frontier containing plausible problem decompositions. The procedure takes a frontier as an optional argument so that a previous search process can be extended.

```
(defmethod diagnose
    (symptoms &optional (frontier (make-initial-frontier)))
  (loop for s in symptoms
        do (enter-symptom-in-case s)
           (setf frontier (make-frontier
                           (compute-differentials
                            (remove-duplicate-nodes
                             (expand frontier)))))
        finally (return frontier)))
```

Expansion consists of accumulating the successors for each node in the previous frontier. These successors are created by the symptom assignment operators. If covering is possible, no other operator need be tried. Otherwise, restricting, adjoining, and admixing operators are applied. Each new decomposition is then ambiguated and disambiguated.

```
(defmethod expand ((frontier frontier))
  (loop for n in (nodes frontier)
        append (successor n (get-next-symptom (level n)))))

(defmethod successor ((n node) (s symptom))
  (let ((C (decomp n)))
    (or (make-nodes (cover-op C s) n :cover)
        (append (make-nodes (restrict-op C s) n :restrict)
                (make-nodes (adjoin-op C s) n :adjoin)
                (make-nodes (admix-op C s) n :admix)))))
```

The following procedure removes duplicate decompositions. It speeds up the process by sorting the decompositions by size and lexicographic order.

```
(defmethod remove-duplicate-nodes (nodes)
  (let ((nodes (sort nodes #'node<?)))
    (unless (null nodes)
      (loop for node in (rest nodes)
            with test-node = (first nodes)
            with result = (list (first nodes))
            unless (equal-nodes? node test-node)
            do (push node result) (setf test-node node)
            finally (return (nreverse result))))))
```

The following procedure computes the differentials for a set of nodes. It removes those nodes that have incoherent decompositions.

```
(defmethod compute-differentials (nodes)
  (loop for node in nodes
        for decomp = (formulate-differentials (decomp node))
        unless (eq decomp :incoherent)
        do (setf (decomp node) decomp)
        and collect node))
```

Appendix B

Implementation of Candidate Generation Algorithm

The following is an implementation of the candidate generation algorithm written in ANSI Common Lisp [70]. This is the implementation used for comparison with decompositional search.

B.1 Class Definitions

Candidate generation uses the same classes for symptoms and disorders as in symptom clustering. It requires an additional class to specify a candidate, which consists of a set of disorders.

```
(defclass candidate (search-class)
   ((disorder-set :accessor disorder-set :initarg :disorder-set)))

(defmethod make-candidate (disorders level)
  (make-instance 'candidate
                 :level level
                 :disorder-set disorders))

(defmethod make-initial-candidate ()
  (reset-case)
  (make-instance 'candidate
                 :level 0
                 :disorder-set (make-empty-set)))

(defmethod update-level ((H candidate))
  (setf (level H) (1+ (level H)))
  H)
```

B.2 Search Routines

The following procedure is the top-level routine for candidate generation. Like the implementation for decompositional search, this implementation uses a breadth-first search strategy. The procedure takes a set of candidates as an optional argument so that previous searches can be extended.

```
(defmethod generate-candidates
    (symptoms &optional (candidates (list (make-initial-candidate))))
  (loop for s in symptoms
```

```
        do (enter-symptom-in-case s)
           (setf candidates (expand-candidates candidates))
        finally (return candidates)))
```

Expansion adds the possible causes for the new symptom to each existing candidate, unless the candidate already explains the symptom.

```
(defmethod expand-candidates (candidates)
  (loop for H in candidates
        append (cand-successor H (get-next-symptom (level H)))))

(defmethod cand-successor ((H candidate) (s symptom))
  (if (explains? H s)
      (list (update-level H))
      (loop with disorder-list = (disorders-of (disorder-set H))
            with seen-symptoms = (seen-symptoms (level H))
            for d in (disorders-of (causes s))
            when (minimal? disorder-list d seen-symptoms)
            collect (add-disorder d H))))

(defmethod add-disorder ((d disorder) (H candidate))
  (make-candidate (insert-elt d (disorder-set H)) (1+ (level H))))

(defmethod seen-symptoms (level) (make-full-set level))
```

B.3 Predicates

The following predicates provide the notions of minimality and validity.

```
(defmethod minimal? (disorder-list (new-d disorder) seen-symptoms)
  (let ((new-disorder-list (cons new-d disorder-list)))
    (notany #'(lambda (d)
                (valid? (remove d new-disorder-list) seen-symptoms))
            disorder-list)))

(defmethod valid? (disorder-list seen-symptoms)
  (superset? (compute-explained-symptoms disorder-list)
             seen-symptoms))

(defmethod explains? ((H candidate) (s symptom))
  (not (empty-set? (intersect (disorder-set H) (causes s)))))

(defmethod compute-explained-symptoms (disorder-list)
  (set-union-sets (map 'list #'effects disorder-list)))
```

Appendix C

Support Routines

In appendices A and B, we presented the basic implementation of the decompositional search and candidate generation algorithms. However, to use the algorithms themselves, additional supporting routines are needed. These routines are not part of the algorithms per se, but provide input and output facilities. Thus, the following support routines are only simple suggestions. Undoubtedly, an implementation intended for actual use would require more elaborate supporting facilities. As in the previous appendices, these routines are written in ANSI Common Lisp [70].

C.1 Link Probability Data Structures

The decompositional search algorithm in appendix A does not specify data structures for storing link probabilities. Here we offer one possible set of data structures for holding link probabilities.

```
(defvar *link-probabilities* (make-hash-table :test #'equal))

(defmethod reset-probability-tables ()
  (clrhash *link-probabilities*))

(defmethod link-prob ((d disorder) (s symptom))
  (let ((prob (gethash (cons (univ-index d) (univ-index s))
                       *link-probabilities*)))
    (if (null prob) 0.0 prob)))

(defmethod enter-link-prob (disorder-index symptom-index link-prob)
  (setf (gethash (cons disorder-index symptom-index)
                 *link-probabilities*)
        link-prob))
```

C.2 Knowledge Base Input

Here we provide a simple interface to read in a knowledge base. The knowledge base is assumed to be in three files. One file contains a list of disorders, with their prior probabilities and indices. A second file contains a list of symptoms, with their indices. A third file contains the links between symptoms and disorders. Each disorder, symptom, or link is contained on a separate line. The following definitions give one possible format for these files:

```
(defmethod dz-indexfile-index (string) (string-first string))
(defmethod dz-indexfile-prior-prob (string) (string-second string))
(defmethod dz-indexfile-name (string) (string-cddr string))

(defmethod sx-indexfile-index (string) (string-first string))
(defmethod sx-indexfile-name (string) (string-tail string))

(defmethod linkfile-disorder-index (string) (string-first string))
(defmethod linkfile-symptom-index (string) (string-second string))
(defmethod linkfile-link-prob (string) (string-third string))
```

The following procedures provide utility functions to obtain input from a file:

```
(defmacro doline ((string file) &body body)
  (let ((f (gensym)))
    `(with-open-file (,f ,file :direction :input)
       (loop while (not (end-of-file? ,f))
             for ,string = (read-next-line ,f)
             until (empty-string? ,string)
             do ,@body))))

(defmethod end-of-file? (file)
  (eq :eof (peek-char nil file nil :eof)))

(defmethod read-next-line (file)
  (unless (end-of-file? file)
    (loop for string = (read-line file)
          until (or (end-of-file? file)
                    (not (empty-string? string)))
          finally (return (trim-spaces string)))))

(defmethod empty-string? (string)
  (every #'(lambda (char) (member char '(#\Space #\Tab #\Page)))
         string))

(defmethod trim-spaces (string)
  (string-trim '(#\Space #\Tab #\Page) string))
```

The following procedures provide abstractions for extracting strings from each line of a file:

```
(defmethod string-head (string)
  (multiple-value-bind (value index)
      (read-from-string string)
    value))

(defmethod string-tail (string)
  (trim-spaces
   (subseq string
           (multiple-value-bind (value index)
               (read-from-string string)
             index))))

(defmethod string-first (string)
  (string-head string))
(defmethod string-second (string)
  (string-head (string-tail string)))
(defmethod string-third (string)
  (string-head (string-tail (string-tail string))))
(defmethod string-cddr (string)
  (string-tail (string-tail string)))
```

Often, we will want to look up disorders and symptoms by the name. The following procedures implement tables to associate the names of these objects with the objects themselves:

```
(defmethod enter-primitive ((p primitive))
  (enter-univ-index p)
  (enter-name p))

(defvar *symptom-nametable* (make-table #'equal))
(defvar *disorder-nametable* (make-table #'equal))

(defmethod reset-nametables ()
  (reset-table *symptom-nametable*)
  (reset-table *disorder-nametable*))

(defmethod enter-name ((s symptom))
  (enter-table (name s) s *symptom-nametable*))
(defmethod enter-name ((d disorder))
  (enter-table (name d) d *disorder-nametable*))
```

```
(defmethod lookup-symptom-name (name)
  (lookup-table name *symptom-nametable*))
(defmethod lookup-disorder-name (name)
  (lookup-table name *disorder-nametable*))
```

Finally, the following procedures provide routines for loading a diagnostic knowledge base.

```
(defmethod load-kb (dz-indexfile sx-indexfile linkfile)
  (reset-nametables)
  (reset-probability-tables)
  (read-dz-indexfile dz-indexfile)
  (read-sx-indexfile sx-indexfile)
  (read-linkfile-for-causes linkfile)
  (read-linkfile-for-probs linkfile))

(defmethod read-dz-indexfile (file)
  (doline (string file)
          (enter-primitive
           (make-disorder (dz-indexfile-name string)
                          (dz-indexfile-index string)
                          (dz-indexfile-prior-prob string)))))

(defmethod read-sx-indexfile (file)
  (doline (string file)
          (enter-primitive
           (make-symptom (sx-indexfile-name string)
                         (sx-indexfile-index string)))))

(defmethod read-linkfile-for-causes (file)
  (doline (string file)
          (let* ((disorder-index (linkfile-disorder-index string))
                 (symptom-index (linkfile-symptom-index string))
                 (s (lookup-symptom-univ-index symptom-index)))
            (reinitialize-instance
             s :univ-causes
             (set-insert (make-singleton-set disorder-index)
                         (univ-causes s))))))

(defmethod read-linkfile-for-probs (file)
  (doline (string file)
```

```
          (enter-link-prob
           (linkfile-disorder-index string)
           (linkfile-symptom-index string)
           (linkfile-link-prob string))))
```

C.3 Interface

Interfaces are highly machine-dependent. Here we provide a simple interface to print the results of the decompositional search and candidate generation algorithms. The output is intended to be used for processing by TEX [32, 35].

```
(defvar *print* 'name)

(defmethod print-object ((s symptom) stream)
  (cond ((eq *print* 'name)
         (format stream "~S" (name s)))
        ((eq *print* 'univ-index)
         (format stream "s_{~D}" (univ-index s)))
        (t (format stream "s_{~D}" (case-index s)))))

(defmethod print-object ((d disorder) stream)
  (cond ((eq *print* 'name)
         (format stream "~S" (name d)))
        ((eq *print* 'univ-index)
         (format stream "d_{~D}" (univ-index d)))
        (t (format stream "d_{~D}" (case-index d)))))

(defmethod print-frontier ((f frontier))
  (loop for node in (nodes f)
        for i from 1
        do (format t "~%~S <--(~S)-- ~S"
                   node (operator node) (parent node))
           (format t "~%${\\cal C}_~D =" i)
           (print-node node)))

(defmethod print-node (node)
  (print-tasks (sort-task-set (tasks (decomp node)))))
```

```
(defmethod print-tasks (tasks)
  (print-decomposition tasks)
  (format t "$\\\\")
  (format t "~%Differentials $\\langle ")
  (print-diff-sizes tasks)
  (format t " \\rangle$:~%")
  (format t "$")
  (print-differentials tasks)
  (format t "$~%"))

(defmethod print-decomposition (tasks)
  (loop for task in tasks do (print-cluster task)))

(defmethod print-diff-sizes (tasks)
  (loop for task in (butlast tasks)
        do (print-diff-size task)
           (format t " \\cross ")
        finally (print-diff-size (car (last tasks)))))

(defmethod print-diff-size (task)
  (format t "~D" (cardinality (diff task))))

(defmethod print-differentials (tasks)
  (loop for task in (butlast tasks)
        do (print-diff task)
           (format t "~%  \\cross~%")
        finally (print-diff (car (last tasks)))))

(defmethod print-cluster ((task task))
  (format t "~S" (symptoms-of (cluster task))))

(defmethod print-diff ((task task))
  (format t "\\{")
  (loop for disorder in (disorders-of (diff task))
        do (format t "~S" disorder))
  (format t "\\}"))

(defmethod sort-candidates-by-size (candidates)
  (stable-sort candidates #'< :key #'size))

(defmethod size ((H candidate))
  (cardinality (disorder-set H)))
```

The following procedures provide similar output routines for printing candidates:

```
(defmethod print-candidates (candidates)
  (loop for H in (sort-candidates-by-size candidates)
        with current-size = 0
        when (> (size H) current-size)
        do (format t "~%~%Cardinality ~D:" (size H))
           (setf current-size (size H))
           (print-candidate H)
        else do (print-candidate H)))

(defmethod print-candidate ((H candidate))
  (format t "[")
  (loop for d in (disorders-of (disorder-set H))
        do (format t "~S " d))
  (format t "]"))
```

Appendix D

Subdomain for Prerenal Azotemia

D.1 Symptoms

This is a list of the possible effects of the disorder prerenal azotemia:

s_1 Azotemia Of Two Week (s) Duration Or Less
s_2 Creatinine Clearance Decreased
s_3 Creatinine Serum 3 To 10 Mg Per Dl
s_4 Creatinine Serum Increased Not Over 2.9 Mg Per Dl
s_5 Dehydration
s_6 Mouth Mucosa Dry (Xerostomia)
s_7 Oliguria Hx
s_8 Ph Urine Less Than 6
s_9 Sodium Urine Less Than 20 Meq Per Day
s_{10} Urea Nitrogen Serum 30 To 59
s_{11} Urea Nitrogen Serum 60 To 100
s_{12} Urine Osmolality Gtr Than 320
s_{13} Urine Output Less Than 400 Ml Per Day
s_{14} Urine Specific Gravity Gtr Than 1.020

D.2 Disorders

This is a list of the competitors of the prerenal azotemia. These are disorders that can cause one or more of the effects of prerenal azotemia.

d_1 Addisons Disease Secondary To Adrenal Destruction
d_2 Addisons Disease Secondary To Idiopathic Atrophy
d_3 Adrenal Apoplexy
d_4 Adrenal Insufficiency Secondary To Hypopituitarism
d_5 Aldosteronism Primary
d_6 Aldosteronism Secondary
d_7 Alzheimers Disease
d_8 Amebic Colitis
d_9 Amyloidosis Systemic
d_{10} Analgesic Nephropathy
d_{11} Angiodysplasia Of Right Colon
d_{12} Anorexia Nervosa

d_{13} Arteriolar Nephrosclerosis Benign (Essential Hypertension)
d_{14} Arteriolar Nephrosclerosis Malignant (Malignant Hypertension)
d_{15} Aspergillosis Disseminated
d_{16} Atheromatous Embolism
d_{17} Botulism
d_{18} Brain Neoplasm Secondary Multiple
d_{19} Campylobacter Enteritis
d_{20} Carcinoid Syndrome Secondary To Bronchial Neoplasm
d_{21} Carcinoid Syndrome Secondary To Hepatic Metastases
d_{22} Carcinoma Of Esophagus
d_{23} Cardiac Failure Left Chronic Congestive
d_{24} Cardiac Failure Right Congestive
d_{25} Cardiogenic Shock Acute
d_{26} Celiac Sprue
d_{27} Ceramide Trihexoside Lipoidosis (Fabrys Disease)
d_{28} Cerebral Artery Thrombosis Or Dissection With Encephalomalacia
d_{29} Cerebral Embolism
d_{30} Cerebral Lymphoma Primary
d_{31} Cerebral Malaria
d_{32} Cerebral Neoplasm Single Frontal
d_{33} Cerebral Neoplasm Single Parietal
d_{34} Cerebral Neoplasm Single Temporal
d_{35} Constrictive Pericarditis
d_{36} Crohns Disease Of Colon
d_{37} Crohns Disease Of Small Intestine
d_{38} Cryoimmunoglobulinemic Syndrome
d_{39} Cryptococcal Meningitis
d_{40} Cushings Syndrome Secondary To Adrenal Adenoma (s)
d_{41} Cushings Syndrome Secondary To Adrenal Carcinoma
d_{42} Cushings Syndrome Secondary To Iatrogenic Steroid Excess
d_{43} Diabetes Insipidus
d_{44} Diabetes Insipidus Nephrogenic
d_{45} Diabetes Mellitus
d_{46} Diabetic Ketoacidosis
d_{47} Diabetic Nephropathy

d_{48} Ectopic Acth Syndrome
d_{49} Encephalitis Acute Viral
d_{50} Fatty Liver Of Pregnancy Acute
d_{51} Gastrointestinal Sarcoidosis
d_{52} Glomerulonephritis Acute
d_{53} Glomerulonephritis Advanced Chronic
d_{54} Glomerulonephritis Focal
d_{55} Glomerulonephritis Latent
d_{56} Glomerulonephritis Rapidly Progressive
d_{57} Goodpasture Syndrome (Renal Component)
d_{58} Gouty Nephropathy Chronic
d_{59} Heat Exhaustion
d_{60} Hereditary Nephritis (Alports Syndrome)
d_{61} Herpes Simplex Encephalitis
d_{62} Histoplasma Meningitis
d_{63} Histoplasmosis Disseminated
d_{64} Hydronephrosis
d_{65} Hyperparathyroidism Primary
d_{66} Hyperthyroidism (Graves Disease)
d_{67} Hypokalemic Nephropathy
d_{68} Hypovolemic Shock
d_{69} Iga Nephropathy
d_{70} Immune Deficiency Syndrome Acquired (AIDS)
d_{71} Intestinal Giardiasis
d_{72} Intracerebral Hematoma
d_{73} Lead Nephropathy Chronic
d_{74} Lead Poisoning
d_{75} Left Ventricular Failure Acute
d_{76} Leukemia Chronic Lymphocytic
d_{77} Listeria Meningitis
d_{78} Lupus Nephritis
d_{79} Malaria
d_{80} Mallory Weiss Syndrome
d_{81} Medullary Cystic Kidney
d_{82} Membranous Glomerulopathy

d_{83} Meningococcal Meningitis
d_{84} Meningococcemia Acute
d_{85} Myeloid Metaplasia (Primary Myelofibrosis)
d_{86} Nephritis Acute Interstitial Allergic
d_{87} Nephritis Interstitial Non Allergic
d_{88} Nephrolithiasis
d_{89} Nephrotic Syndrome
d_{90} Pancreatic Cholera
d_{91} Pancreatitis Acute
d_{92} Paroxysmal Nocturnal Hemoglobinuria Involving Kidneys
d_{93} Peptic Ulcer With Hemorrhage
d_{94} Peritonitis Acute Generalized
d_{95} Pheochromocytoma
d_{96} Pituitary Cushings Syndrome
d_{97} Plague Meningitis
d_{98} Plasma Cell Myeloma
d_{99} Pneumococcal Meningitis
d_{100} Polycystic Renal Disease
d_{101} Polymyositis/Dermatomyositis
d_{102} Porphyria Acute Intermittent
d_{103} Prerenal Azotemia
d_{104} Presinusoidal Portal Hypertension
d_{105} Progressive Systemic Sclerosis
d_{106} Progressive Systemic Sclerosis Involving Kidneys
d_{107} Pseudomembranous Colitis
d_{108} Pyelonephritis Acute
d_{109} Pyelonephritis Chronic
d_{110} Pyloric Obstruction
d_{111} Renal Amyloidosis
d_{112} Renal Artery Stenosis
d_{113} Renal Cell Carcinoma
d_{114} Renal Failure Acute
d_{115} Renal Failure Chronic (Uremia)
d_{116} Renal Failure Secondary To Liver Disease (Hepatorenal Syndrome)
d_{117} Renal Infarction

d_{118} Renal Interstitial Sarcoidosis
d_{119} Renal Leptospirosis
d_{120} Renal Thrombotic Thrombocytopenic Purpura
d_{121} Renal Tuberculosis
d_{122} Renal Tubular Acidosis Distal
d_{123} Renal Tubular Acidosis Proximal (Fanconi Syndrome)
d_{124} Renal Vasculitis
d_{125} Renal Vein Thrombosis
d_{126} Rocky Mountain Spotted Fever
d_{127} Salt Losing Nephritis
d_{128} Shigellosis
d_{129} Sinusoidal Or Postsinusoidal Portal Hypertension
d_{130} Sjogrens Syndrome
d_{131} Small Bowel Obstruction
d_{132} Small Intestinal Lymphoma
d_{133} Staphylococcal Scarlet Fever (Toxic Shock Syndrome)
d_{134} Staphylococcus Aureus Meningitis
d_{135} Subdural Hematoma
d_{136} Superior Mesenteric Artery Insufficiency Acute
d_{137} Superior Mesenteric Vein Thrombosis
d_{138} Thrombotic Thrombocytopenic Purpura
d_{139} Thyrotoxic Storm
d_{140} Toxemia Of Pregnancy
d_{141} Trichinosis
d_{142} Tuberculous Meningitis
d_{143} Tubular Necrosis Acute
d_{144} Tularemia
d_{145} Typhoid Fever
d_{146} Ulcerative Colitis
d_{147} Waldenstroms Macroglobulinemia

D.3 Causal Links

This is a listing of the links between symptoms and disorders in the subdomain for prerenal azotemia:

$\text{Causes}(s_1) = \{d_{103}\ d_{114}\}$

$\text{Causes}(s_2) = \{d_{10}\ d_{13}\ d_{14}\ d_{16}\ d_{47}\ d_{52}\ d_{53}\ d_{54}\ d_{55}\ d_{56}\ d_{57}\ d_{58}\ d_{60}\ d_{64}\ d_{69}\ d_{73}\ d_{74}\ d_{78}\ d_{81}\ d_{82}\ d_{92}\ d_{98}\ d_{100}\ d_{103}\ d_{106}\ d_{109}\ d_{111}\ d_{112}\ d_{114}\ d_{115}\ d_{116}\ d_{117}\ d_{118}\ d_{120}\ d_{122}\ d_{123}\ d_{124}\ d_{125}\ d_{140}\ d_{143}\}$

$\text{Causes}(s_3) = \{d_{10}\ d_{13}\ d_{14}\ d_{16}\ d_{27}\ d_{47}\ d_{52}\ d_{53}\ d_{56}\ d_{57}\ d_{58}\ d_{60}\ d_{64}\ d_{69}\ d_{73}\ d_{78}\ d_{81}\ d_{82}\ d_{86}\ d_{87}\ d_{92}\ d_{100}\ d_{103}\ d_{106}\ d_{109}\ d_{111}\ d_{112}\ d_{114}\ d_{115}\ d_{116}\ d_{117}\ d_{118}\ d_{119}\ d_{120}\ d_{121}\ d_{124}\ d_{125}\ d_{143}\}$

$\text{Causes}(s_4) = \{d_5\ d_9\ d_{10}\ d_{13}\ d_{14}\ d_{16}\ d_{27}\ d_{38}\ d_{43}\ d_{44}\ d_{47}\ d_{52}\ d_{53}\ d_{54}\ d_{55}\ d_{56}\ d_{57}\ d_{58}\ d_{60}\ d_{63}\ d_{64}\ d_{67}\ d_{69}\ d_{73}\ d_{74}\ d_{76}\ d_{78}\ d_{81}\ d_{82}\ d_{84}\ d_{85}\ d_{86}\ d_{87}\ d_{89}\ d_{92}\ d_{95}\ d_{98}\ d_{100}\ d_{101}\ d_{103}\ d_{105}\ d_{106}\ d_{108}\ d_{109}\ d_{111}\ d_{112}\ d_{116}\ d_{117}\ d_{118}\ d_{119}\ d_{120}\ d_{121}\ d_{122}\ d_{123}\ d_{124}\ d_{125}\ d_{127}\ d_{133}\ d_{138}\ d_{140}\ d_{143}\ d_{147}\}$

$\text{Causes}(s_5) = \{d_1\ d_2\ d_3\ d_4\ d_7\ d_8\ d_{10}\ d_{11}\ d_{12}\ d_{17}\ d_{18}\ d_{19}\ d_{22}\ d_{26}\ d_{28}\ d_{29}\ d_{30}\ d_{31}\ d_{32}\ d_{33}\ d_{34}\ d_{36}\ d_{37}\ d_{39}\ d_{43}\ d_{44}\ d_{45}\ d_{46}\ d_{49}\ d_{50}\ d_{59}\ d_{61}\ d_{62}\ d_{64}\ d_{65}\ d_{68}\ d_{70}\ d_{71}\ d_{72}\ d_{77}\ d_{79}\ d_{80}\ d_{83}\ d_{86}\ d_{87}\ d_{90}\ d_{91}\ d_{93}\ d_{94}\ d_{97}\ d_{98}\ d_{99}\ d_{102}\ d_{103}\ d_{104}\ d_{107}\ d_{110}\ d_{114}\ d_{115}\ d_{125}\ d_{126}\ d_{127}\ d_{128}\ d_{129}\ d_{131}\ d_{132}\ d_{134}\ d_{135}\ d_{136}\ d_{137}\ d_{139}\ d_{142}\ d_{143}\ d_{144}\ d_{145}\ d_{146}\}$

$\text{Causes}(s_6) = \{d_1\ d_2\ d_3\ d_4\ d_{12}\ d_{17}\ d_{20}\ d_{21}\ d_{27}\ d_{43}\ d_{44}\ d_{45}\ d_{46}\ d_{50}\ d_{59}\ d_{66}\ d_{86}\ d_{87}\ d_{90}\ d_{102}\ d_{103}\ d_{114}\ d_{115}\ d_{127}\ d_{130}\ d_{139}\ d_{141}\ d_{143}\}$

$\text{Causes}(s_7) = \{d_{14}\ d_{23}\ d_{24}\ d_{52}\ d_{53}\ d_{56}\ d_{57}\ d_{59}\ d_{69}\ d_{78}\ d_{103}\ d_{106}\ d_{112}\ d_{119}\ d_{120}\ d_{124}\ d_{133}\ d_{143}\}$

$\text{Causes}(s_8) = \{d_{10}\ d_{46}\ d_{52}\ d_{67}\ d_{103}\ d_{115}\ d_{123}\}$

$\text{Causes}(s_9) = \{d_5\ d_6\ d_{23}\ d_{24}\ d_{35}\ d_{103}\ d_{116}\}$

$\text{Causes}(s_{10}) = \{d_5\ d_9\ d_{10}\ d_{13}\ d_{14}\ d_{15}\ d_{16}\ d_{25}\ d_{27}\ d_{38}\ d_{40}\ d_{41}\ d_{42}\ d_{43}\ d_{44}\ d_{47}\ d_{48}\ d_{51}\ d_{52}\ d_{53}\ d_{54}\ d_{55}\ d_{56}\ d_{57}\ d_{58}\ d_{60}\ d_{63}\ d_{64}\ d_{67}\ d_{68}\ d_{69}\ d_{73}\ d_{74}\ d_{76}\ d_{78}\ d_{80}\ d_{81}\ d_{82}\ d_{84}\ d_{86}\ d_{87}\ d_{89}\ d_{92}\ d_{95}\ d_{96}\ d_{98}\ d_{100}\ d_{103}\ d_{104}\ d_{105}\ d_{106}\ d_{108}\ d_{109}\ d_{111}\ d_{112}$

d_{116} d_{117} d_{118} d_{119} d_{120} d_{121} d_{122} d_{123} d_{124} d_{125} d_{127} d_{129} d_{133} d_{138} d_{140} d_{143} $d_{147}\}$

Causes(s_{11}) = $\{d_{10}$ d_{13} d_{14} d_{15} d_{16} d_{27} d_{47} d_{52} d_{53} d_{56} d_{57} d_{58} d_{60} d_{64} d_{69} d_{73} d_{78} d_{81} d_{86} d_{87} d_{92} d_{100} d_{103} d_{106} d_{109} d_{111} d_{112} d_{114} d_{115} d_{116} d_{117} d_{118} d_{119} d_{120} d_{121} d_{124} d_{125} $d_{143}\}$

Causes(s_{12}) = $\{d_{14}$ d_{52} d_{69} d_{103} d_{116} d_{124} $d_{140}\}$

Causes(s_{13}) = $\{d_{14}$ d_{16} d_{47} d_{52} d_{53} d_{56} d_{57} d_{64} d_{69} d_{78} d_{86} d_{87} d_{88} d_{100} d_{103} d_{106} d_{108} d_{111} d_{112} d_{113} d_{116} d_{117} d_{119} d_{120} d_{124} d_{125} d_{127} d_{140} $d_{143}\}$

Causes(s_{14}) = $\{d_{14}$ d_{23} d_{24} d_{52} d_{69} d_{75} d_{103} d_{116} d_{124} $d_{140}\}$

Bibliography

[1] Eric T. Bell. Exponential numbers. *American Mathematical Monthly*, 41:411–419, 1934.

[2] Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts, 1984.

[3] Tom Bylander, Dean Allemang, Michael C. Tanner, and John R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49:25–60, 1991.

[4] Eugene Charniak and Robert Goldman. A probabilistic model of plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*, pages 160–165. American Association for Artificial Intelligence, 1991.

[5] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA, 1985.

[6] William J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[7] William J. Clancey and Edward H. Shortliffe, editors. *Readings in Medical Artificial Intelligence: The First Decade*. Addison Wesley, Reading, Mass., 1984.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[9] P. T. Cox and T. Pietrzykowski. General diagnosis by abductive inference. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 183–189, 1987.

[10] Paul Cutler. *Problem Solving in Clinical Medicine*. Williams and Wilkins, Baltimore, 1985.

[11] Randall Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[12] Randall Davis and Walter Hamscher. Model-based reasoning: Troubleshooting. In Shrobe [66], pages 297–346.

[13] Johan de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, 28:197–224, 1986.

[14] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[15] David M. Eddy and Charles H. Clanton. The art of diagnosis: Solving the clinicopathological exercise. *New England Journal of Medicine*, 306:1263–1268, 1982.

[16] Arthur S. Elstein, Lee S. Shulman, and Sarah A. Sprafka. *Medical Problem Solving: An Analysis of Clinical Reasoning*. Harvard University Press, Cambridge, Mass., 1978.

[17] Douglas Fisher and Pat Langley. Approaches to conceptual clustering. In *International Joint Conference on Artificial Intelligence*, pages 691–697, 1985.

[18] Janet Gale and Philip Marsden. *Medical Diagnosis: From Student to Clinician*. Oxford University Press, Oxford, 1983.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[20] H. Gelernter. Realization of a geometry-theorem proving machine. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 134–152. McGraw-Hill, New York, 1963.

[21] G. Anthony Gorry. Strategies for computer-aided diagnosis. *Mathematical Biosciences*, 2:293–318, 1968.

[22] Edmund J. Graves. National hospital discharge survey: Annual summary, 1987. Vital and Health Statistics 13(99), National Center for Health Statistics, 1989.

[23] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis (research note). *Artificial Intelligence*, 41:79–88, 1989.

[24] Walter C. Hamscher. *Model-Based Troubleshooting of Digital Systems*. PhD thesis, Massachusetts Institute of Technology, August 1988.

[25] Gilbert Harman. The inference to the best explanation. *Philosophical Review*, LXXIV:88–95, 1965.

[26] David Heckerman and Randolph A. Miller. Towards a better understanding of INTERNIST-1 knowledge base. In R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 22–26, Washington, October 1986. North-Holland.

[27] David E. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In Max Henrion, Ross D. Shachter, Laveen N. Kanal, and John F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, Machine Intelligence and Pattern Recognition Series, Volume 10, pages 163–171. North-Holland, 1990.

[28] Jerry R. Hobbs, Mark Stickel, Paul Martin, and Douglas Edwards. Interpretation as abduction. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 95–103, 1988.

[29] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, 1986.

[30] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[31] Paul D. Hubbe and Eugene C. Freuder. An efficient cross product representation of the constraint problem search space. In *Proceedings of the National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 1992.

[32] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[33] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[34] Thomas S. Kuhn. *The Structure of Scientific Revolutions*. The University of Chicago Press, second edition, 1970.

[35] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, 1986.

[36] Douglas B. Lenat. The nature of heuristics. *Artificial Intelligence*, 19:189–249, 1982.

[37] Ryszard S. Michalski and Robert E. Stepp. Automated construction of classifications: conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5:396–410, 1983.

[38] R. A. Miller, M. A. McNeil, S. M. Challinor, F. E. Masari, Jr., and J. D. Myers. The Internist-1/Quick Medical Reference project—Status report. *Western Journal of Medicine*, 145:816–822, 1986.

[39] Randolph A. Miller, Harry E. Pople, Jr., and Jack D. Myers. Internist-1, an experimental computer-based diagnostic consultant for general internal medicine. *New England Journal of Medicine*, 307:468–476, 1982.

[40] Marvin Minsky. Steps toward artificial intelligence. In Edward A. Feigenbaum and Jerome Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, 1963.

[41] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[42] Tom M. Mitchell, R. M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1(1):47–80, 1986.

[43] Edmond A. Murphy. *The Logic of Medicine*. Johns Hopkins, Baltimore, 1976.

[44] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, 1972.

[45] Ramesh S. Patil. Causal representation of patient illness for electrolyte and acid-base diagnosis. TR 267, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, October 1981.

[46] Ramesh S. Patil. Causal reasoning in computer programs for medical diagnosis. *Computer Methods and Programs in Biomedicine*, 25:117–124, 1987.

[47] Ramesh S. Patil. Artificial intelligence techniques for diagnostic reasoning in medicine. In Shrobe [66], pages 347–379.

[48] Ramesh S. Patil, Peter Szolovits, and William B. Schwartz. Causal understanding of patient illness in medical diagnosis. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 893–899, 1981.

[49] Stephen G. Pauker, G. Anthony Gorry, Jerome P. Kassirer, and William B. Schwartz. Towards the simulation of clinical cognition: Taking a present illness by computer. *American Journal of Medicine*, 60:981–996, 1976.

[50] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

[51] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[52] Judea Pearl and Richard E. Korf. Search techniques. *Annual Reviews of Computer Science*, 2:451–467, 1987.

[53] Charles S. Peirce. *Collected Papers*. Harvard University Press, 1960.

[54] Yun Peng and James A. Reggia. Plausibility of diagnostic hypotheses: The nature of simplicity. In *Proceedings of the National Conference on*

Artificial Intelligence, pages 140–145. American Association for Artificial Intelligence, 1986.

[55] Yun Peng and James A. Reggia. A probabilistic causal model for diagnostic problem solving—Part I: Integrating symbolic causal inference with numeric probabilistic inference. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17:146–162, 1987.

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

[57] Harry E. Pople, Jr. Heuristic methods for imposing structure on ill-structured problems: The structuring of medical diagnostics. In Szolovits [73], pages 119–190.

[58] James A. Reggia, Dana S. Nau, and Pearl Y. Wang. Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, 19:437–460, 1983.

[59] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–96, 1987.

[60] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[61] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.

[62] Earl D. Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, 1977.

[63] William B. Schwartz, Ramesh S. Patil, and Peter Szolovits. Artificial intelligence in medicine: Where do we stand? *New England Journal of Medicine*, 316:685–688, 1987.

[64] Bart Selman and Hector J. Levesque. Abductive and default reasoning: A computational core. In *Proceedings of the National Conference on Artificial Intelligence*, pages 343–348. American Association for Artificial Intelligence, 1990.

[65] Edward H. Shortliffe. MYCIN: *Computer-based Medical Consultations*. American Elsevier, New York, 1976.

[66] Howard Shrobe, editor. *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*. Morgan Kaufman, 1988.

[67] Herbert A. Simon. The structure of ill-structured problems. *Artificial Intelligence*, 4:181–201, 1973.

[68] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, second edition, 1981.

[69] James R. Slagle. A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the Association for Computing Machinery*, 10:507–520, 1963.

[70] Guy L. Steele Jr. *Common LISP: The Language*. Digital Press, second edition, 1990.

[71] Mark Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–140, 1981.

[72] Mark Stefik. Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16:141–170, 1981.

[73] Peter Szolovits, editor. *Artificial Intelligence in Medicine*, volume 51 of *AAAS Selected Symposium Series*. Westview Press, Boulder, Colorado, 1982.

[74] Peter Szolovits, Ramesh S. Patil, and William B. Schwartz. Artificial intelligence in medical diagnosis. *Annals of Internal Medicine*, 108:80–87, 1988.

[75] Peter Szolovits and Stephen G. Pauker. Categorical and probabilistic reasoning in medical diagnosis. *Artificial Intelligence*, 11:115–144, 1978.

[76] Richard Waldinger. Achieving several goals simultaneously. In E. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 94–136. Edinburgh University Press, 1977.

[77] Sholom M. Weiss, Casimir A. Kulikowski, Saul Amarel, and Aaron Safir. A model-based method for computer-aided medical decision making. *Artificial Intelligence*, 11:145–172, 1978.

[78] Jean D. Wilson et al., editors. *Harrison's Principles of Internal Medicine*. McGraw-Hill, New York, twelfth edition, 1991.

[79] Patrick H. Winston. Learning structural descriptions from examples. In Patrick H. Winston, editor, *The Psychology of Computer Vision*, chapter 5, pages 157–209. McGraw Hill, New York, 1975.

[80] Thomas D. Wu. Symptom clustering and syndromic knowledge in diagnostic problem solving. In *Proceedings of the Thirteenth Symposium on Computer Applications in Medical Care*, pages 45–49, Washington, November 1989. IEEE Computer Society.

[81] Thomas D. Wu. Efficient diagnosis of multiple disorders based on a symptom clustering approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 357–364. American Association for Artificial Intelligence, 1990.

[82] Thomas D. Wu. Domain structure and the complexity of diagnostic problem solving. In *Proceedings of the National Conference on Artificial Intelligence*, pages 855–861. American Association for Artificial Intelligence, 1991.

[83] Thomas D. Wu. Probabilistic evaluation of candidate sets for multidisorder diagnosis. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 107–115. Elsevier Science Publishers B. V., 1991.

[84] Thomas D. Wu. A problem decomposition method for efficient diagnosis and interpretation of multiple disorders. *Computer Methods and Programs in Biomedicine*, 35:239–250, 1991.