CONSTRAINT PROCESSING

Rina Dechter

September 25, 2001

Contents

Preface

A constraint can intuitively be thought of as a restriction on a space of possibilities. Every piece of knowledge can be interpreted as narrowing the scope of this space, hence, as a constraint. It is not surprising, therefore, that constraints arise naturally in most areas of human endeavor. They are the most general means for formulating regularities that govern the computational, physical, biological and social worlds. For instance, the three angles of a triangle sum to 180 degrees; the four bases that make up DNA strands can only combine in particular orders; the sum of the currents flowing into a node must equal zero; Susan and John cannot both teach at the same time. Such restrictions represent useful pieces of knowledge from diverse disciplines, yet they share one feature in common: they identify the impossible, narrow down the realm of possibilities and, thus, permit us to focus more effectively on the possible.

Constraint problems have proven successful in modeling mundane cognitive tasks such as vision, language comprehension, default reasoning, diagnosis, scheduling, temporal and spatial reasoning. They allow for a natural, declarative formulation of what has to be satisfied, without the need to say how it has to be satisfied.

The current book provides a comprehensive in-depth coverage of the theory that underlies constraint processing algorithms as they emerged in the past three decades, primarily in the area of Artificial Intelligence.

This book aims at readers in diverse areas of computer science including Artificial Intelligence, Databases, Programming Languages, Systems, as well as fields related to computer science such as operation research, management science and applied mathematics.

The book focuses on the fundamental tools and principles that underlie reasoning with constraints, with special emphasis on the representation and analysis of algorithms. The books takes the reader in a step by step manner through definitions, examples, theory, algorithms and complexity analysis. It is intended primarily to graduate students, senior undergraduates, researchers and practitioners and could be used in a quarter/semester course dedicated to this topic, or as a reference book in a general class on artificial intelligence or optimization techniques. To accommodate different levels of reading, some chapters can be omitted without disturbing the flow of readings. Every chapter starts from the basic and practical aspects of its topic, and proceeds to the more advanced aspects, including those under current research.

The book focuses on constraints over discrete and finite domains. It describes the basic principles underlying relational representation first, and presents processing algorithms divided into two main categories: Inference-based and search-based. Search algorithms are characterized by backtracking search and its various enhancements, while inference algorithms are presented through a variety of constraint propagation methods (also known as consistency enforcing methods). Hybrids algorithms are also presented. A Graph-based view of both types of algorithms is emphasized throughout the book, to enhance the understanding and analysis of the various methods.

Chapter 1

Introduction

The notion of a constraint is one with which we are all very familiar in our day-to-day lives. We are used to dealing with situations in which, for example, the memory in our computers is constrained to hold a finite number of bytes, or only a limited number of hours remain until an important deadline arrives. Such problems, though we may not like them, rarely give us reason to pause. The situation becomes more complicated, however, as the number of constraints that must be satisfied simultaneously and the number of variables involved begin to grow. We find that it can take excruciatingly long to decide on the seating arrangement of guests for a formal dinner party or to get a large group of friends to agree on a movie for a night out.

In an even more complicated example, we can imagine the difficulty in scheduling classrooms for a semester of university instruction. We would need to find a classroom for every course while simultaneously satisfying the constraints that no two classes may be held in the same classroom at the same time, no professor can teach in two different classrooms at the same time, no class may be scheduled in the middle of the night, all classes must be offered in appropriately sized rooms or lecture halls, certain classes must not be scheduled at the same time, and so on.

As the complexity of the problem grows, it becomes appropriate to call on the help of computers to find an acceptable solution. In order to do so, computer scientists have devised languages to model constraint satisfaction problems and have developed methods for solving them. Using the language of constraints, computers have modeled mundane cognitive tasks such as vision, language comprehension, default reasoning, and abduction, as well as tasks that require high levels of human expertise such as scheduling, design, diagnosis, and temporal and spatial reasoning. In general, the tasks posed in the language of constraints are computationally intractable (NP-hard).

Over the last two to three decades, a great deal of theoretical and experimental research has been focused on developing general algorithms for solving constraint satisfaction problems, on identifying restricted subclasses that can be solved efficiently, and on developing approximation algorithms. This book will describe the theory and practice underlying such constraint processing methods.

The rest of the chapter is divided into two parts. First is an informal overview of constraint networks, starting with common examples of problems that can be modeled as constraint satisfaction problems and followed by an overview of the book along its chapters. Second is a review of mathematical concepts relevant to our discussion throughout the book.

1.1 Overview

1.1.1 Introduction to constraint networks

In general, problems involving constraints include two important components. First of all, every constraint problem must include *variables*, objects or items that can take on a variety of values. The set of possible values for a given variable is called its *domain*. For example, in trying to find an acceptable seating arrangement for a dinner party, we may chose to see the chairs as our variables, each with the same domain, which is the list of all guests.

It is important to note that there is often more than one way to model a problem. We could just as logically have decided that the guests would be the variables and that their domains would be the set of chairs at the table. In this case, with a (hopefully) one-to-one correspondence between chairs and guests, the choice makes little difference, but in other cases, one formulation of a problem may lend itself more readily to solution techniques than another.

The second component to every constraint problem is the set of constraints themselves. *Constraints* impose some kind of limitation on the values that a variable, or a combination of variables, may be assigned. If the host and hostess must sit at the two ends of the table, then their choices of seats are constrained. If two feuding guests must not be placed next to or directly opposite one another, then we must include this constraint as part of the overall problem statement.

Such a model including variables, their domains, and constraints is called a *constraint network*. Other terms used are *constraint problem* or *constraint store*. We use the term "network" to reflect the historical perspective when focus was initially restricted to sets of constraints whose dependencies can naturally be captured by simple graphs as well as to emphasize the importance of the constraint dependency structure for reasoning algorithms.

A *solution* is an assignment of a single value from its domain to each variable such that no constraint is violated. A problem may have one, many, or no solutions. A problem that has one or more solutions is termed *satisfiable* or *consistent*. If there is no possible assignment of values to the variables that satisfies all the constraints, then the network is termed *unsatisfiable* or *inconsistent*.

Typical tasks over constraint networks are: determining whether a solution exists, finding one or all solutions, finding whether a partial instantiation can be extended to a full solution, and finding an optimal solution relative to a given cost function. Such problems are referred to as *constraint satisfaction problems* or $CSPs$ for short.

1.1.2 Examples of constraint problems

We next give several common examples of problems that can be modeled as constraint satisfaction problems, including both simple, toy problems that help illustrate the principles involved, as well as more complex, real-world problems. At this point the specification of the constraints will be made rather informally. We revisit some of these examples in more detail in the next chapter and throughout the book.

The n-queens problem

A classic example frequently used to demonstrate constraint satisfaction problems is the n-queens problem. The n-queens problem asks us to place n queens on an $n \times n$ chess board, so that the placement of no queen constitutes an attack on any other. One possible constraint network formulation of the problem is as follows: there is a variable for each column of the chess board $x_1, \ldots, x_n$, the domains of the variables are the possible row positions $D_i = \{1, \ldots, n\}$, and the constraint on each pair of columns is that the two queens must not be on the same row or diagonal. One special property of this problem is that the number of variables is always the same as the number of values in each domain.

Crossword puzzles

Crossword puzzles have also been used in evaluating algorithms for solving constraint problems. The problem asks us to assign words from the dictionary (or from a given set of words) either vertically or horizontally according to certain constraints. If we allow each word to be placed in any space of the correct length, a possible constraint network formulation of the problem in Figure 1.1 is as follows: there is a variable for each square that can hold a character, the domains of the variables are dictated by the alphabet letters, and the constraints are dictated by the input of possible words:

$$\{HOSES, LASER, SHEET, SNAIL, STEER, ALSO, EARN, HIKE, IRON, SAME$$

$$EAT, LET, RUN, SUN, TEN, YES, BE, IT, NO, US\}$$

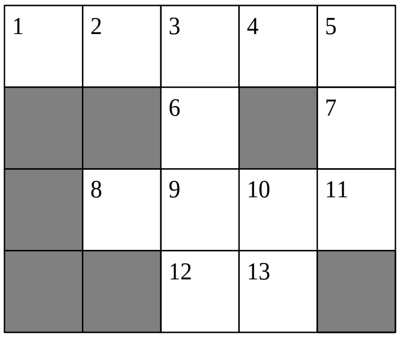


Figure 1.1: A crossword puzzle

So, for example, there is a constraint over the variables x_8, x_9, x_{10}, x_{11} that allows assigning only the four-letter words from the list to these four variables. The constraint can be described by the relation $C = \{(A, L, S, O), (E, A, R, N), (H, I, K, E)(I, R, O, N), (S, A, M, E)\}$. This means that x_8, x_9, x_{10}, x_{11} can be assigned, respectively, either $\{A, L, S, O\}$ or $\{E, A, R, N\}$, and so on. A solution to the constraint problem will generate an assignment of letters to the squares so that only the legal words are entered.

Map coloring and k-colorability

The map coloring problem is a well known problem that asks whether it is possible to color a map with only four colors when no two adjacent countries may share the same color. The problem was open for many years, and only recently was it positively solved.

Many resource allocation and communication problems can be abstracted to a more general form of this problem, known in the field of classical graph theory as *k-colorability*. Here, the map is abstracted to a graph with a node for each country and an edge joining any two countries that are neighbors. Given a graph of arbitrary size, the problem is to decide whether or not the graph can be colored with only k colors such that any two adjacent nodes in the graph must be of different colors. If the answer is yes, then a possible k-coloring of the graph's nodes should be given.

The graph k-colorability problem is a constraint satisfaction problem where there is a variable for each node in the graph, the domains of the variables are the possible colors (e.g., for every variable x_i, its domain is $D_i = \{red, blue, green\}$ or $D_i = \{1, 2, 3\}$), and the constraints are that every two adjacent nodes must be assigned different values (e.g., $A \neq B$). A specific example is given in Figure 1.2 where the variables are the countries denoted A, B, C, D, E, F, G. A solution to this constraint satisfaction problem is a legal k-coloring of the graph. A property of k-colorability problems is that their constraints are inequality constraints which appear in many problems and are among the "loosest"

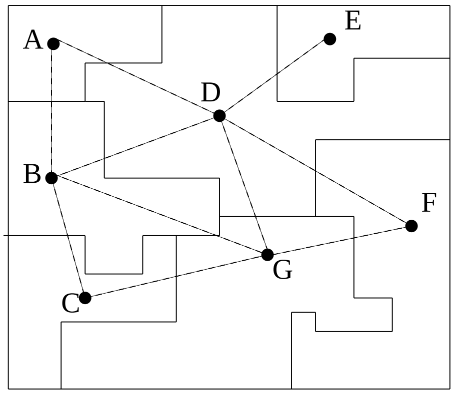


Figure 1.2: A map coloring problem

or "weakest" of constraints. A solution to the graph-coloring problem in Figure 1.2 using three colors is $(A = red, D = green, E = blue, B = blue, F = red, C = red, G = green)$.

Configuration and design

Configuration and location problems are a particularly interesting class for which constraint satisfaction formalism can be used. These problems arise in applications as diverse as automobile transmission design, micro-computer system configuration, and floor-plan layout. We consider here a simple example from the domain of architectural site location. Consider the map in Figure 1.3 [1], showing eight lots available for development. Five developments are to be located on these lots: a recreation area, an apartment complex, a cluster of 50 single-family houses, a large cemetery, and a dumpsite. Assume the following information and conditions are given:

- The recreation area should be near the lake.
- Steep slopes are to be avoided for all but the recreation area.
- Poor soil should be avoided for those developments that involve construction, namely the apartments and the family houses.
- The highway, being noisy, should not be near the apartments, the housing, or the recreation area.
- The dumpsite should not be visible from the apartments, the houses, or the lake.
- Lots 3 and 4 have bad soil.

[1] This example is due to Nadel.

- Lots 3, 4, 7, and 8 are on steep slopes .
- Lots 2, 3, and 4 are near the lake.
- Lots 1 and 2 are near the highway.

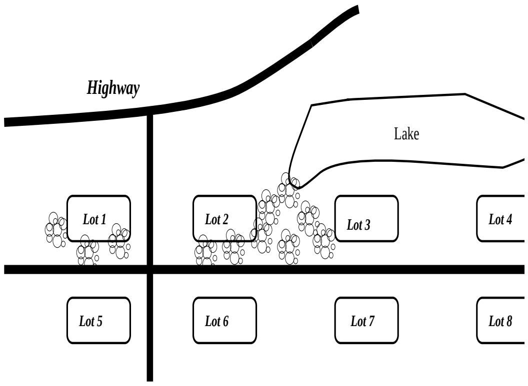


Figure 1.3: Development map

The problem of siting the five developments on the eight available lots so as to satisfy the given conditions can be naturally formulated as a constraint satisfaction problem. A variable is associated with each development:

- *Recreation*, *Apartments*, *Houses*, *Cemetery*, and *Dump*

The eight a priori possible sites for the developments constitute the common domain for the variables:

- $Domain = \{1, 2, 3, 4, 5, 6, 7, 8\}$

The information and conditions above are the basis for the constraints on the variables.

Huffman-Clowes scene labeling

One of the earliest constraint satisfaction problems was introduced in the early 1970's. This problem concerns the three-dimensional interpretation of a two-dimensional drawing. Huffman and Clowes [156] developed a basic labeling scheme of the arcs in block world picture graph, where $+$ stands for convex edge, $-$ for a concave edge, and $\rightarrow$ for occluding boundaries. They demonstrated that the possible combination of legal labelings associated with different junctions in such scenes can be enumerated as in Figure 1.4. The interpretation problem is to label the junction of a given drawing such that every junction is labeled according to one of its legal labelings and such that edges common to two junctions will receive the same label. The possible consistent labelings of a cube are presented in Figure 1.5. One formulation of this problem assigns variables for the lines in the figure with the domains being the possible labelings $\{+, -, \rightarrow, \leftarrow\}$. The constraints will restrict the labeling of adjacent lines in a junction according to the junction type.

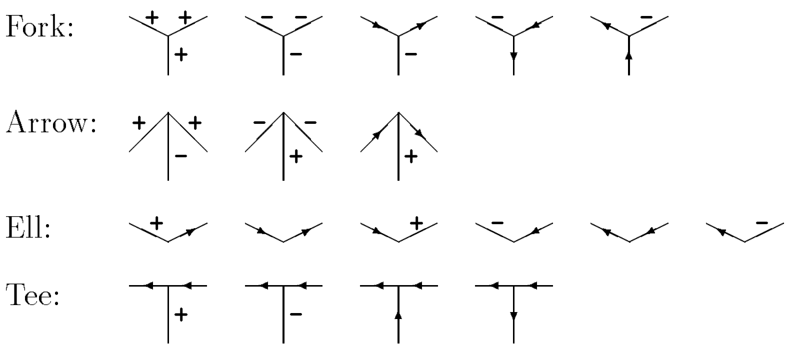


Figure 1.4: Huffman-Clowes junction labelings

Scheduling problems

One of the most practical application for constraint problems is scheduling. Such tasks involve scheduling activites or jobs, such that certain resource and temporal precedence constraints are satisfied. Well known problem classes are job-shop scheduling (where a typical resource is the availability of a machine for processing certain tasks) and time-tabling problems, requiring to schedule classroms and teachers to classes and to time slots such that natural constriants are satisfied. In such problems it is common to associate the tasks with variabes and their domain with the starting time of the task, and the constraints restricts those domains based on the problems constraints. A simple demonstrating example will be given in Chapter 2.

So far we gave examples of problems and their informal description as constraint problems. A formal treatment will be given in Chapter 2. This chapter concludes with overview of the book chapters and with mathematical background of concepts used throughout the book.

1.2 Book chapters overview

The book provides a comprehensive description of methods for processing and solving constraint problems. Techniques for processing constraints can be classified into two main categories: (1) search and (2) inference, and these techniques may also be combined. As

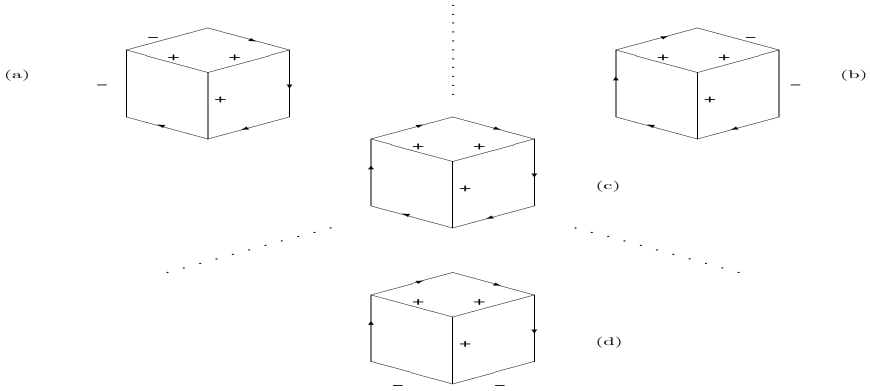


Figure 1.5: Solutions: (a) stuck on left wall, (b) stuck on right wall, (c) suspended in mid-air, (d) resting on floor.

mentioned, solving constraint problem is hard. The technical term used is NP-complete (see mathematical foundations).

Both search and inference offer methods that are systematic and complete, as well as other methods that are stochastic and incomplete. Complete algorithms are guaranteed to solve the problem is a solution exists or to decide that the problem is inconsistent. Incomplete algorithms, or approximation sometimes solve hard problems quickly, however they are not guaranteede to solve the problem even if given unbounded amount of time and space.

Inference. Chapters 3 and 4 as well as Chapters 8 and 9 focus on inference algorithms. Inference algorithms reason through the creation of desirable equivalent problems through problem reformulation. Most often, they create simpler problems that are easier to solve by a subsequent search algorithm. Occasionally, inference methods can even deliver a solution or prove the inconsistency of the network without the need for any further search.

In Chapter 3 we focus on the main concepts associated with inference methods for constraint networks which are called *local consistency* algorithms. Such algorithms take a telescopic view of subparts of the constraint network and demand that they contain no obviously extraneous or contradictory parts. For example, if a network consists of several variables $x_1, \ldots, x_n$, all with domains $D = \{1, \ldots, 10\}$, and if x_1 is constrained such that its value must be strictly greater than the value of every other variable, then it is easy to infer that there is no need for the value 1 in x_1's domain. Even if we ignore the whole network for the time being and focus only on x_1 and x_2, for example, we can see that if x_1 is instantiated with the value of 1, then there is no possible assignment of value to x_2 that satisfies the constraint. In the language of constraint networks, the value 1 is *inconsistent* and should be removed from x_1's domain because there is no solution to the problem that assigns $x_1 = 1$. (We could similarly reason that the value 10 should be removed from the domain of x_2.)

In general *local consistency* algorithms (also known as or constraint propagation) are polynomial algorithms that transform a given constraint network into an equivalent, yet more explicit, network by deducing new constraints which are added to the problem. For example, the most basic consistency algorithm which we will present, called *arc-consistency*, ensures that any legal value in the domain of a single variable has a legal match in the domain of any other selected variable. This is the level of consistency-enforcing that we saw in our example. *Path-consistency* ensures that any consistent solution to any two variables is extensible to any third variable, and, in general, i-*consistency* algorithms guarantee that any locally consistent instantiation of $i-1$ variables is extensible to any i^{th} variable. One can transform a given network to an i-consistent one by inferring some constraints. However, we will show that enforcing i-consistency is computationally expensive; it can be accomplished in time and space exponential in i. Moreover, these methods are not guarantee to find a solution. Sometimes however, they eliminate domain values to the point of completely emptying one or more domains, in which case they are able to declare the network *inconsistent*. All these methods and concepts will be ellaborated in Chapter 3.

Chapter 4 as well as Chapter 8 continue to focus on more advanced inference methods, some of which can provide a complete solution to the problem by by making the problem *globally consistent*. We will define such concepts and present less expensive directional consistency algorithms that enforce *global consisteny* only relative to a certain variable ordering such as Adaptive-consistency, will be presented. This is an inference algorithm that is typical of a class of variable-elimination algorithm. A unifying descrption of such algorithms is the focus of Chapter 8. Chapters 3 and 4 expose the readers to stucture-based analysis and parameters, such as *induced-width* that will accompany many of the subsequent chapters.

Search. The focus of Chapters 5 and 6 is on complete search algorithms. The most common algorithm for performing systematic search is *backtracking*, which traverses the space of partial solutions in a depth-first manner. Each step in the search represents the assignment of a value to one additional variable, thus extending the current partial (candidate) solution. When a variable is encountered such that none of its possible values are consistent with the current partial solution (a situation referred to as a *dead-end*), backtracking takes place. Namely, the algorithm returns to an earlier variable and attempts to assign it a new value such that the dead-end can now be avoided. The best performance of backtracking algorithm occurs when the algorithm is able to successfully assign a value to every variable without encountering any dead-ends. In such a case, backtracking is both time and space linear in the number of variables. Worst-case performance, however, while still requiring only linear space, is exponential in time.

Improvements to backtracking have focused on the two phases of the algorithm: moving forward (look-ahead schemes) described in Chapter 5, and backtracking (look-back schemes), the focus of Chapter 6. When moving forward to extend a partial solution, look-ahead schemes perform some computation to decide which variable (and even which of that variable's values) to choose next in order to enhance the efficiency of the search. Variables that participate in many constraints maximally constrain the rest of the search space and are thus preferred. In choosing a value from the domain of the selected variable, however, the least constraining value is preferred, in order to maximize options for future instantiations.

Look-back schemes are described in Chapter 6. They are are invoked when the algorithm encounters a dead-end and they perform two functions: One, they decide how far to backtrack by analyzing the reasons for the dead-end, a process often referred to as *back-jumping*. Two, they record the reasons for the dead-end in the form of new constraints, so that the same conflict will not arise again, a process known as *constraint learning* or *no-good recording*.

Chapter 7 describes approximation algorithms for search, in particular, a class known as *Stochastic local search* strategies (SLS). Those have been recently reintroduced into the satisfiability and constraint satisfaction literature under the umbrella name *GSAT (Greedy SATisfiability)*. These methods move in a hill-climbing manner in the space of complete instantiations of all the variables. The algorithm improves its current instantiation by iteratively changing (or 'flipping') the value of the variable to maximize the number of constraints satisfied. Such search algorithms are incomplete, may get stuck in a local maxima of their guiding cost function, and cannot always prove inconsistency. Nevertheless, when equipped with some heuristics for randomizing the search or for revising the guiding criterion function (constraint reweighting), they were shown successful in solving large and hard problems that are frequently difficult for backtracking-style search.

Chapter 9 takes the reader again to an inference algorithm. It concentrates on a structure-based compilation method called *tree-clustering*. It belongs to *structure-driven algorithms* that take advantage of the fact that it is possible to depict constraint networks as graphs where nodes represent variables and edges represent constraints. These techniques emerged from an attempt to topologically characterize constraint problems that are tractable. The basic network structure that support tractability is a tree. This has been observed repeatedly in constraint networks, complexity theory and database theory. Most other graph-based techniques can be viewed as transforming a given network into a meta-tree. For example, *tree-clustering* compiles a constraint problem into an equivalent tree of subproblems whose respective solutions can be efficiently combined into a complete solution. Algorithm is quite related to tree-clustering, and both are time and space exponentially bounded in a parameter of the constraint graph called induced-width.

Chapter 7 center on SAT. It specialize all the methods studied in general for CSPs to the special case of SAT. These include the main search algorithm known as DPbacktracking, the specialized variable-elimination algorithm Directional resolution as well as specialized local search methods.

Chapter 10 focus on hybrid of search and inference methods. When a problem is computationally hard for directional consistency algorithms, it can be solved by bounding the amount of consistency-enforcing (e.g. arc- or path-consistency) and augmenting the algorithm with a search component. As described, bounded inference can be done as preprocessing to search or during search. The chapter presents, the *cycle-cutset scheme* combining search and inference, which is exponentially bounded by the constraint graph's *cycle-cutset*. It then extends this approach to a general parameterized hybrid scheme whose parameter b bounds the level of inference allowed. Such combined methods allow also tradeoff between time and space.

Chapter 11 extends the theory of tractable constraint probem by looking at special constraints. The chapter will present a general theory for identifying the tractability of constraint problems based on the constraint themselves using algebraic closure properties. These includes implicational and max-ordered constraints. Additional tractable classes exploit notions such as tight domains and tight constraints row-convex constraints, as well as causal networks.

In Chapter 12, special classes of constraints associated with temporal reasoning, that has received much attention in the last decade, will be introduced. These tractable classes include subsets of what is known as the *qualitative interval algebra* expressing relationships such as time interval A overlaps or precedes time interval B, as well as quantitative binary linear inequalities over the Real numbers of the form $X - Y \leq a$.

Chapter 13 extends the main task of constraint processing to general combinatorial optimization. It is often the case that the problem at hand induces preferences among solutions. Such problems can be expressed by augmenting the constraint problem with a cost function. The chapter extends both search, inference, complete and incomplete approaches to optimization. It summarizes the two most well known approaches for combinatorial optimization developed in Operation Research: Branch and Bound and dynamic programming. The chapter introduces these algorithms from the constraint framework perspective and contrast methods such as integer programming with their constraint methods counterparts.

An important aspect to constraint networks, that of modeling a given problem within this framework, is addressed to a limited extent only, in this book via sporadic examples and exrcises. One reason is that this aspect is somewhat an art and is not formally well understood. Chapter 14, the final chapter, provides a window towards a language approach for addressing the modeling issue. It provides an introduction a class of languages that exploit constraint processing algorithms known as Constraint logic programming.

1.2.1 Reading through the book

Some chapters of this book are more advanced and technically demanding than others. We will denote by an asterisk those sections and chapters that can be skipped without disturbing the flow of the book.

1.3 Mathematical background

The formalization of constraint networks relies upon concepts drawn from the related areas of discrete mathematics, logic, the theory of relational databases, and graph theory. This section is a compilation of the mathematical knowledge needed for understanding the formalization of constraint networks and the analyses presented in subsequent chapters. Here we present the basic notations and definitions for sets, relations, operations on relations, and graphs. For those readers already familiar with these topics, a skim of this material will suffice to ensure your understanding of notation used in this book.

1.3.1 Sets, domains, and tuples

A set is a collection of distinguishable objects, and an object in the collection is called a *member* or an *element* of the set. A set cannot contain the same object more than once, and its elements are not ordered. If an object x is a member of set A, we write $x \in A$; if an object x is not a member of set A, we write $x \notin A$. A set can be defined explicitly by listing the members of the set or implicitly by stating a property satisfied by elements of the set. For example, $A = \{1, 2, 3\}$ and $A = \{x \mid x \text{ an integer and } 1 \leq x \leq 3\}$ both represent the same set with three members, 1, 2, and 3. If each element of a set A is also an element of set B, then we write $A \subseteq B$ and say that A is *subset* of B. Set A is a *proper subset* of B, written $A \subset B$, if $A \subseteq B$ but $A \neq B$.

Given two sets A and B, we can also define new sets by applying set operations: the *intersection* of two sets A and B is the set $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$, the *union* of two sets A and B is the set $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$, and the *difference* of two sets A and B is the set $A - B = \{x \mid x \in A \text{ and } x \notin B\}$. A set containing no members is called an *empty set* and is denoted $\emptyset$. The number of elements in a set S is called the *size* (or *cardinality*) of the set and is denoted $|S|$. Two sets A and B are *disjoint* if they have no elements in common; that is, if $A \cap B = \emptyset$.

The *domain* of a variable is simply a set that lists all of the possible objects that a variable can denote or all of the possible values that a variable can be assigned. A *k-tuple* (or simply a *tuple*) is a sequence of k not necessarily distinct objects denoted by $(a_1, \ldots, a_k)$, and an object in the sequence is called a *component*. The *Cartesian product* (or simply the *product*) of a list of domains $D_1, \ldots, D_k$, written $D_1 \times \cdots \times D_k$, is the set of all k-tuples $(a_1, \ldots, a_k)$ such that a_1 is in D_1, a_2 is in D_2, and so on.

Example 1.3.1 Let D_1 = {black, green} and D_2 = {apple juice, coffee, tea}. The Cartesian product $D_1 \times D_2$ is the set of tuples {(black, apple juice), (black, coffee), (black, tea), (green, apple juice), (green, coffee), (green, tea)}. □

1.3.2 Relations

Given a set of variables $X = \{x_1, \ldots, x_k\}$, each associated with a domain $D_1, \ldots, D_k$ respectively, a *relation* R on the set of variables is any subset of the Cartesian product of their domains. The set of variables on which a relation is defined is called the *scope* of the relation, denoted $scope(R)$. Each relation that is a subset of some product $D_1 \times \cdots \times D_k$ of k domains is said to have *arity* k. If $k = 1$, 2, or 3, then the relation is called a *unary*, *binary*, or *ternary* relation, respectively. If $R = D_1 \times \cdots \times D_k$, then R is called a *universal* relation. We will frequently denote a relation defined on a scope S by R_S.

Example 1.3.2 Let D_1 = {black, green} be the domain of variable x_1 and let D_2 = {apple juice, coffee, tea} be the domain of variable x_2. The set of tuples {(black, coffee), (black, tea), (green, tea)} is a relation on $\{x_1, x_2\}$, since the tuples are a subset of the product of D_1 and D_2. The scope of this relation is $\{x_1, x_2\}$. □

The empty set is another example of a relation. The scope of the empty relation is itself the empty set.

Representing relations

Relations are sets of tuples defined over the same scope, and, as discussed above for sets, they may be defined either explicitly or implicitly. For example, let R be a relation on the set of variables $\{x_1, x_2\}$, where D_1 = {black, green} and D_2 = {apple juice, coffee, tea}. Then the relation R_1 on the scope $\{x_1, x_2\}$ given by R_1 = {(black, coffee), (black, tea), (green, tea)} and $R_2 = \{(x_1, x_2) \mid x_1 \in D_1, x_2 \in D_2$, and x_1 is before x_2 in dictionary ordering$\}$ both represent the same relation. Using arithmetic expressions we can also write $x_1 \leq x_2$ more succintlly, where $\leq$ is a lexicographic ordering in this case.

Two additional ways to express a relation explicitly make use of tables and (0,1)-matrices. In a table representation, each row is a tuple, and each column corresponds to one component of the tuple. Each column is identified by the variable associated with that component (in the database community, the names of columns are called attributes). The ordering of the columns is inconsequential; two relations that differ only in the ordering of their columns are considered to be the same relation.

In the (0,1)-matrix representation, a binary relation between two variables x_i and x_j is represented as a (0,1)-matrix with $|D_i|$ rows and $|D_j|$ columns by imposing an ordering on the domains of the variables. A 0-entry at row a column b means that the pair consisting of the ath element of D_i and the bth element of D_j is not permitted; a 1-entry means that the pair is permitted. The (0,1)-matrix representation can be generalized to nonbinary relations by increasing the number of dimensions of the matrix.

Figure 1.6 shows two alternative representations of the binary relation R on the scope $\{x_1, x_2\}$, where $D_1 = \{\text{black, green}\}$ and $D_2 = \{\text{apple juice, coffee, tea}\}$.

x_1	x_2
black	coffee
black	tea
green	tea

(a) table

$$\begin{array}{cc} & \underline{x_2} \\ & \begin{array}{ccc} \text{apple juice} & \text{coffee} & \text{tea} \end{array} \\ \underline{x_1} \begin{array}{c} \text{black} \\ \text{green} \end{array} & \left[\begin{array}{ccc} 0 & 1 & 1 \\ 0 & 0 & 1 \end{array}\right] \end{array}$$

(b) (0,1)-matrix

Figure 1.6: Two alternative views of relation $R = \{(\text{black, coffee}), (\text{black, tea}), (\text{green, tea})\}$.

We often use special languages to express languages. One such language is applied to Boolean domains, another is Numerical constraints using mathematical algebraic expressions.

Operations on relations

Having introduced the mathematical notion of a relation, we can now consider operations on relations. First we consider how general set operations apply to relations, and then we discuss three operations specific to relations: selection, projection, and join.

Intersection, union, and difference. Given two relations, R and R', on the same scope, the intersection of R and R', denoted $R \cap R'$, is the relation containing all tuples that are in both R and R'; the union $R \cup R'$ is the relation containing all the tuples that are in either R or R' or both, and the difference $R - R'$ is the relation containing those tuples that are in R but not in R'. The scope of the resulting relations is the same as the scope of the relations R and R'.

Example 1.3.3 Let the relations R, R', and R'' be as shown in Figure 1.7. The relations R and R' have the same scopes $\{x_1, x_2, x_3\}$, so the set operations intersection, union,

x_1	x_2	x_3
a	b	c
b	b	c
c	b	c
c	b	s

(a) Relation R

x_1	x_2	x_3
b	b	c
c	b	c
c	n	n

(b) Relation R'

x_2	x_3	x_4
a	a	1
b	c	2
b	c	3

(c) Relation R''

Figure 1.7: Three relations.

and difference are well-defined for these sets. Since the scope of R'' is not the same, none of these three operations is well-defined for R'' and either of the other two relations. Figure 1.8 shows the relations $R \cap R'$, $R \cup R'$, and $R - R'$. □

x_1	x_2	x_3
b	b	c
c	b	c

(a) $R \cap R'$

x_1	x_2	x_3
a	b	c
b	b	c
c	b	c
c	b	s
c	n	n

(b) $R \cup R'$

x_1	x_2	x_3
a	b	c
c	b	s

(b) $R - R'$

Figure 1.8: Example of set operations intersection, union, and difference applied to relations.

Selection. Let us now consider those operations specific to relations. A selection takes a relation R and yields a new relation that is the subset of tuples of R with specified values on specified variables. That is to say, in a table representation of a relation, selection chooses a subset of the rows of a relation. Let R be a relation, let $x_1, \ldots, x_k$ be variables in the scope of R, and let a_i be an element of D_i, the domain of x_i. We use the notation $\sigma_{x_1=a_1,\ldots,x_k=a_k}(R)$ to denote the selection of those tuples in R that have the value a_1 for variable x_1, the value a_2 for variable x_2, and so on. An alternative and more succinct notation is of the form: if $Y = \{x_1, \ldots, x_k\}$ and $t = (a_1, \ldots, a_k)$, then $\sigma_{Y=t}(R)$ is equivalent to $\sigma_{x_1=a_1,\ldots,x_k=a_k}(R)$. The scope of the resulting relation is the same as the scope of R.

Projection. Projection takes a relation R and yields a new relation that consists of the tuples of R with certain components removed. In the table representation of a relation, projection chooses a subset of the columns of a relation. Let R be a relation, and let

$Y = \{x_1, \ldots, x_k\}$ be a subset of the variables in the scope of R. We use the notation $\pi_Y(R)$ to denote the projection of R onto Y, that is, the set of tuples obtained by taking in turn each tuple in R and forming from it a new, smaller tuple, keeping only those components associated with variables in Y. Projection specifies a subset of the variables of a relation, and the scope of the resulting relation is that subset of variables.

Join. The join operator takes two relations R_S and R_T and yields a new relation that consists of the tuples of R_S and R_T combined on all their common variables in S and T. Let R_S be a relation with scope S and R_T be a relation with scope T. A tuple r is in the join of R_S and R_T, denoted $R_S \bowtie R_T$, if it can be constructed according to the following steps: (i) take a tuple s from R_S, (ii) select a tuple t from R_T such that the components of s and t agree on the variables that R_S and R_T have in common (that is, on the variables in $S \cap T$), and (iii) form a new tuple r by combining the components of s and t, keeping only one copy of those components corresponding to variables in $S \cap T$. The scope of the resulting relation is the union of the scopes of R and S, that is, $S \cup T$. We can see that a join of two relations with the same scopes is equivalent to the intersection of the two relations.

Example 1.3.4 Let the relations R, R', and R'' be as shown in Figure 1.7. Figure 1.9 shows examples of the selection, projection, and join operations applied to these relations. The relation $\sigma_{x_3=c}(R')$ consists of those tuples in R' that have the value c for variable x_3. The relation $\pi_{\{x_2,x_3\}}(R')$ consists of the tuples in R', each with the component that corresponds to the variable x_1 removed; only the components that correspond to variables x_2 and x_3 are kept. Duplicate entries are removed, since a relation is a set and sets do not contain duplicate objects. The relation $R' \bowtie R''$ consists of tuples that are combinations of pairs of tuples from R' and R'' which agree on their common variables $\{x_2, x_3\}$. To construct $R' \bowtie R''$, we consider each tuple in R', match it up with all possible tuples in R'' that agree with it on the common variables, and delete duplicate components associated with these variables. For example, the tuple (b, b, c) in R' agrees with the tuples (b, c, 2) and (b, c, 3) in R'', resulting in the tuples (b, b, c, 2) and (b, b, c, 3). Similarly, the tuple (c, b, c) in R' results in the tuples (c, b, c, 2) and (c, b, c, 3). The tuple (c, n, n) in R' does not agree with any tuple in R'' on variables x_2 and x_3, so no additional tuples are added to $R' \bowtie R''$. □

1.3.3 Graphs: general concepts

Definition: A *graph* $G = (V, E)$ is a structure which consists of a finite set of *vertices*, or *nodes*, $V = \{v_1, ..., v_n\}$ and a set of *edges*, or *arcs*, $E = \{e_1, e_2, ...e_l\}$. Each edge e is incident to an unordered pair of vertices $\{u, v\}$ which are not necessarily distinct (as in

x_1	x_2	x_3
b	b	c
c	b	c

(a) $\sigma_{x_3=c}(R')$

x_2	x_3
b	c
n	n

(b) $\pi_{\{x_2,x_3\}}(R')$

x_1	x_2	x_3	x_4
b	b	c	2
b	b	c	3
c	b	c	2
c	b	c	3

(c) $R' \bowtie R''$

Figure 1.9: Example of selection, projection, and join operations on relations.

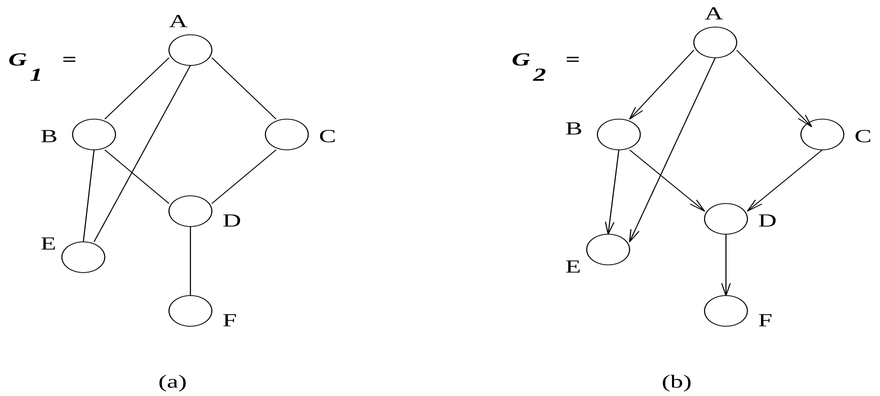


Figure 1.10: Two graphs: (a) undirected and (b) directed.

the case of a loop). Although the vertices are unordered, they will often be written as an ordered pair (u,v). If $e = (u,v) \in E$ we say that e connects u and v and that u and v are *adjacent* or *neighbors*. The degree $d(u)$ of a vertex u in a graph is the number of its adjacent vertices.

A *path* is a sequence of edges $e_1, e_2, ..., e_k$ such that e_i and e_{i+1} share an endpoint. Namely, if $e_i = (u_1, v_1)$ and $e_{i+1} = (u_2, v_2)$, then v_1 and u_2 are the same. It is also convenient to describe a path using its vertices $v_0, v_1, ...v_k$, where $e_i = (v_{i-1}, v_i)$. Node v_o is called the start-vertex of the path, v_k is called the end vertex, and the length of the path is k. A *cycle* is a path whose start and end vertices are the same. A path is called *simple* if no vertex appears on it more than once. A cycle is called *simple* if no vertex other than the start-end vertex appears more than once and the start-end vertex does not appear elsewhere in the cycle. If for every two vertices u and v in the graph there exists a path from u to v. then the graph is said to be *connected*. An undirected graph with no cycles is called a *tree*. Given a subset of the nodes S in the graph G, a *subgraph* relative to S, denoted G_S is the graph whose nodes are in S and whose edges are all edges in G that are incident only to nodes in S. A graph is *complete* if every two nodes are adjacent.

A *clique* in a graph is a complete subgraph.

A *directed graph (digraph)* is defined similarly to an undirected graph except that the pair of endpoints of an edge is now ordered; the first endpoint is called the *start-vertex* of the edge, and the second is called the *end-vertex*. The edge $e = (u, v)$, also denoted $u \rightarrow v$, is said to be directed from u to v. The *outdegree* of a vertex v is the number of edges which have v as their start-vertex; *in-degree* of v is the number of edges which have v as their end-vertex. The set of nodes that point to node u is called its *parents* and is denoted $pa(u)$. Similarly, the set of vertices to which u points is called the set of *child nodes* of u and is denoted $ch(u)$. A *directed path* is a sequence of edges $e_1, e_2, ..., e_k$ such that the end-vertex of e_{i-1} is the start-vertex of e_i. A directed path is a *a directed cycle* if the start-vertex of the path is the same as the end-vertex. A directed graph is *strongly connected* if for every vertex u and every vertex v there is a directed path from u to v. A directed graph is *acyclic* if it has no directed cycles. The following example demonstrates the above definitions.

Example 1.3.5 The graph G_1 in Figure 1.10a is an undirected graph over vertices $\{A, B, C, D, E, F\}$. The edge $e = (A, B)$ is in the graph while (B, C) is not. The sequence (A, B, D, F) is a path whose start-vertex is A and whose end-vertex is F. The path (A, B, D, C, A) is a simple cycle. The degree of vertex D is 3. The subgraph $\{A, B, E\}$ is a clique. The subgraph $\{A, B, E, D\}$ contains four edges: $\{(A, B), (B, E), (A, E), (B, D)\}$. The graph G_2 in Figure 1.10b is a directed graph which is acyclic. The indegree of D is 2, and its outdegree is 1. In other words, D in G_2 has two parents and one child node. □

1.3.4 Background in complexity

[to be written]

1.4 Chapter notes

Section 1.2 already provides a breif overview of the constraint area. The area of constraint started with the work of Waltz [156] in vision for identifying 3-dimentional interpretation from 2-dimentional drawings It was followd by the seminal work of Montanari introducing the concept of Constraint networks [112] formally, establishing many of the concepts that became the building blocks of this area such as path-consistency. Machworth [100] further extended and popularized this work defining the notion of local node, arc and path consistency and discuss the implication of such methods on search. There are several surveys on aspects of constraint-based reasoning including Mackworth [101], Dechter [36], Kumar [89]. There is also an alternative full-length book treatment of the subject by

Tsang [143] and a new book on constraint programming [105]. For surveys on constraint programming see [72]. More recent surveys can be found in [39].

For mathematical foundation see Cormen, Leiserson, and Rivest [25] is a useful reference for some of the elements of discrete mathematics discussed in this chapter and for material on asymptotic notation and analyzing algorithms, which is used but not presented in this book. There is a close relationship between constraint networks and the relational data model in database systems, as both use relations as the primary notation for representing data or knowledge. Maier [104] and Ullman [144] are useful references for relational databases and the mathematical language for expressing queries to the database called the relational algebra, from which our operations on relations are drawn.

A parallel significant development is *Constraint programing.* These are programming languages that address modeling issues focusing on various languages for expressing constraints. Such languages allowed the practical solution of many constraint satisfaction algorithms [105].

1.5 Exercises

1. Let $R_{xy} = \{(a,b),(c,d),(d,e)\}$ and $R_{yz} = \{(b,c),(e,a),(b,d)\}$. Compute:
   (a) $R_1 \cup R_2$,
   (b) $R_1 - R_2$,
   (c) $R_{xy} \bowtie R_{yz}$,
   (d) $\pi_x R_{xy}$
   (e) $\sigma_{x=c}(R_{xy} \times R_{yz})$,

Chapter 2

Constraint Networks

In this chapter we will begin the process of formally modeling constraint satisfaction problems as constraint networks. The initial formal work on constraint networks introduced in the paper by Montanari [112] was restricted to binary constraints, defined on pairs of variables only. Also, some of the beginning work and especially experiments was limited to the binary case. Indeed it is possible to show that any set of constraints can be mapped to the binary case. Nevertheless the book will always assume the general case when constraints are defined on arbitrary size sets of variables and will refer to the special case of binary constraints explicitly, when appropriate.

2.1 Constraint networks and constraint satisfaction

2.1.1 The basics of the framework

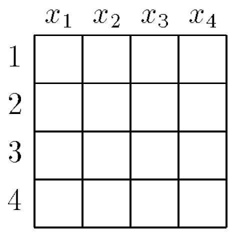
A *constraint network* $\mathcal{R}$ consists of a finite set of *variables* $X = \{x_1, \ldots, x_n\}$, with respective *domains* $D = \{D_1, \ldots, D_n\}$ which list the possible values for each variable $D_i = \{v_1, \ldots, v_k\}$, and a set of *constraints* $C = \{C_1, \ldots, C_t\}$. Thus a constraint network can be viewed as a triple (X, D, C).

A constraint C_i is a relation R_i defined on a subset of variables S_i, $S_i \subseteq X$, which denotes their simultaneous legal value assignments. S_i is called the *scope* of R_i. If $S_i = \{x_{i_1}, ...x_{i_r}\}$, then R_i is a subset of the Cartesian product $D_{i_1} \times ... \times D_{i_r}$. Thus a constraint can also be viewed as a pair $C_i =< S_i, R_i >$. When the scopes identity is clear, we will often identify the constraint C_i with its relation R_i. Otherwise, for clarity a constraint relation may be denoted R_{S_i}. A scope will be denoted explicitly as $\{x, y, z\}$, or, when no confusion can arise, as xyz. For example, we can denote by R_{xyz}, (or by $< xyz, R >$) the constraint defined over the variables x, y, and z whose relation is R. The set of scopes $S = \{S_1, \ldots, S_i, \ldots, S_t\}$ is called the *scheme* of the network. Without loss of generality, we assume that only a single constraint is defined over a subset S_i in S, namely if $i \neq j$, then $S_i \neq S_j$.

The *arity* of a constraint refers to the cardinality, or size, of its scope. A *unary constraint* is defined on a single variable. A *binary constraint* is defined on two variables. A *binary constraint network* is one that has only unary and binary constraints.

Formulating the n-queen problem

Consider a possible constraint network formulation of the n-queens problem. The columns of the chess board will be the variables, $x_1, \ldots, x_n$, and the domains of the variables are the possible row positions, $D_i = \{1, \ldots, n\}$. Assigning a value $j \in D_i$ to a variable x_i corresponds to placing a queen in row j on column x_i of the chess-board. The constraints are binary and state that no two queens should attack one another, namely that no two queens should be placed on the same row or diagonal. Figure 2.1(a) shows the chess-board for the 4-queens problem, and Figure 2.1(b) shows all of the constraints in relational form. For example, the tuple (1,3) is in the relation R_{12} defined over $\{x_1, x_2\}$, which means that simultaneously assigning the value 1 to variable x_1 and the value 3 to variable x_2 is allowed by the constraint. The complete definition of the 4-queen $\mathcal{R} = (X, D, C)$, where $X = \{x_1, x_2, x_3, x_4\}$, and for every i $D_i = \{1, 2, 3, 4\}$. There are 6 constraints, $C_1 = R_{12}$, $C_2 = R_{13}$, $C_3 = R_{14}$, $C_4 = R_{23}$, $C_5 = R_{24}$ and $C_6 = R_{34}$.



(a)

$$
\begin{aligned}
R_{12} &= \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\} \\
R_{13} &= \{(1,2), (1,4), (2,1), (2,3), (3,2), (3,4), (4,1), (4,3)\} \\
R_{14} &= \{(1,2), (1,3), (2,1), (2,3), (2,4), (3,1), (3,2), (3,4) \\
&\qquad (4,2), (4,3)\} \\
R_{23} &= \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\} \\
R_{24} &= \{(1,2), (1,4), (2,1), (2,3), (3,2), (3,4), (4,1), (4,3)\} \\
R_{34} &= \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}
\end{aligned}
$$

(b)

Figure 2.1: The 4-queens constraint network. The network has four variables, all with domains $D_i = \{1, 2, 3, 4\}$. (a) The labeled chess board. (b) The constraints between variables.

2.1.2 Solution of a constraint network

Instantiation. When a variable is assigned a value from its domain, we say that the variable has been *instantiated*. An *instantiation* of a subset of variables is an assignment from its domain to each variable in the subset. Formally an instantiation of a set of variables $\{x_{i_1}, \ldots, x_{i_k}\}$ is a tuple of ordered pairs $(< x_{i_1}, a_{i_1} >, \ldots, < x_{i_k}, a_{i_k} >)$, where each pair $< x, a >$ represents an assignment of the value a to the variable x, where a is in the domain of x. We also use the notation $(x_1 = a_1 \ldots, x_i = a_i)$. For simplicity, however we often abbreviate $(< x_1, a_1 >, \ldots, < x_i, a_i >)$ to $\bar{a} = (a_1, \ldots, a_i)$, and perceive an intantiation or a tuple as a relation over some scope having a single tuple.

Satisfying a constraint. An instantiation or a tuple $\bar{a}$, satisfies a constraint R iff the components of the tuple $\bar{a}$ associated with R's scope are in the relation R. For example, let $R_{xyz} = \{(1,1,1), (1,0,1), (0,0,0\}$. then the instantiation $\bar{a}$ whose scope is $\{x, y, z, t\}$ given by $\bar{a} = (x = 1, y = 1, x = 1, t = 0)$, satisfies R_{xyz} because its projection on $\{x, y, z\}$ is $(1,1,1)$ which is an element of R. However, the instantiation $(x = 1, y = 0, x = 0, t = 0)$, violates R.

Solution. A *solution* of a constraint network $\mathcal{R} = (X, D, C)$ where $X = \{x_1, ..., x_n\}$, is an instantiation of all its variables such that all the constraints are satisfied. A projection of a tuple $\bar{a}$ on a subset of its scope S is denoted also by $\bar{a}[S_i]$ (as well as $\pi_{S_i}(\bar{a})$). The solution relation $sol(\mathcal{R})$, also denoted ρ_X, is defined by:

$$sol(\mathcal{R}) = \{\bar{a} = (a_1, ..., a_n) | a_i \in D_i, \forall S_i \in \text{ scheme of } R, \bar{a}[S_i] \in R_i\}.$$

A network of constraints is said to *express* or *represent* the relation of all its solutions.

A consistent partial instantiation. A partial instantiation is consistent if it satisfies all of the constraints that have no uninstantiated variables. More formally, let S be a subset of variables, and let $\bar{a}$ be an instantiation over S. $\bar{a}$ is *consistent* relative to network $\mathcal{R}$ if and only if for all S_i in the scheme of $\mathcal{R}$, s.t., $S_i \subseteq S$, $\bar{a}[S_i] \in R_{S_i}$. If we have a constraint network $\mathcal{R}$ over X and a subset of variables $A \subseteq X$, $sol(A)$ or ρ_A is the set of all consistent instantiations over A.

Consider again the constraint network for the 4-queens problem. There is a constraint between every pair of variables, hence the scheme of the network is given by $\{\{x_1, x_2\}, \{x_1, x_3\}, \ldots, \{x_3, x_4\}\}$. Consider the set of variables $Y = \{x_1, x_2, x_3\}$. The instantiation $\bar{a} = (< x_1, 1 >, < x_2, 4 >, < x_3, 2 >)$ shown in Figure 2.2(a) is consistent, since

$$\begin{array}{lcl}
\bar{a}[\{x_1, x_2\}] = (1,4) & \text{and} & (1,4) \in R_{12} \\
\bar{a}[\{x_1, x_3\}] = (1,2) & \text{and} & (1,2) \in R_{13} \\
\bar{a}[\{x_2, x_3\}] = (4,2) & \text{and} & (4,2) \in R_{23}
\end{array}$$

although $\bar{a}$ is not part of any full solution. There are only two full solutions to the 4-queens problem, as shown in Figure 2.2(b) and (c). The 4-queen problem represents the relation $\rho_{1234} = \{(2,4,1,3)(3,1,4,2)\}$.

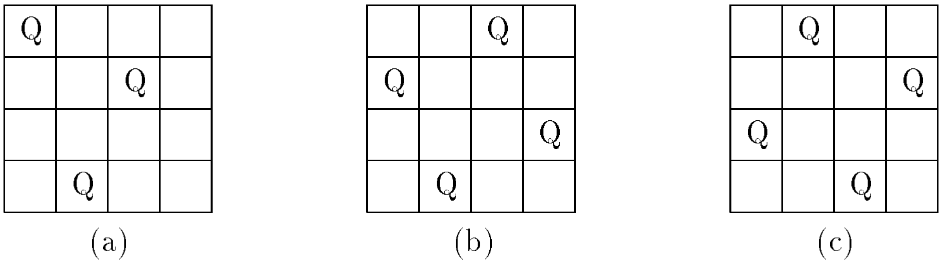


Figure 2.2: Not all consistent instantiations are part of a solution: (a) A consistent instantiation that is not part of a solution. (b) The placement of the queens corresponding to the solution $(2,4,1,3)$. (c) The placement of the queens corresponding to the solution $(3,1,4,2)$.

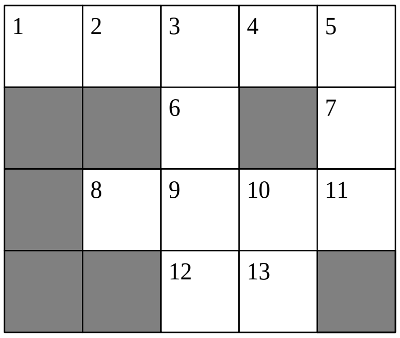


Figure 2.3: A crossword puzzle

The crossword puzzle

We return now to our crossword puzzle example from Chapter 1. (See again Figure 2.3.) A formal constraint network formulation is as follows: there is a variable for each square that can hold a character, $x_1, \ldots, x_{13}$, the domains of the variables are the alphabet letters, and the constraints are the possible words. For this example, the constraints are given by:

$$
\begin{aligned}
R_{1,2,3,4,5} &= \{(\mathrm{H,O,S,E,S}),\ (\mathrm{L,A,S,E,R}),\ (\mathrm{S,H,E,E,T}),\ (\mathrm{S,N,A,I,L}),\ (\mathrm{S,T,E,E,R})\} \\
R_{3,6,9,12} &= \{(\mathrm{A,L,S,O}),\ (\mathrm{E,A,R,N}),(\mathrm{H,I,K,E}),\ (\mathrm{I,R,O,N}),\ (\mathrm{S,A,M,E})\} \\
R_{5,7,11} &= \{(\mathrm{E,A,T}),\ (\mathrm{L,E,T}),\ (\mathrm{R,U,N}),\ (\mathrm{S,U,N}),\ (\mathrm{T,E,N}),\ (\mathrm{Y,E,S})\} \\
R_{8,9,10,11} &= R_{3,6,9,12} \\
R_{10,13} &= \{(\mathrm{B,E}),\ (\mathrm{I,T}),\ (\mathrm{N,O}),\ (\mathrm{U,S})\} \\
R_{12,13} &= R_{10,13}.
\end{aligned}
$$

The reader should verify that the consistent partial assignments over the set of variables $\{x_1, x_2, x_3, x_4, x_5, x_6, x_9, x_{12}\}$ are:

$$\rho_{1,2,3,4,5,6,9,12} = \{(H, O, S, E, S, A, M, E), (L, A, S, E, R, A, M, E),$$

$$(S, H, E, E, T, A, R, N), (S, N, A, I, L, L, S, O), (S, T, E, E, R, A, R, N)\}.$$

An alternative constraint formulation of the crossword puzzle associates each first digit that can start a word as a variable and its possible word assignments as values. In this case, the variables are x_1 (1, horizontal), x_2 (3, vertical), x_3 (5, vertical), x_4 (8, horizontal), x_5 (12, horizontal), and x_6 (10, vertical). The scheme of this problem is

$$\{\{x_1, x_2\}, \{x_1, x_3\}, \{x_4, x_2\}, \{x_4, x_3\}, \{x_5, x_2\}, \{x_6, x_4\}, \{x_6, x_5\}\}$$

because there is a constraint between x_1 (1, horizontal) and x_2 (3, vertical), and so on. The domains are: $D_1 = \{hoses, laser, sheet, snail, steer\}$, $D_2 = D_4 = \{also, earn, hike, iron, same\}$, $D_3 = \{eat, let, run, sun, ten, yes\}$ and $D_5 = D_6 = \{be, it, no, us\}$. The constraint between x_1 and x_2, is given by

$$R_{12} = \{(hoses, same)(laser, same), (sheet, earn)(snail, also), (steer, earn)\}.$$

A partial consistent tuple in this case is : $(x_1 = sheet, x_2 = earn, x_3 = ten, x_4 = iron, x_5 = no)$. Note that this formulation of the problem is binary, namely, involving constraints on pairs of variables only. Also note that the network has no solutions.

2.1.3 Constraint graphs

A constraint network can be represented by a graph called a (primal) *constraint graph*. Each node represents a variable, and the arcs connect all nodes whose variables are included in a constraint scope of the problem's scheme. The lack of an arc between two nodes indicates that the direct constraint, the one specified in the input, between the two is the *universal relation* allowing every tuple. Figure 2.4(a) shows the constraint graph associated with our second formulation of the crossword puzzle, and Figure 2.4(b) shows the constraint graph of the 4-queens problem, which is complete.

A scheduling example

The constraint framework is useful for expressing and solving scheduling problems. Consider the problem of scheduling five tasks, T1, T2, T3, T4, T5, each of which takes one hour to complete. The tasks may start at 1:00, 2:00, or 3:00. Any number of tasks can

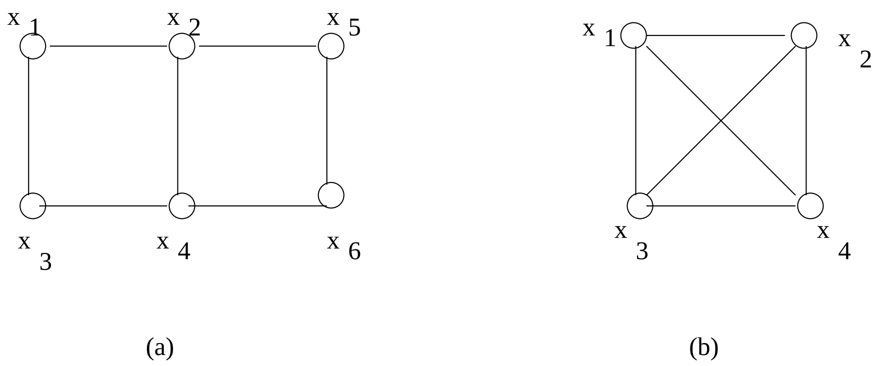


Figure 2.4: Constraint graphs of (a) the crossword puzzle and (b) the 4-queens problem.

be executed simultaneously, subject to the restrictions that T1 must start after T3, T3 must start before T4 and after T5, T2 cannot execute at the same time as T1 or T4, and T4 cannot start at 2:00.

With five tasks and three time slots, we can model the scheduling problem by creating five variables, one for each task, and giving each variable the domain {1:00, 2:00, 3:00}. Another possible approach is to create three variables, one for each starting time, and to give each of these variables a domain which is the powerset of {T1, T2, T3, T4, T5}. This requires a tricky means of expressing the constraints and is left as an exercise at the end of the chapter. Adopting the first approach, the problem's constraint graph is shown in Figure 2.5. The constraint relations are shown on the right of the figure.

The (primal) constraint graph is well defined for both binary and non-binary constraints. However, a hypergraph representation maintains the association between arcs and constraints in the non-binary case more accurately.

Definition 2.1.1 (hypergraphs) *A hypergraph is a structure* $\mathcal{H} = (V, S)$ *that consists of vertices* $V = \{v_1, .., v_n\}$ *and a set of subsets of these vertices* $S = \{S_1, ..., S_l\}$, $S_i \subseteq V$, *called hyperedges. The hyperedges differ from regular edges in that they "connect" (or are defined over) more than two variables.*

In the *constraint hypergraph* representation of a constraint problem, nodes represent the variables, and *hyperarcs* (drawn as regions) are the scopes of constraints. Namley, they group those variables that belong to the same scope. A related representation is the *dual constraint graph*. A *dual constraint graph* represents each *constraint scope* by a node and associates a labeled arc with any two nodes whose *constraint scopes* share variables. The arcs are labeled by the shared variables. Figure 2.6 depicts the *primal* and the *dual* graphs of our first formulation of the crossword puzzle. Notice that the dual

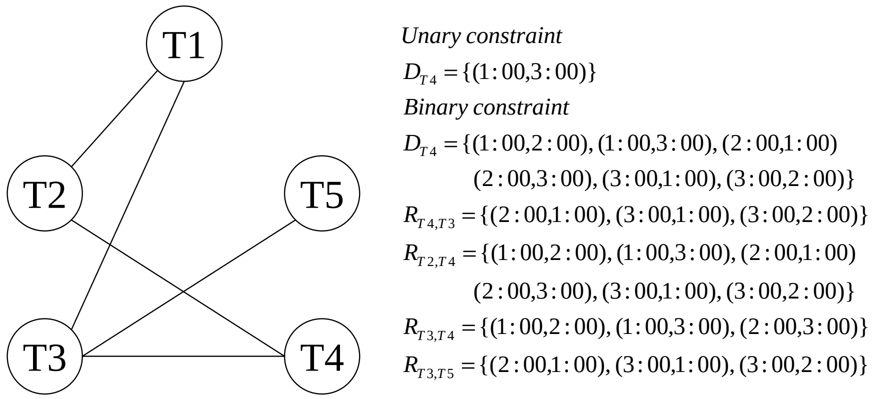


Figure 2.5: The constraint graph and constraint relations of the scheduling problem in Example 1.

graph of the first formulation (Figure 2.6b) is identical to the primal constraint graph of the second formulation (Figure 2.4a). Indeed, the *dual* constraint graph is essentially a

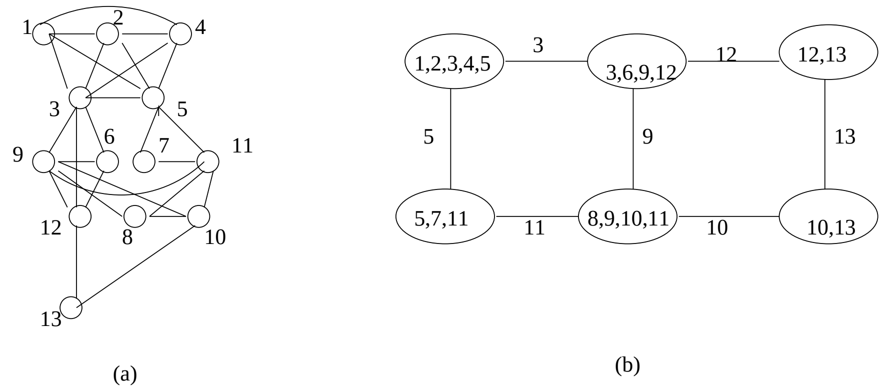


Figure 2.6: Constraint graphs of the crossword problem: (a) primal and (b) dual.

transformation of a nonbinary network into a special type of *binary* network called the *dual problem*, where the constraints are the variables, denoted *c-variables*. The domain of each c-variable ranges over all possible value combinations permitted by the corresponding constraint, and any two adjacent c-variables must obey the restriction that their shared (original) variables must have the same values (i.e., the c-variables are bound by equality constraints). Viewed in this way, any network can be transformed into a binary network and solved by binary network techniques. For the crossword puzzle, both the dual and the primal representations are natural, that is, people frequently will come up with each of these formulations as their first representation of the problem.

The Huffman-Claws labeling

Another natural dual-primal formulation can be demonstrated through the Huffman-Clows junction labellings. In one formulation of the problem as a constraint network, variables denote the various junctions in the problem instance, their domains are the possible label combinations on junction types, and the constraints are expressed using (0,1)-matrices as in Figure 2.8. For example, x_i denotes junction i in the cube. Since 1 is a Fork junction, the values of x_1 are the label combinations that can be assigned to such a junction. The constraints require that if an edge connects two junctions, it will get the same label from each junction. The $(0,1)$-matrix representation of the constraints assumes that the domains of each junction variable (in Figure 2.7) are ordered from left to right. In this formulation, all the constraints are binary and the constraint graph is identical to the input problem graph. The set of solutions for this problem is depicted in Figure 2.9.

This formulation can also be perceived as the dual graph of a nonbinary formulation whereby the variables are the edges, having the domains of $\{+, -, \rightarrow, \leftarrow\}$, and the constraints are binary or ternary depending on the junction types. In the primal constraint graph, two nodes (representing edges) will be connected if they participate in the same junction (see exercises). %endexample

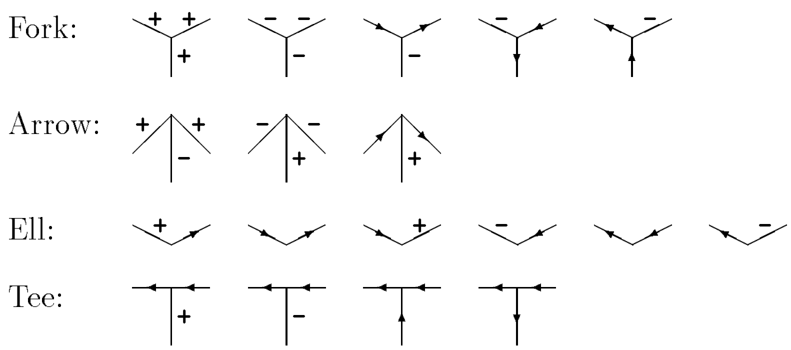
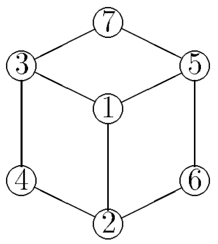


Figure 2.7: Huffman-Clowes junction labelings

2.2 Numeric and Boolean constraints

Relations provide the underlying meaning of a constraint and a very explicit specification syntax of the allowed tuples, as seen in Figure 2.1b. However, this method of specification can sometimes prove very tedious and unwieldy. In many instances, we can utilize

$$R_{21} = \begin{bmatrix} 0\ 1\ 1\ 0\ 0 \\ 1\ 0\ 0\ 0\ 0 \\ 1\ 0\ 0\ 0\ 0 \end{bmatrix} \quad R_{31} = \begin{bmatrix} 0\ 1\ 0\ 1\ 0 \\ 1\ 0\ 0\ 0\ 0 \\ 1\ 0\ 0\ 0\ 0 \end{bmatrix} \quad R_{51} = \begin{bmatrix} 0\ 1\ 0\ 0\ 1 \\ 1\ 0\ 0\ 0\ 0 \\ 1\ 0\ 0\ 0\ 0 \end{bmatrix}$$



$$R_{24} = R_{37} = R_{56} = \begin{bmatrix} 0\ 0\ 1\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0\ 1 \\ 0\ 0\ 0\ 1\ 1\ 0 \end{bmatrix}$$

$$R_{26} = R_{34} = R_{57} = \begin{bmatrix} 1\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1\ 0\ 0 \\ 0\ 0\ 0\ 0\ 1\ 1 \end{bmatrix}$$

Figure 2.8: Scene labeling constraint network

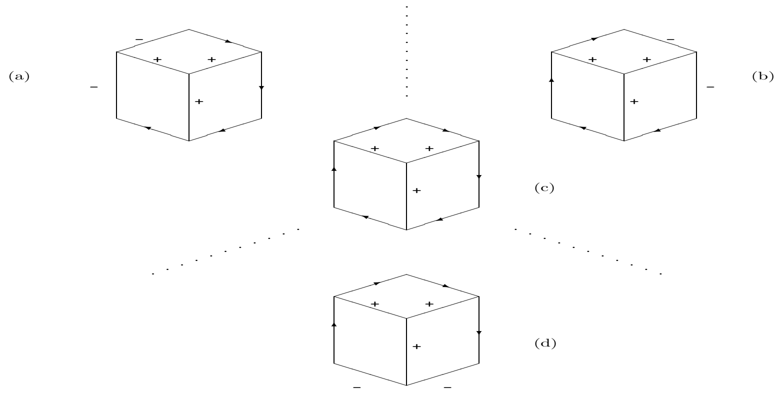


Figure 2.9: Solutions: (a) stuck on left wall, (b) stuck on right wall, (c) suspended in mid-air, (d) resting on floor.

mathematical conventions to create a description of the relationships between objects that is more concise or more convenient than the relational description. Two examples of alternative languages for describing constraints are arithmetic constraints and boolean constraints. The first allows more concise expression while the second, in addition to being restricted to bi-valued domains, expresses explicitly the forbidden tuples rather then the legal ones.

2.2.1 Numeric Constraints

Numeric, or arithmetic, constraints use the language of arithmetic to express the constraints in a network. Consider expressing our 4-queens problem with numeric constraints.

Instead of describing the constraints by enumerating the allowed elements of each relation, we can very succinctly state that every two variables x_i and x_j should satisfy: $\forall i, j, \;\; x_i \neq x_j, \; and \; |x_i - x_j| \neq |i - j|$ defining the relation

$$R_{ij} = \{(x_i, x_j) \mid x_i \in D_i, x_j \in D_j, x_i \neq x_j, \text{ and } |x_i - x_j| \neq |i - j|\}.$$

In another example, let the domains of the variables be finite subsets of the integers, and let a binary constraint between two variables be a conjunction of linear inequalities of the form $ax_i - bx_j = c$, $ax_i - bx_j < c$, or $ax_i - bx_j \leq c$, where a, b, and c are integer constants. For example, the conjunction

$$(3x_i + 2x_j \leq 3) \wedge (-4x_i + 5x_j < 1)$$

is a legitimate constraint between variables x_i and x_j. A network with constraints of this form can be formulated as "an integer linear program" where each constraint is defined over two variables, and the domains of the variables are restricted to being finite subsets of the integers.

Linear constraints are a special subclass of numeric constraints widely applicable in the areas of scheduling and temporal and spatial reasoning. Many techniques for treating these type of constraints were developed in the Operations Research (OR) community.

Cryptaritmetic puzzles

One class of toy problems that can be easily formulated with linear constraints are the *cryptarithmetic puzzles*, such as: SEND + MORE = MONEY. Here we are asked to replace each letter by a different digit so that the above equation is correct. The constraint formulation of these puzzles will associate each letter with a variable whose domains are the digits $\{0..9\}$. The exact formulation of this problem is left as an exercise at the end of the chapter.

2.2.2 Boolean constraints and propositional cnf

When the variables of a constraint problem range over two values, we frequently use a boolean propositional language to describe the various relationships. Assume that you would like to invite your freinds Alex, Beki, and Chris to a party. Let A, B, and C denote the propositions "Alex comes", "Beki comes" and "Chris comes", respectively. Yu know that is Alex comes to the party, Beki will come as well, and if Chris comes, then Alex will too. This can be expressed in propositional calculus as $(A \rightarrow B) \wedge (C \rightarrow A)$, or equivalently as $(\neg A \vee B) \wedge (\neg C \vee A)$, where the disjunctive formulas $(\neg A \vee B)$ and $(\neg C \vee A)$ are called *clauses*. Assume now that Chris came to the party; should you expect to see

Beki? Or, in propositional logic, does the propositional theory $\varphi - C \wedge (A \rightarrow B) \wedge (C \rightarrow A)$ entails B? A common way to answer this query is to assume that Beki will not come and check wether this is a plausible situation (i.e., decide if $\varphi\prime = \varphi \wedge \neg B$ is satisfiable). If $\varphi\prime$ is unsatisfiable, we can conclude that φ entails B. Formally, the propositional satisfiability problem (SAT) is to decide wether a given cnf theory has a *model*, i.e., an assignment to its proposition that does not violate any clause.

Propositional satisfiability can be described as a CSP, where propositions correspond to the variables, domains are $\{0, 1\}$, and constraints are represented by clauses (for example, clause $(\neg A \vee B)$ allows all the tuples (A, B) except $(A = 1, B = 0)$.

Propositional variables take only two values $\{true, false\}$ or "1" and "0." We denote propositional *variables* by uppercase letters $P, Q, R, \ldots$, propositional literals (i.e., $P, \neg P$) stand for $P =$ "$true$" or $P =$ "$false$," and disjunctions of literals, or *clauses*, are denoted by $\alpha, \beta, \ldots$. A *unit clause* is a clause of size 1. The notation $(\alpha \vee T)$, when $\alpha = (P \vee Q \vee R)$ is shorthand for the disjunction $(P \vee Q \vee R \vee T)$. $\alpha \vee \beta$ denotes the clause whose literal appears in either α or β. A formula φ in conjunctive normal form (*cnf*),referred to as a *theory*, is a set of clauses $\varphi = \{\alpha_1, \ldots, \alpha_t\}$ that denotes their conjunction. The set of *models* or *solutions* of a formula φ is the set of all truth assignments to all its variables that do not violate any clause.

In general, a cnf theory is a constraint network whose variables are the propositions, the domains have two values $\{true, false\}$ or $\{0, 1\}$. Each clause is a constraint on the correponding propositional variables. For instance, the cnf theory $\varphi = (A \vee B) \wedge (C \vee \neg B)$ has three variables and two constraints. The constraint $A \vee B$ expresses the relation $R_{AB} = \{(01), (10), (11)\}$.

The structure of a propositional theory can be described by an *interaction graph*. The interaction graph of a propositional theory φ, denoted $G(\varphi)$, is an undirected graph that contains a node for each propositional variable and an edge for each pair of nodes that correspond to variables appearing in the same clause. For example, the interaction graph of theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$ is shown in Figure 10.6a.

2.2.3 Combinatorial circuits diagnosis

Boolean propositional languages are often used to express combinatorial circuits built out of AND, OR, and XOR gates (see Figure **??**). The diagnosis task over such circuits can be formulated as a constraint satisfaction problem. In *circuit diagnosis* we are given a description of the circuit composed of inputs combined through the boolean gates to their output. If the expected output and the observed output are different, the task is to identify a subset of the gates that, if assumed to be faulty, can explain the observed output.

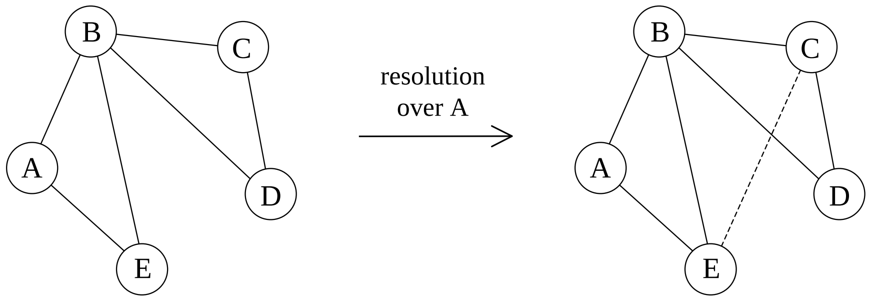


Figure 2.10: The interaction graph of theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$

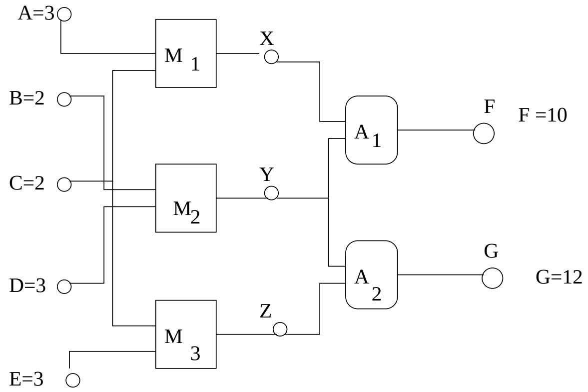


Figure 2.11: A combinatorial circuit: M is a multiplier, A is an adder

One method of formulating the diagnosis task within a constraint network is to associate each gate, as well as its inputs and outputs, with a variable and then to describe by boolean constraints the relationship between the input and output of each gate under the assumption that the gate is either proper or faulty. Once a fault is observed at the output, the task of explaining the circuit behavior is completed by identifying a set of gates that is assumed faulty. (often the minimal such set is required). Such an explanation amounts to finding type of consistent solution for the formulated network.

In the specific example of Figure 2.11 there are three multipliers and two adders. The constraint formulation will have a variable for each input $\{A, B, C, D, E\}$, output $\{F, G\}$ and intermediate output $\{X, Y, Z\}$ and for each component M_1, M_2, M_3, A_1, A_2. The domains of the input and output variables are any number (integer or Boolean if we so choose to restrict the problem). The domain of the components are $\{0, 1\}$, when 1 indicate a faulty behavior of the component and 0 a correct behavior. A constraint will be associated with each component variable, its input variables and output variables, yielding 5 constraints, each defined over 4 variabes.

If the input and outputs are Boolean variables, the constaints can be expressed by a Boolean expression. For example, if M_1 is an AND gate its associated constraint is: $M_1 \rightarrow (A \wedge C \rightarrow X)$. A detailed description of the constraints is requested in the exercises.

2.3 Properties of binary constraint networks

Many of the concepts for constraint processing were initially introduced for binary networks. We find it helpful to discuss binary networks separately in this section since many important concepts of constraint processing are easier to digest in this restricted case. In particular, concepts such as mimimal network and decomposability, allow deep understanding of issues that underly the general theory of constraints.

2.3.1 Equivalence and deduction with constraints

One central concept with which we should be comfortable is constraint deduction or *constraint inference*. New constraints can be inferred from an initial set of constraints. These newly inferred constraints might take the form of constraints between variables that were not initially constrained or might be tightenings of existing constraints. For instance, from the algebraic constraints $x \geq y$ and $y \geq z$ we can infer that $x \geq z$. In particular, it means that adding the inferred constraint $x \geq z$ yields an equivalent constraint network. The next examble demonstrate such notions using relational representation.

Example 2.3.1 Consider the graph-coloring problem in Figure 2.12a. The problem

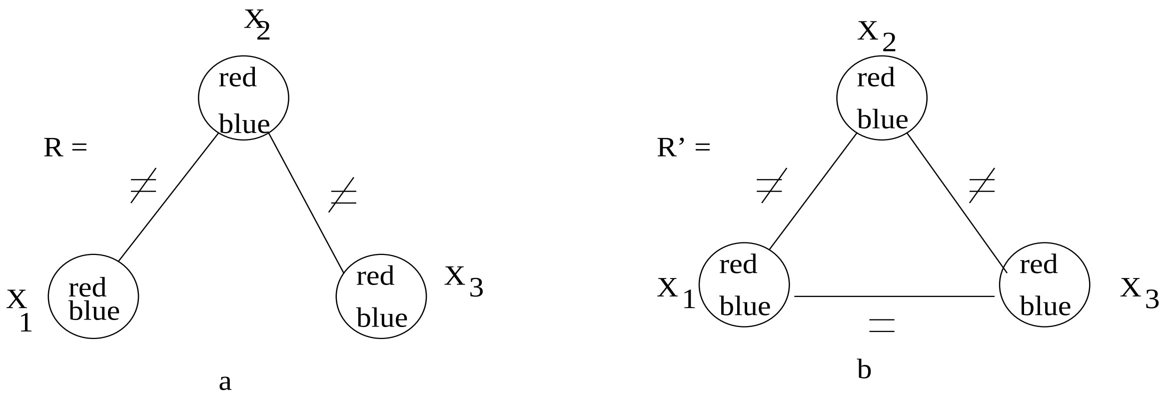


Figure 2.12: (a) A graph $\mathcal{R}$ to be colored by two colors, (b) an equivalent representation $\mathcal{R}'$ having a newly inferred constraint between x_1 and x_3.

can be described by a constraint network with three variables, x_1, x_2, x_3, one for each node, all defined on the same domain values $\{red, blue\}$, and the inequality constraints: $R_{21} = R_{32} = \{(blue, red), (red, blue)\}$. The lack of an arc between x_1 and x_3 represents the universal relation $R_{13} = \{(red, blue), (blue, red), (red, red), (blue, blue)\}$. The problem has two solutions: $\rho_{123} = \{(red, blue, red)(blue, red, blue)\}$. Assume now that we tighten the constraint between x_1 and x_3, disallowing the pair $(< x_1, red >, < x_3, blue >)$. Since this pair does not participate in any of the original problem's solutions, this restriction does not alter the set of solutions. In fact, if we add the constraint $R'_{13} = \{(red, red)(blue, blue)\}$, we get a new constraint network $\mathcal{R}'$ having the same set of solutions (see Figure 2.12b). Indeed, the constraint R'_{13} which enforces $x_1 = x_3$ can be *inferred* from the original network $\mathcal{R}$. The two networks $\mathcal{R}$ and $\mathcal{R}'$ are said to be *equivalent*. □

An inferred constraint can also be viewed as *redundant* relative to a constraint network since its deletion from the network will not change the set of all solutions. For instance, in $\mathcal{R}'$ the inequality constraint between x_1 and x_2 is redundant since its deletion does not change the set of solutions. The same redundancy exists for each of the other two constraints. However, once R'_{12} is removed from $\mathcal{R}'$, the other constraints are no longer redundant. In summary, two constraint networks are *equivalent* if they are defined on the same set of variables and they express the same set of solutions. A constraint R_{ij} is redundant relative to $\mathcal{R}$ iff $\mathcal{R}$ is equivalent to $\mathcal{R} - R_{ij}$.

One type of constraint deduction is accomplished by the *composition* operation.

Definition 2.3.2 (composition) *Given two binary or unary constraints R_{xy} and R_{yz}, the composition $R_{xy} \cdot R_{yz}$ generates the binary relation R_{xz} defined by:*

$$R_{xz} = \{(a,b) | a \in D_x,\ b \in D_z\ , \exists c \in D_y\ s.t.\ (a,c) \in R_{xy}\ and\ (c,b) \in R_{yz}\}$$

An alternative, operational definition is formulated in terms of the join and projection operators:

$$R_{xz} = R_{xy} \cdot R_{yz} = \pi_{\{x,z\}}(R_{xy} \bowtie R_{yz}).$$

In the example in Figure 2.12a we deduce that $R'_{13} = \pi_{\{x_1,x_3\}}(R_{12} \bowtie R_{23}) = \{(red, red), (blue, blue)\}$, thus yielding the equivalent network in Figure 2.12b. The composition operation can be described via boolean matrix multiplication when binary relations are expressed using $(0, 1)$ matrices.

Example 2.3.3 Continuing with our simple graph-coloring example, the two inequality constraints can be expressed as 2×2 matrices having zeros along the main diagonal.

$$R_{12} = \left(\begin{array}{c|cc} & red & blue \\ \hline red & 0 & 1 \\ blue & 1 & 0 \end{array}\right) \qquad R_{23} = \left(\begin{array}{c|cc} & red & blue \\ \hline red & 0 & 1 \\ blue & 1 & 0 \end{array}\right)$$

Multiplying two such matrices yields the following two-dimensional identity matrix:

$$R_{12} \cdot R_{23} = R_{13} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \left(\begin{array}{c|cc} & red & blue \\ \hline red & 1 & 0 \\ blue & 0 & 1 \end{array} \right)$$

It is important to note that composition, in general, is not distributive with respect to intersection, namely: $R_{12} \cdot (R_{23} \cap R'_{23}) \neq (R_{12} \cdot R_{23}) \cap (R_{12} \cdot R'_{23})$, as you will be required to demonstrate in the exercises. □

Example 2.3.4 For example.... □

2.3.2 The minimal and the projection networks

In his seminal paper [112], Montanari considers the expressive power of binary networks. The question is whether an arbitrary relation can be a set of solutions to some underlying binary constraint network having the same set of variables.

In fact, most relations cannot be represented by a collection of binary constraints, since there are many more different relations on n variables than there are networks of binary constraints on n variables. Given n variables each having a domain of size k, cardinality arguments dictate that the number of different relations on n variables is 2^{k^n}, which is far greater than the number of different binary constraint networks, $2^{k^2 n^2}$. The proof of this argument is left as an exercise at the end of the chapter. In those special cases where a relation is expressible by a binary constraint network, such a representation is highly desirable, since it may require less space to specify.

A relation that cannot be expressed by a binary network may still be approximated by one. Let's consider the approximation of a relation by its binary *projection network.*

Definition 2.3.5 (projection network) *The projection network of a relation ρ is obtained by projecting ρ onto each pair of its variables. Formally, if ρ is a relation over $x_1, ..., x_n$, its projection network, $P(\rho)$ is defined by the network $\mathcal{P} = \{P_{ij}\}$,*

$$P_{ij} = \{(a_i, a_j) | \exists t \in \rho \ s.t. \ t[x_i] = a_i, t[x_j] = a_j\}$$

Example 2.3.6 Let $\rho_{123} = \{(1,1,2)(1,2,2)(1,2,1)\}$. The projection network $P(\rho)$ includes the constraints $P_{12} = \{(1,1)(1,2)\}$, $P_{13} = \{(1,2)(1,1)\}$, and $P_{23} = \{(1,2)(2,2)(2,1)\}$. Generating all solutions of $P(\rho)$ yields $sol(P(\rho)) = \{(1,1,2)(1,2,2)(1,2,1)\}$. □

What is the relationship between the original relation ρ and $P(\rho)$? In Example 2.3.6 generating all solutions of $P(\rho)$ yields back the relation ρ. But this cannot be characteristic of the general case, else we would have proven that every relation has a binary network representation by its projection network, and we know this to be false due to the cardinality argument presented earlier. What does hold in general is that the projection network is the best *upper bound network approximation* of a relation. Namely, for every relation ρ, the solution set of $P(\rho)$ contains ρ. Moreover, as we will show, any other upper bound network will express a solution set that includes the projection network's solutions. The following example elaborates.

Example 2.3.7 Consider a slightly different relation ρ:

x_1	x_2	x_3
1	1	2
1	2	2
2	1	3
2	2	2

The projection network $P(\rho)$ has the following constraints: $P_{12} = \{(1,1)(1,2)(2,1)(2,2)\}$, $P_{23} = \{(1,2)(2,2)(1,3)\}$ and $P_{13} = \{(1,2)(2,3)(2,2)\}$. The set of solutions to $P(\rho)$, $sol(P(\rho))$, are:

x_1	x_2	x_3
1	1	2
1	2	2
2	1	2
2	1	3
2	2	2

$\square$

We see that all the tuples of ρ appear in the solution set of $P(\rho)$, while some additional solutions to $P(\rho)$ are not in ρ. In general:

Theorem 2.3.8 *For every relation ρ, $\rho \subseteq sol(P(\rho))$.*

Proof: Let $t \in \rho$. We have to show onlythat $t \in sol(P(\rho))$, namely that it satisfies every binary constraint in $P(\rho)$. This is clearly true since by its definition every pair of values of t was included, by projection in the corresponding constraint of $P(\rho)$. $\square$

Is there another binary network of constraints $\mathcal{R}'$ whose solution set contains ρ but is smaller than $P(\rho)$? Perhaps we might discover such a network by tightening the constraints of $P = P(\rho)$? Let's try to eliminate the superfluous tuple $(2,1,2)$ from $sol(P)$ in Example 2.3.7 by deleting the pair $(2,1)$ from the projection constraint P_{12}. Unfortunately, if we do this, we also exclude the tuple $(2,1,3)$ which is in ρ. And, if we try to exclude $(2,1,2)$ by deleting the pair $(1,2)$ from P_{23} we eliminate $(1,1,2)$ which is also in ρ. Lastly, if we exclude $(2,2)$ from P_{13}, we will eliminate $(2,2,2)$. Indeed, $P(\rho)$ cannot be tightened any further if its solutions are to include all the tuples in ρ. In fact,

Theorem 2.3.9 *The projection network $P(\rho)$ is the tightest upper bound network representation of ρ; there is no binary network $\mathcal{R}'$, s.t. $\rho \subseteq sol(R') \subset sol(P(\rho))$.*

Consequently, if a relation is not expressible by its projection network, it cannot be expressed by any binary network of constraints.

Theorem 9.5.2 shows that the projection network is the most accurate binary network upper bound for a relation. But is it also the tightest explicit description of that relation $sol(P(\rho))$? This question leads us to the notion of a partial order of tightness among binary constraint networks. As we have seen in Figure 2.12a and b, the networks $\mathcal{R}$ and $\mathcal{R}'$ are semantically equivalent since they represent the same set of solutions, yet $\mathcal{R}'$ is tighter than $\mathcal{R}$ using a pairwise comparison of their binary relations.

Definition 2.3.10 *Given two binary networks, $\mathcal{R}'$ and $\mathcal{R}$, on the same set of variables $x_1, ..., x_n$, $\mathcal{R}'$ is* at least as tight as *$\mathcal{R}$ iff for every i and j, $R'_{ij} \subseteq R_{ij}$.*

Obviously, if $\mathcal{R}'$ is tighter than $\mathcal{R}$, then the solution set of $\mathcal{R}'$ is contained in the solution set of $\mathcal{R}$. However, often one network which is tighter than another still expresses the same set of solutions. Moreover, if we take the intersection of two equivalent networks (by intersecting constraints pairwise), we get a new equivalent network that is tighter than either.

Definition 2.3.11 *The intersection of two networks $\mathcal{R}$ and $\mathcal{R}'$, denoted $\mathcal{R} \cap \mathcal{R}'$, is the binary network obtained by pairwise intersection of the corresponding constraints in the two networks.*

Clearly,

Proposition 2.3.12 *If $\mathcal{R}$ and $\mathcal{R}'$ are two equivalent networks, then $\mathcal{R} \cap \mathcal{R}'$ is equivalent to and is at least as tight as both.*

For a proof see exercises.

Example 2.3.13 Consider the network in Figure 2.12a and the network in Figure 2.12b without R_{23}. Intersecting the two networks yield a equivalent network in Figure 2.12b that is tighter than both. □

There exists, therefore, a partial order of tightness amongst all networks that are equivalent to each other. If we intersect all these equivalent networks, we get one unique network that is equivalent to all the networks and is at least as tight as all networks. This network is called the *minimal network*. The minimal network $M(\mathcal{R})$ of a binary network $\mathcal{R}$ is the tightest network that is equivalent to $\mathcal{R}$. $M(\mathcal{R})$ is also denoted $M(\rho)$ when ρ is the set of solutions to $\mathcal{R}$.

Definition 2.3.14 *Let $\{\mathcal{R}_1, ...\mathcal{R}_l\}$ be the set of all networks equivalent to $\mathcal{R}_0$ and let $\rho = sol(\mathcal{R}_0)$. Then the minimal network M of $\mathcal{R}_0$ is defined by $M(\mathcal{R}_0) = \cap_{i=1}^{l} \mathcal{R}_i$.*

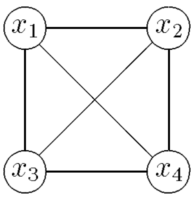
Finally, it is possible to show that the minimal network is identical to the projection network of its set of solutions.

Theorem 2.3.15 *For every binary network $\mathcal{R}$ s.t. $\rho = sol(\mathcal{R})$, $M(\rho) = P(\rho)$.*

Proof: Left as an exercise

Figure 2.13 shows the constraint graph and the unary and binary minimal constraints of the 4-queens problem. We denote the binary constraints in the minimal network by M_{ij}. The unary constraints are the reduced domains. (Compare this network to the equivalent one in Figure 2.1.) The minimal network is perfectly explicit for unary and binary constraints. That is to say, if a pair of values is permitted by the minimal network, then it is guaranteed to appear in at least one solution. Indeed, it immediately follows from Theorem 2.3.15 that,

Proposition 2.3.16 *If $(a, b) \in M_{ij}$ then $\exists\ t \in sol(M)$ s.t. $t[i] = a$ and $t[j] = b$.*



$M_{12} = \{(2,4), (3,1)\}$
$M_{13} = \{(2,1), (3,4)\}$
$M_{14} = \{(2,3), (3,2)\}$
$M_{23} = \{(1,4), (4,1)\}$
$M_{24} = \{(1,2), (4,3)\}$
$M_{34} = \{(1,3), (4,2)\}$

(a) (b)

Figure 2.13: The 4-queens constraint network. (a) The constraint graph. (b) The minimal binary constraints. (c) The minimal unary constraints (the domains).

We should note here that finding a single solution of a minimal network of constraints is not guaranteed to be easy. In fact, it is known that deciding if ρ can be representable by its projection network is NP-hard. It is still not clear, however, whether or not generating a single solution of a minimal network is hard. Empirical experience shows that normally generating a single solution using the minimal network is easy. We do speculate that generating a single solution from the minimal network is hard and we leave this as an exercise.

2.3.3 Decomposable networks

One reason that the minimal network does not solve the problem of generating a solution is that, while we can always find easily a partial solution of size 2 that is part of a full solution, we cannot guarantee that we can extend a two variable solution to a third variable In an easy way given an arbitrary minimal network, unless the minimal network is also *decomposable.*.

We know by now that a relation has a binary network representation iff it is equivalent to its projection network. We also know that the projection network is its most explicit form. It turns out, however, that a relation may be representable by a binary network while many of its projections are not.

Example 2.3.17 Consider the relations

$$\rho = \left(\begin{array}{cccc} x & y & z & t \\ \hline a & a & a & a \\ a & b & b & b \\ b & b & a & c \end{array} \right) \qquad \pi_{xyz}\rho = \left(\begin{array}{ccc} x & y & z \\ \hline a & a & a \\ a & b & b \\ b & b & a \end{array} \right)$$

One can easily verify that ρ is representable by a binary constraint network. However, the projected relation $\pi_{xyz}\rho$ is not expressible by binary networks because

$$\pi_{xyz}\rho \subset sol(P(\pi_{xyz}\rho))$$

$\square$

Definition 2.3.18 *A relation is* binary-decomposable *if it is expressible by a network of binary constraints and iff each of its projected relations is expressible by a binary network of constraints.*

It can be shown that if a relation is decomposable, then the projection network expresses the relation and all its projections. In other words, if ρ is a decomposable relation and S is a subset of the variables, then ρ_S is expressible by the subnetwork of $P(\rho)$ whose variables are restricted to the variables in S (see exercises).

2.4 Summary

The chapter provides a formal presentation of constraint networks, their solutions and thier graph representation. It demonstrates these concepts through several examples. Subsequently it focuses on some formal properties of binary constraint networks.

2.5 Chapter Notes

Graphical properties of constraint networks were initially investigated through the class of *binary constraint networks*. Montanari [112] was the first to formally define binary constraint networks. Montanari also introduced most of the concepts mentioned in the second part of this chapter, including minimal network, projection network and decomposable network. He also discussed important notions of constraint propagations which are the focus of the next chapter.

Exercises

1. Nadel [114] proposes a variant of n-queens called *confused n-queens*. The problem is to find all ways to place n-queens on an $n \times n$ chess board, one queen per column, so that all pairs of queens *do* attack each other. Propose a formulation of the problem as a constraint network. Identify variables, domains and constraints.

2. (Introductory Combinatorics, R. A. Brunaldi, 1977.) A Latin Square of order n is defined to be an $n \times n$ array made out of n distinct symbols (usually the integers $1, 2, \ldots, n$) with the defining characteristic that each of the n symbols occurs exactly once in each row of the array and exactly once in each column. For example,

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

Orthogonal Latin Squares: Let **A** and **B** be Latin squares of order n and let the entry in the i^{th} row and the j^{th} column of **A** and **B** be denoted as a_{ij} and b_{ij}, respectively, with $i, j = 1, 2, \ldots, n$. **A** and **B** are *orthogonal* if the n^2 order pairs (a_{ij}, b_{ij}) are all distinct. For example, the following juxtaposed Latin Squares are orthogonal:

$$\begin{bmatrix} (3,2) & (2,3) & (1,1) \\ (2,1) & (1,2) & (3,3) \\ (1,3) & (3,1) & (2,2) \end{bmatrix}$$

Propose a formulation of the as a constraint network. Identify variables, domains and constraints.

3. Formulate the scheduling problem appearing in Figure 2.5 where the starting time are the variables as a CSP.

4. Formulate the Huffman-Claws labelling problems as a constraint network where each arc in the drawing is a variable. Draw the primal constraint graph and the dual constraint graph.

5. Formulate the following zebra problem as a constraint network. Provide the variables, domains and constraints. Draw the primal constraint graph, the constraint hypergraph and the dual constraint graph.

   The zebra problem: There are five houses in a row, each of a different color, inhabited by women of different nationalities. The owner of each house owns a different pet, serves different drinks, and smokes different cigarettes. The following facts are also known:

   The Englishwoman lives in the red house
   The Spaniard owns a dog
   Coffee is drunk in the green house
   The Ukrainian drinks tea
   The green house is immediately to the right of the ivory house
   The Oldgold smoker owns the snail
   Kools are being smoked in the yellow house
   Milk is drunk in the middle house
   The Norwegian lives in the first house on the left
   The Chesterfield smoker lives next to the fox owner
   The yellow house is next to the horse owner
   The LuckyStrike smoker drinks orange juice
   The Japanese smokes Parliament
   The Norwegian lives next to the blue house

   The Problem: who drinks water and who owns the zebra?

6. Provide two formulations for each of the Cryptarithmetic problem below as a constraint network; provide the variables, domains and constraints. Draw the primal and the dual constraint graphs. Discuss which formulation is superior in your opinion.

   $$\text{(a)}\quad \begin{array}{rccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline & M & O & N & E & Y \end{array}$$

   (b) HOCUS +POCUS = PRESTO

   (c) GERALD + DONALD = ROBERT

7. Formulate the 3x3 magic square as a constraint problem. Draw its primal and dual graphs.

   A magic square of order n is an an $n \times n$ array of the integers $1, 2, \ldots, n^2$ arranged so that the sum of every row, column, and the two main diagonals is the same. Since

$$\sum_{i=1}^{n^2} = \frac{1}{2}n^2(n^2+1),$$

   the sum must be $\frac{1}{2}n(n^2+1)$. For example,

$$\begin{bmatrix} 1 & 15 & 24 & 8 & 17 \\ 23 & 7 & 16 & 5 & 14 \\ 20 & 4 & 13 & 22 & 6 \\ 12 & 21 & 10 & 19 & 3 \\ 9 & 18 & 2 & 11 & 25 \end{bmatrix}$$

   is a magic square of order 5, each row, column, and main diagonal add up to $\frac{1}{2}5(5^2+1) = 65$.

8. Think about a problem of your choice that can be formulated as a constraint problem. Describe and formulate as in the above questions.

9. Find the minimal network of the crossword puzzle (Figure 2.3) when the problem is formulated as a set of binary constraints.

10. Consider the following relation ρ on variables x,y,z,t.

$$\rho(xyzt) = \{(aaaa)(abbb)(bbac)\}$$

   (a) Find the projection network $P(\rho)$. Is ρ representable by a network of binary constraints? Justify your answer.

   (b) Is the projection $\pi_{xyz}(\rho)$ representable by a network of binary constraints? Is ρ decomposable?

   (c) A search space is backtrack-free along an order of its variables, d, if any partial solution along this order can be extended to a full solution. Can you find an ordering such that $P(\rho)$ is backtrack-free?

   (d) Is there a binary network representation of ρ that is decomposable?

11. Prove that if $\mathcal{R}$ and $\mathcal{R}'$ are two equivalent networks, then $\mathcal{R} \cap \mathcal{R}'$ is equivalent to and is at least as tight as both.

12. Let R_1 and R_2 be tow binary networks on n variables and the same domains. Prove that If $\mathcal{R}$ is tighter than $\mathcal{R}'$ then $sol(\mathcal{R}) \subseteq sol(\mathcal{R}')$.

13. Prove: for every binary network $\mathcal{R}$ whose set of solutions is ρ, $M(\rho) = P(\rho)$.

14. *Decomposability*

    (a) Is the set of solutions of the 4-queen problem binary decomposable?

    (b) Prove that if ρ is a binary decomposable relation and S is a subset of its variables, then $\pi_S(\rho)$ is expressible by the subnetwork restricted to variables in S of the projection network $P(\rho)$.

    (c) Is the minimal network always decomposable? Prove or show a counterexample.

15. Prove that the number of relations over n variables with domain size k is 2^{k^n}, while the number of binary constraint networks is $2^{k^2 n^2}$.

16. Hypothesis: Prove that the problem of finding a solution from the minimal network is NP-hard (suggested approach: look at graph coloring problems)

Bibliography

[1]

[2] Bar-Yehuda R A. Becker and D. Geiger. Random algorithms for the loop-cutset problem. In *Uncertainty in AI (UAI'99)*, pages 81–89, 1999.

[3] P. Ladkin ad A. Reinefeld. Effective solution of qualitative interval constraint problrms. *Artificial Intelligence*, 57:105–124, 1992.

[4] James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[5] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — a survey. *BIT*, 25:2–23, 1985.

[6] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.

[7] F. Bacchus and P. van Run. Dynamic variable ordering in csps. In *Principles and Practice of Constraints Programming (CP'95)*, Cassis, France, 1995. Available as Lecture Notes on CS, vol 976, pp 258 – 277, 1995.

[8] A. B. Baker. The hazards of fancy backtracking. In *Proceedings of National Conference of Artificial Intelligence (AAAI'94)*, 1994.

[9] A. B. Baker. Intelligent backtracking on constraint satisfaction problems: experimental and theoretical results. Technical report, Ph.D. thesis, Graduate school of the University of Oregon, Oregon, 1995.

[10] R. Bayardo and D. Miranker. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *AAAI'96: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

[11] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Uncertainty in AI (UAI'96)*, pages 81–89, 1996.

[12] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database ochemes. *Journal of the ACM*, 30(3):479–513, 1983.

[13] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[14] J. R. Bitner and E. M. Reingold. Backtracking programming techniques. *Communications of the ACM*, 18(11):651–656, 1975.

[15] M. Bjäreland and P. Jonsson. Exploiting bipartiteness to identify yet another tractable subclass of CSP. In J. Jaffar, editor, *Principles and Practice of Constraint Programming—CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 118–128. Springer-Verlag, 1999.

[16] M. Bruynooghe. Solving combinatorial search problems by intelligent backtracking. *Information Processing Letters*, 12:36–39, 1981.

[17] A.A. Bulatov and P.G. Jeavons. Tractable constraints closed under a binary operation. Technical Report PRG-TR-12-00, Oxford University Computing Laboratory, 2000.

[18] A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings 27th International Colloquium on Automata, Languages and Programming—ICALP'00*, volume 1853 of *Lecture Notes in Computer Science*, pages 272–282. Springer-Verlag, 2000.

[19] A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. The complexity of maximal constraint languages. In *Proceedings of STOC'01*, 2001.

[20] P. Cheesman, B. Kanefsky, and W. Taylor. Where the *really* hard problems are. In *International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[21] D.A. Cohen, M. Gyssens, and P.G. Jeavons. Derivation of constraints and database relations. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 134–148. Springer-Verlag, 1996.

[22] P.M. Cohn. *Universal Algebra*. Harper & Row, 1965.

[23] G.F. Cooper. The computational complexity of probabistic inferences. *Artificial Intelligence*, pages 393–405, 1990.

[24] M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65:347–361, 1994.

[25] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[26] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. In *AAAI'93: Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 21–27, 1993.

[27] V. Dalmau. A new tractable class of constraint satisfaction problems. In *6th International Symposium on Mathematics and Artificial Intelligence*, 2000.

[28] V. Dalmau and J. Pearson. Closure functions and width 1 problems. In J. Jaffar, editor, *Principles and Practice of Constraint Programming—CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 159–173. Springer-Verlag, 1999.

[29] G. B. Dantzig. *Linear programming and extensions*. Princeton university Press, Princeton,NJ, 1962.

[30] E. Davis. Constraint propagation with interva labels. *Artificial Intelligence*, 32:281–331, 1987.

[31] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[32] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960.

[33] T. L. Dean. Using temporal hierarchies to efficiently maintain large temporal databases. *Journal of the ACM*, 36:687–714, 1989.

[34] T. L. Dean and D. V. McDermott. Temporal database management. *Artificial Intelligence*, 32:1–55, 1987.

[35] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[36] R. Dechter. Constraint networks. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence, 2nd Edition*, pages 276–285. John Wiley & Sons, 1992.

[37] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55(1):87–107, 1992.

[38] R. Dechter. Mini-buckets: A general scheme of generating approximations in automated reasoning. In *IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1297–1302, 1997.

[39] R. Dechter. Constraint satisfaction. In *The MIT Encyclopedia of Cognitive Sciences (MITECS)*, pages 195–197, 1999.

[40] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

[41] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[42] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[43] Y. Deville and P. Van Hentenryck. An efficient arc consistency algorithm for a of CSP problems. pages 325–330, Sydney, Australia, 1991. Use updated reference VHDT92.

[44] Thomas Drakengren and Peter Jonsson. A complete classification of tractability in Allen's algebra relative to subsets of basic relations. *Artificial Intelligence*, 106(2):205–219, 1998.

[45] T. Drankengren and P. Jonsson. Towards a complete classification of tractability in allen's algebra. In *International Joint Conference on Artificial Intelligence (Ijcai-97)*.

[46] T. Drankengren and P. Jonsson. Maximal tractable subclasses of allen'sinterval algebra: preliminary report. In *National Conference on Artificial Intelligence (AAAI'96)*, pages 389–394, 1996.

[47] S. Even. Graph algorithms. In *Computer Science Press*, 1979.

[48] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM Journal of Computing*, 28(1):57–104, 1998.

[49] E. C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–965, 1978.

[50] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[51] E. C. Freuder and M. J. Quinn. The use of lineal spanning trees to represent constraint satisfaction problems. Technical Report 87-41, University of New Hampshire, Durham, 1987.

[52] E.C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21:958–966, 1978.

[53] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32:755–761, 1985.

[54] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI-91*, pages 227–233, 1991.

[55] D. Frost and R. Dechter. Dead-end driven learning. In *AAAI'94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 294–300, 1994.

[56] D. Frost and R. Dechter. In search of best search: An empirical evaluation. In *AAAI'94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 301–306, Seattle, 1994.

[57] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 572–578, 1995.

[58] D. Frost and R. Dechter. Looking at full look-ahead. In *Proceedings of the Second International Conference on Constraint Programming (CP-96)*, 1996.

[59] D. H. Frost. Algorithms and heuristics for constraint satisfaction problems. Technical report, Ph.D. thesis, Information and Computer Science, University of California, Irvine, California, 1997.

[60] M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA., 1979.

[61] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[62] J. Gaschnig. Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems. In *Proceedings of the Second Canadian Conference on Artificial Intelligence*, pages 268–277, Toronto, Ont., 1978.

[63] J. Gaschnig. Performance measurement and analysis of search algorithms. Technical Report CMU-CS-79-124, Carnegie Mellon University, 1979.

[64] Nicola Leone Georg Gottlob and Francesco Scarello. A comparison of structural csp decomposition methods. *Ijcai-99*, 1999.

[65] A. Gerevini and L. Schubert. Efficient algorithms for qualitative reasoning about-time. *Artificial Intelligence*, 74.

[66] M.L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[67] S. Golomb and L. Baumert. Backtrack programming. *Journal of the ACM*, 12:516–524, 1965.

[68] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: a graph-theoretic approach. *Journal of the ACM*, 40:1108–1133, 1993.

[69] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 394–399. Morgan Kaufmann, 1999.

[70] M. Haralick and G. L. Elliot. Increasing tree-search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[71] J. F. Allenand P. J. Hayes. A common-sense theory of time. In *International Joint Conference oon Artificial Intelligece (Ijcai-85)*, pages 528–531, 1985.

[72] J. Jaffar and J. Lassez. Constraint logic programming: A survey. *Journal of Logic Programming*, 19(20):503–581, 1994.

[73] P.G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[74] P.G. Jeavons and D.A. Cohen. An algebraic characterization of tractable constraints. In *Computing and Combinatorics. First International Conference COCOON'95 (Xi'an,China,August 1995)*, volume 959 of *Lecture Notes in Computer Science*, pages 633–642. Springer-Verlag, 1995.

[75] P.G. Jeavons, D.A. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *Proceedings 1st International Conference on Constraint Programming—CP'95 (Cassis, France, September 1995)*, volume 976 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 1995.

[76] P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.

[77] P.G. Jeavons, D.A. Cohen, and J.K. Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence*, 24:51–67, 1999.

[78] P.G. Jeavons and M.C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

[79] K. Kask. Approximation algorithms for graphical models. Technical report, Ph.D. thesis, Information and Computer Science, University of California, Irvine, California, 2001.

[80] H. Kautz and P. B. Ladkin. Integrating metric and qualitative temporal reasoning. In *National Conference of Artificial Intelligence (AAAI-91)*, pages 241–246, 1991.

[81] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, pages 191–194, 1979.

[82] L. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64:147–160, 1993.

[83] G. Kondrak and P. van Beek. A theoretical evaluation of selected backtracking algorithms. In *Proceedings of International Joint Conference of Artificial Intelligence (IJCAI-94)*, 1994.

[84] G. Kondrak and P. van Beek. A theoretical valuation of selected algorithms. *Artificial Intelligence*, 89:365–387, 1997.

[85] M. Koubarakis. Dense time and temporal constraints with inequalities. In *Proceedings of the Third international Conference on Principles of Knowledge-bases and Reasoning (KR-92)*, pages 24–35, 1992.

[86] M. Koubarakis. From local to global consistency in temporal constraint networks. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, pages 53–69, 1995.

[87] M. Koubarakis. tractable disjunctions of linear constraints. In *Principles and Practice of Constraint Programming (CP'96)*, pages 297–307, 1996.

[88] Manolis Koubarakis. From local to global consistency in temporal constraint networks. *Theoretical Computer Science*, 173(1):89–112, 20 February 1997.

[89] V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI magazine*, 13(1):32–44, 1992.

[90] P. B. Ladkin. Time representation: A taxonomy of interval relations. In *National Conference of Artificial Intelligence (AAAI-86)*, pages 360–366, 1988.

[91] P. B. Ladkin. Metric constraint satisfaction with intervals. In *Technical Report, TR-89-038, Interional Computer Science Institut, Berkeley, CA*, 1989.

[92] P. B. Ladkin and R. D. Maddux. On binary constraint networks. In *Technical Report, Kerstel Institute, Palo Alto, CA*, 1989.

[93] P.B. Ladkin and R.D. Maddux. On binary constraint problems. *Journal of the ACM*, 41:435–469, 1994.

[94] J. larrosa. Boosting search with variable-elimination. *In pproceedings of CP2000*, 2000.

[95] J. L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1), 1978.

[96] S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[97] C.E. Leiserson and J. B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. In *21st annual Allerton Conference on Communication, Control and Computing*, pages 204–213, 1983.

[98] H. Kautz M. Villain and P. van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In *Readings in Qualitative Reasoning about Physical Systems¡ Daniel Weld and J. de Kleer, Morgan Kaufman*, pages 373–381, 1989.

[99] H. Machida and I.G. Rosenberg. Classifying essentially minimal clones. In *Proceedings 14th International Symposium on Multiple-Valued Logic (Winnipeg, 1984)*, pages 4–7, 1984.

[100] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[101] A. K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, 1987. Use updated reference Mackworth92.

[102] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25, 1985.

[103] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[104] D. Maier. The theory of relational databases. In *Computer Science Press, Rockville, MD*, 1983.

[105] K. Marriot and P. Stuckey. Programming with constraints, an introduction. In *MIT Press*, 1998.

[106] D. A. McAllester. Truth maintenance. In *AAAI'90: Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1109–1116, 1990.

[107] J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Science*, 19:229–250, 1979.

[108] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87:343–385, 1996.

[109] I. Meiri. Combining qualitative and quantitative constraint satisfaction problems. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, Anaheim, CA, July 1991.

[110] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[111] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[112] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(66):95–132, 1974.

[113] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[114] B. A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–299, 1989.

[115] B. A. Nadel. Some applications of the constraint satisfaction problem. In *AAAI-90: Workshop on Constraint Directed Reasoning Working Notes*, Boston, Mass., 1990.

[116] B. Nebel and H. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[117] B. Nebel and H-J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, January 1995.

[118] N. Nillson. In *Introduction to Artificial Intelligence*. Tioga Publishing, 1978.

[119] B. Nudel. Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics. *Artificial Intelligence*, 21:135–178, 1983.

[120] Y. Deville P. van Henetenryck and Teng C. A generic arc-consistency algorithm and its specialization. *Artificial Intelligence*, 57:291–321, 1992.

[121] J. Pearl. Heuristics: Intelligent search strategies. In *Addison-Wesley*, 1984.

[122] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[123] P. Prosser. Forward checking with backmarking. Technical Report AISL–48–93, University of Strathclyde, 1993.

[124] P. Prosser. Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.

[125] P. W. Purdom. Search rearangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.

[126] Jr. R. Bayardo and D. P. Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *Fourteenth International Joint Conference on Artificial Intelligence(I95)*, pages 558–562, 1995.

[127] I. Meiri R. Dechter and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1990.

[128] I. Rish and R. Dechter. Resolution vs. search; two strategies for sat. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000.

[129] I.G. Rosenberg. Minimal clones I: the five types. In *Lectures in Universal Algebra (Proc. Conf. Szeged 1983)*, volume 43 of *Colloq. Math. Soc. Janos Bolyai*, pages 405–427. North-Holland, 1986.

[130] D. G. Corneil S. A. Arnborg and A. Proskourowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics.*, 8:277–284, 1987.

[131] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *ECAI-94*, pages 125–129, Amsterdam, 1994.

[132] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[133] E. Schwalb. Temporal reasoning with constraints. Technical report, Ph.d. thesis, Information and Computer Science, Universiy of California, Irvine, 1998.

[134] E. Schwalb and R. Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, pages 29–61, 1997.

[135] E. Schwalb and L. Villa. Temporal constaints: A survey. *Constraints*, pages 1–20, 1998.

[136] P.P. Shenoy. Binary join trees. pages 492–499, 1996.

[137] K. Shoiket and D. Geiger. A proctical algorithm for finding optimal triangulations. In *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 185–190, 1997.

[138] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28:769–779, 1981.

[139] M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196.

[140] S. Stone and J.M. Stone. Efficient search techniques-an empirical study of the n-queen problem. In *Technical report RC (#54343) IBM T.J. Watson*, 1986.

[141] A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures*. University of Montreal, 1986.

[142] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computation.*, 13(3):566–579, 1984.

[143] E. Tsang. *Foundation of Constraint Satisfaction.* Academic press, 1993.

[144] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. 1.* Computer Science Press, 1988.

[145] R. E. Valdes-Perez. Spatio-temporal reasoning and linear inequalities. In *Tech Report AIM-875, Artificial Intelligence Lab, MIT, Cambridge, MA*, 1986.

[146] R. E. Valdes-Perez. The satisfiability of temporal constraint networks. In *National Conference of Artificial Intelligence (AAAI-87)*, pages 256–260, 1987.

[147] P. van Beek. *Exact and Approximate Reasoning about Qualitative Temporal Relations*. PhD thesis, University of Waterloo, 1990. Available as: Department of Computing Science Technical Report TR-90-29, University of Alberta.

[148] P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

[149] P. van Beek and R. Dechter. On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, 42:543–561, 1995.

[150] P. van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 42(3):543–561, May 1995.

[151] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

[152] P. VanBeek. Approximation algorithms for tempral reasoning. In *International Joint Conference on on Artificial Intelligence (IJCAI89)*, pages 1291–1296, 1989.

[153] P. VanBeek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[154] M. Vilain. A system of resoningabout time. In *National confrence of Artificial Intelligence (AAAI'82)*, pages 197–201, 1982.

[155] M. Villain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *National Conference on Artificial Intelligence (AAAI'86)*, pages 377–382, 1986.

[156] D Waltz. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, 1975.

[157] R. Zeidel. A new method for solving constraint satisfaction problems. In *International Joint Conference oon Artificial Intelligece (Ijcai-81)*, pages 338–342, 1981.