Automating Resource Management for Distributed Business Processes

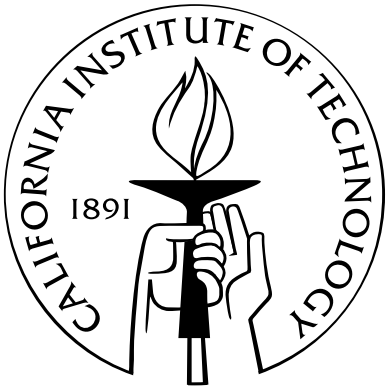
Thesis by

Roman Ginis

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2002

(Defended October 25, 2001)

Acknowledgements

Many thanks to all of my wonderful teachers, specifically: To my research advisor K. Mani Chandy, for continuously inspiring me to look for the "big idea," for giving the opportunity and encouragement to explore a wide range of topics and for teaching by example to identify the essence in research problems. To Jason Hickey, Jim Arvo, Leonard Schulman and Niles Pierce, members of my thesis committee, for helping me to distill key insights and offering alternative perspectives. To Michel Charpentier, who taught me the discipline of rigor in proofs and the rewards that go with it. To John Thornley, for pragmatic advice on the clarity of academic expression. To Victor Fay-Wolfe, for opening the door to a freshman and introducing me to research. To Geri Lifshey and Mary Alice Walsh for teaching me to communicate during my first steps in the new land and for encouraging me to always shoot for the stars. Additional thanks to the members of my research group: Eve Schooler, Joseph Kiniry and Daniel Zimmerman for being good friends and colleagues. Finally, I would like to thank my family for always supporting me with love, wisdom and advice, especially my mother who has always demanded the very best and my dad who continues to raise the bar in intellectual, social and personal achievement, and who has always been my hero.

The research in this thesis was supported in part by grants from the Air Force Office of Scientific Research, the Lee Center for Advanced Networking and a National Science Foundation Graduate Research Fellowship.

Abstract

A distributed business process is a set of related activities performed by independent resources offering services for lease. For instance, constructing an office building involves hundreds of activities such as excavating, plumbing and carpentry performed by machines and subcontractors, whose activities are related in time, space, cost and other dimensions. In the last decade Internet-based middleware has linked consumers with resources and services enabling the consumers to more efficiently locate, select and reserve the resources for use in business processes. This recent capability creates an opportunity for a new automation of resource management that can assign the optimal resources to the activities of a business process to maximize its utility to the consumer and yield substantial gains in operational efficiency.

This thesis explores two basic problems towards automating the management of distributed business processes: 1. How to choose the best resources for the activities of a process (the Activity Resource Assignment - ARA - optimization problem); and 2. How to reserve the resources chosen for a process as an atomic operation when time has value, i.e., commit all resources or no resources (the Distributed Service Commit problem - DSC). I believe these will become the typical optimization and agreement problems between consumers and producers in a networked service economy.

I propose a solution to the ARA optimization problem by modeling it as a special type of Integer Programming and I give a method for solving it efficiently for a large class of practical cases. Given a problem instance the method extracts the structure of the problem and using a new concept of variable independence recursively simplifies it while retaining at least one optimal solution. The reduction operation is guided by a novel procedure that makes use of the recent advances in tree-decomposition of graphs from the graph complexity theory.

The solution to the DSC problem is an algorithm based on financial instruments and the two-phase commit protocol adapted for services. The method achieves an economically sensible atomic reservation agreement between multiple distributed resources and consumers in a free market environment.

I expect the automation of resource management addressed in my thesis and elsewhere will pave the way for more efficient business operations in the networked economy.

Contents

List of Definitions

List of Examples

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses a problem which I believe will become central to the world economy over the next decade: how to better automate and optimize resource management in business operations, as the global economy goes online. The solutions I present here make it possible to quickly identify and reserve the optimal resources to maximize the objectives of business processes operating in a networked, service-oriented economy.

Business processes are sets of related activities (or tasks). For example, constructing a new office building is a business process that involves activities such as digging, carpentry and plumbing. Processes arising in crisis management, vehicle manufacturing and travel also consist of discrete activities. An individual traveling to another city performs the activities of flying to and from the destination and using a hotel and a rental car while there. The resources needed for the activities can be people, machines, organizations and even information access rights. The relationships connecting the activities into a process are usually in time, distance, capability, load and cost. For instance, for a traveler there is a time dependency between the arrival of her flight and the check-in time of the hotel room. Viewing this dependency as a function from the flight and hotel choices to value, would probably give a good value if the time difference is small and a progressively poor value with the larger difference. Similarly, a relationship in cost between the 'to' and 'from' flights could be represented by a function that may have a high value if the flights are operated by the same provider and low otherwise.

I assume that the parties who execute business processes (from now on consumers) operate in a service-oriented, free-market economy and can locate the potential resources to communicate and negotiate with them electronically. The parties can lease the services of machines, individuals and organizations to perform each of the activities of a process and educe a value from a completed process depending on the degree to which the chosen resources satisfy the relationships among the activities. I call the processes executed in this way *compositional business processes.*

The problems I examine are different from many problems in Operations Research in that the resources in our problems are bought and sold in a dynamic marketplace by many independent

buyers and sellers, consumers have a choice among resources to employ in a given activity, and resources have qualities that make them more or less desirable for a given business process.

I propose two key problems associated with managing these processes: First, the *Activity Resource Assignment*, is an optimization problem that formalizes the selection of optimal resources for the activities of a given process based on an objective function expressed as an algebraic composition of relationships among the activities. The problem is a special type of Integer Programming for which I give a new solution method that finds an optimal solution in polynomial time for many seemingly intractable instances. The exponent of the polynomial in the algorithm is related to the tree-width of a relationship graph that can be constructed for any given process specification.

The second, the *Distributed Service Commit* problem, addresses the need to coordinate the resources chosen by an optimization to carry out the business process. Since the resources are distributed, independent and traded in a marketplace, achieving an atomic distributed agreement among them requires both an algorithmic and an economic approach. We offer a solution that combines distributed computing techniques with financial derivatives to achieve the required collective agreements. It is a generalization of the two-phase commit synchronization protocol with the American call option financial instrument.

Together the solutions to these two basic problems, intrinsic to their environment, can help automate and efficiently optimize resource management for many business processes.

1.1 Compositional Business Processes

The new capabilities in the networked service economy create new opportunities and challenges. The key opportunity for any business is to specialize in what it does best, and to outsource all other business functions. An important challenge is to find a way to manage a company where the bulk of its operations are done by external service providers.

Specialization has been the trend even before Adam Smith wrote his "Wealth of Nations" more than two centuries ago and it has accelerated significantly in the last decade. A global networked service economy will take business specialization to the extreme. If services can be automatically located, selected and reserved, there will be less motivation to do those tasks "in-house." With more companies setting up the infrastructures for doing business online, this has become more possible than ever, and the trend is very likely to continue.

From an engineering perspective making compositional business processes work involves solving four fundamental issues: *connectivity, correctness, selection and commitment.*

Connectivity deals with infrastructures and everything related to setting up the "wires" and interfaces between services and consumers. One such infrastructure technology is Object Management Group's CORBA (Common Object Request Broker Architecture) [26] that abstracts

the network to distributed client/server interactions. Another is Sun Microsystems' Java Message Service[33] that offers distributed peer-to-peer communications. In the last several years numerous American corporations, most notably the banking system, have "wrapped" their services using such infrastructures precisely to make service composition possible.

Correctness concerns what can be said about a system of components or services when they are put to work together. Specifically, given the characteristics and the logical properties of each component, the researchers ask what predicates are satisfied in composition. The correctness properties are critical when designing the specifications for any composite system or business process. Without a formal, systematic approach any brute force method is almost certainly doomed to failure for all but the most trivial cases. On the other hand, doing formal modeling of distributed systems and proving their correctness is currently a rather sophisticated art. Thus, the correctness of composition today is a lively area of research.

Selection is the optimization aspect of composition. It addresses how to choose the best resources for a business process. Namely, once a process has been designed and proved correct, and all the infrastructure issues have been worked out, the task is to choose the best resources to execute the process. For example, if an activity of a process involves leasing a plumber for 3 hours as part of a new house construction and there are a dozen plumbers available in the area, one may like to choose the most experienced plumber at the least cost and who fits into the construction schedule.

Selection issues have been extensively studied for specialized settings, such as for factory job scheduling [32]. However, they are usually limited to one or two dimensional constraints (such as time and mode of operation) and these solutions are not easily generalizable.

Commitment between the resources and a consumer who runs a business process becomes paramount when the resources are independent, distributed and whose state (such as availability) changes often. To make a set of resources work for a business process each resource must agree to perform its function at the required time, place and cost. We require that all resources must enter an agreement for a process to succeed and therefore we need a global commitment among the chosen resources and the consumer. Since resources are for lease, one can compute an opportunity cost for any segment of their time. The opportunity cost of resource r during time t is the value that the owner of the resource could have earned by leasing it during that time. Typical distributed agreement algorithms do not take time into account, whereas here time has a direct value and using an algorithm that requires resources to wait without compensation defeats its viability.

This issue is intrinsic to management of any distributed resources and it emerges vividly when the resources become tradeable commodities.

In this thesis we focus on the latter two problems: Selection, defined earlier as the Activity Resource Assignment (ARA) problem, and the Commitment problem, which we call Distributed Service Commit (DSC). In the next two sections we outline our solutions.

1.2 Activity Resource Assignment

1.2.1 Specification

Let business process P be defined by a set of activities $1, ..., n$. Suppose that each activity i can be performed by any of M resources. For example if i is the 'plumbing' activity in a building construction project, then there are m different plumbers who could potentially do this work. For a carpentry activity j there are also m carpenters to choose from. If there are fewer than m resources for a given activity, then we assume that there are "fake" resources to bring the total upto m where the fake resources have exorbitant costs over exorbitant time.

We can associate a variable x_i with each activity i that can take one of m values. That is, if $x_i = 3$ then the 'third plumber' from M is chosen. Assigning a value to each x_i for all $1 \leq i \leq n$ makes a *solution*. That is, exactly one resource has been assigned to each activity.

There are usually relationships among the activities. Suppose we want to specify that a plumber (activity i) must start work after the carpenter (activity j) is done and we prefer the plumbing to begin as soon as possible. We can represent this by a relationship function $f_1(x_i, x_j)$ to indicate our time preference. For each of the m^2 combinations of plumbers and carpenters it gives a 'value' that shows how well each pair satisfies our 'as soon as possible' relationship. For example, for a given carpenter, say $x_j = 4$, the plumbers who are available sooner than others after the carpenter completes his job would have higher relationship value than those who can only start later. Of course, for a different carpenter the value of the relationship for each plumber may be entirely different.

The relationships can be n-ary, forming functions of up to n parameters. A problem can have many relationships among the activities. To assign priorities among the relationships, there is one *objective* function ϕ that is an algebraic composition of the relationship functions. For instance, a process with five activities $\{1, ..., 5\}$ and three relationships $\{f_1, f_2, f_3\}$ can have the objective function defined as $\phi = f_1(x_1, x_2) * f_2(x_1, x_3) + f_3(x_3, x_4, x_5)$. The goal is to find a solution (an assignment of values to variables $x_1, ..., x_n$) so as to maximize the value of ϕ.

In sum, a business process optimization problem (ARA) is given by a triple: (X, F, ϕ), where the set $X = \{x_1, ..., x_n\}$ contains variables corresponding to the activities, the set F contains the relationship functions and ϕ is an objective function. This is demonstrated by the following example:

Example 1.1 (Six-Variable Problem)

Given X, F *and* ϕ *where* $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$,

$$F = \begin{cases} f_1 : x_1 \times x_2 \times x_3 \rightarrow \mathbb{R}^+ \\ f_2 : x_1 \times x_3 \rightarrow \mathbb{R}^+ \\ f_3 : x_4 \times x_5 \rightarrow \mathbb{R}^+ \\ f_4 : x_1 \times x_2 \times x_6 \rightarrow \mathbb{R}^+ \\ f_5 : x_3 \times x_4 \times x_5 \rightarrow \mathbb{R}^+ \end{cases}$$

and $\phi = f_1^{f_2} + f_5 f_3 + f_4$

Find an n-vector $x_{opt} = (\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{x}_4, \dot{x}_5, \dot{x}_6)$ *that maximizes* ϕ*, where* $\dot{x}_i$ *represents an assignment to* x_i *of a value from* $1, ..., m$ *for all* $1 \leq i \leq n$.

1.2.2 Solution

We propose a deterministic method to solve a class of discrete non-linear optimization problems relevant to business process optimization. The first key idea behind the method is a reduction operation (which we call relaxation) that recursively transforms a problem instance to a simpler one while preserving an optimal solution. We define a relaxation operator, $\rho_x(S)$ which returns an optimal value of variable $x \in X$ for any assignment of variables in $S \subseteq X$ with respect to ϕ. Thus when a variable x is relaxed, we replace it by $\rho_x(S)$ everywhere in ϕ, making ϕ no longer a function of x. With S carefully chosen (as will be shown shortly) the relaxation reduces the problem by one variable, at a computation cost of $O(m^{|S|+1})$, yet at least one maximum is preserved.

The function $\rho_x(S)$ can be generated for any $x \in X$ because the problem is discrete and it is straightforward how to do it when $S = X$. Merely attempt every possible assignment of variables $X - \{x\}$ and solve ϕ for an optimal x. However, the computation time to produce $\rho_x(S)$ would then be exponential in $|X|$. The second key idea in our method is an observation that for any x we can find a set $D_x \subseteq X$, which we call the dependent set, such that $\rho_x(X) = \rho_x(D_x)$ and it is often the case that $|D_x| \ll |X|$, making the time to generate $\rho_x(S)$ tractable. We can determine D_x from the structure of ϕ and we show how to do that for all algebraic ϕ. The final computation cost of the complete reduction is the sum of the costs to generate the relaxation functions $\rho_x(S)$ for each variable.

Unfortunately relaxing a variable x in this way has a price – it makes all of the variables in D_x mutually dependent. Thus, relaxing the variables in the "wrong order" can lead to a much costlier route to an optimal than needs to be. The third key insight we offer is a way to choose the order of variable relaxation such that the computation time could be reduced. We propose two approaches: one based on tree-decomposition of graphs that allows us to partition the variables into special sets and then apply the relaxations separately to each set, yielding a dynamic programming solution. The second approach consists of two heuristics that select the variables to relax in a greedy

way based on simple local properties: a) relaxing the variables with the least number of dependent variables first, and b) relaxing the variables which create the least number of new dependencies first.

The tree-decompositon method is optimal cost (note that either way we always find the optimal resource assignment, merely at different costs) for certain types of problem topologies, and it works reasonably well for many others. The heuristics work remarkably well on all the cases that we have tried and compared to the optimal cost, but they are provably not optimal (see chapter 6 for a counter example).

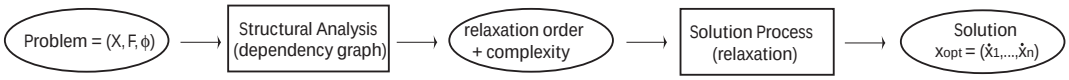


Figure 1.1: Solution Outline

The solution method is outlined in Figure 1.1. It has a nice property that given a problem instance one can first obtain the *complexity* of solving it without carrying out the solution. The Structural Analysis step in the figure works in linear time and determines whether it has a "good" topology for both the tree-decomposition method (due to [15], [23], [35], [29] and others) and the heuristics. It computes the exponent of the computation polynomial without even running the optimization. This allows one to choose the better method for each problem instance.

The worst case is $\phi = f_1(x_1, ..., x_n)$ where all the variables are dependent on each other and $|D_x| = |X|$, yielding $m^{|X|}$ computation time for our method (of course, the size of the input is exponential in this case). Fortunately, the typical problems that seem to arise in the applications we consider often have the topologies that make fast solutions possible.

The tree-decomposition approach allows us to tap a well-developed field of graph complexity to classify optimization problems and to find efficient solutions for a large class of discrete non-linear optimization problems. We give a mapping between the optimization problem topologies and graph topologies (section 4.2) and show how to use it on problems relevant to our applications. This seems to be a novel approach to solving these types of optimization problems, and while the cases we present here are sufficient for our purposes there is much room for further research.

1.3 Distributed Service Commit

1.3.1 Specification

A *resource* is a service or an entity that can perform some function. It can be scheduled or reserved for a duration of time relative to some "universal" clock and it has a unique name – a URL (Universal Resource Locator). Furthermore, it is supervised by a software 'manager' that enables other entities to reserve and release the resource and to determine when the resource is available. An example of a resource is a carpenter, whose software manager is a program that maintains his schedule.

A *consumer* is any entity that attempts to reserve one or more resources to perform a business process. For example, a construction company behaves as a consumer when it reserves some carpenters, plumbers, electricians, etc. to build a house. It can also act as a resource, when some other firm hires it to erect a building complex, along with possibly other service providers, such as an accounting firm to oversee the operation and a bank to loan the money for the construction.

A *distributed system* is a set of computers able to exchange messages. It can be as small as a system that supports a workgroup in a company or as large as the Internet. The fundamental property that makes it harder to engineer applications for distributed systems than programs for personal computers is the uncertainty about the state of the system. Namely, because of the intrinsic delay in communication between any two computers, no computer in a system knows the exact state of its peers. For example, no bank computer knows for certain whether its ATM machines are operating correctly at any given moment, because even though they can exchange messages, by the time an ATM's message reaches the bank, the ATM may have lost power.

To deal with this uncertainty numerous algorithms have been invented to coordinate computers in distributed systems. These solve common problems that designers face when developing their applications. One such classic problem is the *Distributed Agreement Problem*. It requires a predicate to be established among a set of computers in a given system. For example, a common problem that banks face is keeping the databases between the main office and the branch offices synchronized. The standard solutions use a variety of special messages that travel between the computers of the system placing them into certain states when messages are received. This way it is possible to transition the computers from state to state and to compute a function that yields the required predicate.

Now, if we model the resources and the consumers as a distributed system, the state of interest to the consumers is (at least) the availability of the resources they select for their business processes. The Distributed Service Commit problem we introduce here requires a consumer to obtain a commitment from a resource for each activity in order to execute the process. A commitment by each resource to provide a service at a specified time is a type of distributed agreement that the consumer must achieve in its distributed system.

Unfortunately, no standard algorithms can be used to solve this instance of the distributed agreement problem because of the additional property that our resources operate in a service economy. The standard solution methods ignore time, instead promising to achieve an agreement eventually. However, in an environment where resources are for lease, time has value (there is an opportunity cost associated with time). Therefore, the traditional algorithms are unacceptable for this problem.

1.3.2 Solution

We propose an economically-inspired method to solve the distributed agreement problem. The idea is to use short-term financial contracts to mitigate risks and opportunity costs during transactions

between consumers and resources.

Consider a consumer C and a set of resources $\dot{x}_1, ..., \dot{x}_n$, which the consumer wants to reserve. We require the consumer to reserve all n resources atomically, all together or none of them, for the business process to succeed (if the problem instance permits for a subset of the resources to be reserved atomically instead of all of them, the DSC problem is still relevant for that subset). To do this, C can send messages to any $\dot{x}_i$ and attempt to reserve it. Two types of messages are possible: either the consumer requests a commitment from $\dot{x}_i$ (if it is available), or it asks the resource to wait until it ensures that all the other resources are also available and are in a waiting state. In this way the consumer can either get all the resources into a waiting state called "poised to be reserved," or at least one of them is unavailable and the agreement among this set of resources is not possible.

While this protocol of interaction could work well in the typical resource allocation problems, it is unreasonable to expect that the resources in an economy would want to be in a waiting state for free while some consumer is attempting its atomic reservation. From the point of view of the resource, asking the resource to wait is equivalent to using the resource. There is an opportunity cost associated with waiting and either the consumer or the resources must absorb it. To address this problem we propose using a financial instrument we call Micro-Option during the reservation process. Micro-Option is derived from the common American Call Option instrument adapted to services instead of commodities and integrated with our expected mode of consumer-resource interaction. A Micro-Option is a short-term right to reserve a resource at some specific time in the future for a specific price. This right has a duration (e.g. a few seconds or minutes) and a value which, in a free market, is closely related to the opportunity cost of the resource for that duration. Micro-Options can be implemented efficiently with very low transaction costs.

Thus, a consumer who wants to reserve n resources can execute a two-phase algorithm where he first attempts to purchase the rights to reserve the resources, and if he is successful at obtaining all the rights, he then proceeds with the reservation. In this way, the resources can get compensated for waiting while they are being evaluated for reservation and the consumers need not waste funds by paying for the reservations without knowing that all the resources are obtainable.

1.4 Key Contributions

In this thesis we identify two problems that we believe to be fundamental to any applications that require resource management in a networked service economy. We propose a solution for each one that we believe to be general yet practical for the applications we consider. Specifically the thesis contains:

- A formal model for resource management of business processes.
- An exact method for optimal assignment of resources to activities (ARA).
- A method to determine a priori the computational complexity of finding an optimal solution with the given method without performing the optimization search.
- A method for an atomic reservation of a group of resources (DSC) when time has value.

1.5 Thesis Outline

Chapter 2 describes how our two problems (ARA and DSC) relate to the existing body of research. Chapter 3 describes each problem precisely, mapping the general requirements into a formal specification and contains a proof the optimization problem is NP-hard. Chapter 4 explains the intuition for the ARA problem solution. Chapters 5 and 6 contain a formal treatment of the optimization problem. Chapter 7 contains some simulation results of applying our optimization method to a number of generated problems. In Chapter 8 we give a solution to the reservation problem and present some applications in Chapter 9. The conclusions and future work ideas can be found in Chapter 10.

Chapter 2

Background

2.1 Business Process Automation

I am writing this document in the first months of the 21st century. Much has happened to the role of computers in our society. In just a few decades computers have gone from an oddity, to necessity, to main source of entertainment and to some they have even become a new window to the world. The 1950's and 60's saw the beginning of computing automation in the United States. Computers left the confines of military applications and found numerous uses in business. One of the most prominent drivers of automation through computing was Dantzig's Linear Programming model and the Simplex optimization algorithm [12]. It has been applied millions of times to optimize resource allocation and is still one of the most widely used techniques in solving resource management problems. The Critical Path Method (CPM)[18], the Program Evaluation and Review Technique (PERT)[9], Job-Shop and the Resource Constrained Project Scheduling (RCPS)[32] models, invented a few years later, have enabled businesses to optimize their project management, production and supply lines even further. In the 1970's and 80's database technology was developed, refined and finally made into off-the-shelf products available to everyone – from "Mom 'n Pop" shops to multinational corporations. Larger storage capacities have enabled businesses to store the state of their day-to-day operations, intellectual property and key information on reliable computers.

The 1990's may go down in history as the infrastructure decade. Thousands of miles of fiber have been laid across the world and the Internet, via TCP/IP, HTML and XML, has given the businesses and individuals the potential to communicate using a standard data exchange technology at acceptable speeds. The financial and legislative infrastructures have not been far behind. Seeing the opportunity to drive the early electronic markets the banks scrambled to offer micro-payment services and the 42^{nd} President of the United States, William Jefferson Clinton, signed a Digital Signature Act into law, making it possible for the first time to make legal business transactions entirely in the digital realm.

The computer has shown its potential to automate business processes. However, what exists now

is just a precursor to a much larger opportunity to automate processes not only on the scale of an enterprise, but on the scale of the entire economy.

2.2 Economic Perspective

The U.S. economy is currently one of the most advanced capitalist economies in the world. Though not perfect, it came a long way since the economy described in Adam Smith's "Wealth of Nations" over two hundred years ago. A work in progress, it is continuously perfected to be able to handle the complexities of human and business interactions far greater than ever conceived by Mr. Smith. In recent years, starting in mid 80's with the age hailed as 'computerization,' businesses around the country have undergone a fundamental restructuring of their internal operations by delegating the storage and retrieval of their vital business information from humans and paper folders to a network of computers. As inter-business networks have grown and the Internet has swept popular culture, first computer scientists and then CIO's of many high-tech firms realized the potential of the network and have begun to imagine a world where most inter-business transactions would eventually take place entirely on-line. These were assigned catchy acronyms like (B2C - business to consumer and B2B - business to business) and a myriad startups sprinted to revolutionize business operations.

The initial approach was the most obvious one. The reasoning went, since the network was the cheapest medium of information exchange, if we build it they will come. Namely, the assumption was that if the businesses *could* "do business" over the network, they definitely *would.* To this end, a wrapper standard emerged (XML) and for a while was hailed as a breakthrough. However, the migration to an on-line business model is taking much longer than expected.

There was a problem: the inefficiency in the market that a computer B2B network was supposed to alleviate was too *insubstantial* compared to the investments required to implement the network. When a company makes an order of 100,000 widgets, it does so in a transaction with another company that lasts for months until the next transaction. For such substantial transactions, the cost or the speed of doing it over the e-mail, phone or Fedex versus a specialized trading network is virtually immaterial. Namely, if $1000 is saved on a transaction of $10 million, it is not sufficient for the transacting companies to invest millions of dollars to computerize this process. It is not to say that it should not, or would not be computerized eventually, but not immediately. Thus, the main reason why the simple networking of companies has failed to deliver new medium for business is because the existing communication between businesses is only marginally inefficient compared to the proposed new one. It would not be the communication capabilities but applications built on top of the new network, offering new ways to bid for, trade and optimize the assignment and management of resources, that can (and do) yield substantial improvements in efficiency.

The promise of computerization (or automation) is to remove inefficiencies in business processes.

Therefore, the question that many have asked: "how should we build a network to enable businesses to make transactions using computers?" should be replaced by: "how can we remove inefficiencies in the current business processes using technology?" The difference here, is the focus on the disease - business inefficiency rather than on a cure - a computer, network or XML. A true innovation must eliminate substantial inefficiencies in the existing business processes to make an important difference.

An efficient business process can be described as one where all activities of the process are executed with the highest utility and the relationships between activities are satisfied to the best degree possible. The relationships can be in time, space, combined resource usage or any other dimensions. The important inefficiencies in business processes today are in the myriad of "small" and "short term" transactions that businesses want to do with each other on a daily or hourly basis. These transactions involve communication between people and their transportation from place to place, energy utilization and having enough for employees, consultants and contractors to work on. There lie the real inefficiencies, and they can be solved by a new type of computerization: novel systems and optimization algorithms that not only allow one computer database to talk to another, but that efficiently manage the resources used for different tasks – namely people's time, energy, communications and physical facilities of the firms.

Such capabilities can become possible if the following conditions hold:

1) There are many more types of resources available for lease

2) There are electronic markets for exchanging these resources

3) There exist financial instruments designed for trading services

4) Optimization algorithms exit that allow one to choose the best resources for a given task.

5) There exists a common taxonomy of resources and activities

Up to now, businesses have been trading services in large "chunks" and long timeframes. For example, people are employed for months or years and network connections are sold on monthly contracts. One of the reasons for this is the management difficulties in dealing with a large number of different business components. However, if this hassle were reduced by new automation technology, the opportunity to make businesses leaner may drive companies to achieve a new economic optimum.

2.3 Relationship to Integer Programming

The Activity Resource Assignment problem is a special case of Integer Programming. We can represent any instance of ARA as IP as follows:

For example, let $P_{ARA} = (X, F, \phi)$ be an instance of ARA where $X = \{x_1, x_2, x_3\}$. The goal of the problem is to find an assignment of variables $\{x_1, x_2, x_3\}$, each taking a value in $\{1, ..., m\}$, that maximizes ϕ.

The corresponding IP problem can be set up as follows: Let Y be a set of variables where $y_{ijk} \in Y$

takes the value of 0 or 1 and the i, j and k stand for an assignment of the activity variables x_1, x_2 and x_3 respectively. For example, $y_{147} = 1$ if and only if $x_1 = 1, x_2 = 4, x_3 = 7$. Furthermore let there be a variable $c_{ijk} \in \mathbb{R}^+$ such that $c_{ijk} = \phi(x_1 = i, x_2 = j, x_3 = k)$. Then, for any instance of ARA the Integer Programming problem can be stated as follows:

Maximize $\phi' = \mathbf{c}^T\mathbf{y}$, subject to $\sum_{ijk} y_{ijk} \leq 1$, $y_{ijk} = 0$ or 1.

2.4 Other Related Problems

Problems related to the Activity Resource Assignment problem include job-shop scheduling and problems in a new area of research referred to as Market-Based Control [10] .

The Job Shop problem is NP-complete [14]. It can be specified as a set J of jobs each of which has to be performed on machines $M_1, ..., M_m$. Each job consists of a set of activities A, where each activity a has to be done uninterrupted on a different machine M_i and takes d_a time. The sequence in which the activities are performed on the machines differs from job to job and there are precedence constraints among the activities of each job. The objective is to find an ordering of activities for each machine to minimize the total makespan for all the jobs.

There are variations on the problem, such as Open-Shop where the activities have no ordering and Flow-Shop where all the acclivities have the same ordering. Furthermore, Sprecher [32] presents a more general case of Job Shop called Generalized Resource-Constrained Project Scheduling problem that allows perishable and non-perishable resources associated with activities as well as different modes of operation for each machine.

The problems associated with Market-Based Control explore the concept of using market mechanisms such as auctions in simulated markets to allocate resources.

These problems are not directly applicable to solve our ARA problem because the specifications are substantially different. A major difference is that the resources in the Job Shop problem are machines that are not traded in a marketplace.

Chapter 3

Model and Problem Specification

3.1 Overview

Our goal is to model business processes consisting of several related activities. We assume that each activity requires some resources to commence. A process completes when all of its activities complete. Furthermore, the processes we consider have the following basic properties: 1) For each activity there are a number of resource choices (for instance, in a trip the choices could be different flights available between the given source and destination locations), 2) the activities are related to each other in some important dimensions, such as time, space and cost incentives, 3) resources must be chosen for each activity, and 4) the total utility of the business process to the consumer is a function of a) the fitness of the chosen resources with respect to a specification and of b) the degree to which the relationships between the activities are satisfied in composition.

We present a formal model for such business processes. We give an algorithm for optimal selection of resources for the activities and a method to reserve them atomically. We also provide proofs that the solutions found by the algorithm are optimal and show evidence that the algorithm finds an optimal solution in polynomial time for many typical instances. The degree of the polynomial for a given problem instance is determined by the complexity of the relationships between the activities in the process and the form of the objective function.

3.2 Resource Constraints

Each resource used in a business process often needs to satisfy some constraints. For example, airline flights that are part of a business trip may be required to start and end on a specific day. This requirement is absolute and immediately restricts the set of possible options. On the other hand, the requirement may be lax, specifying only the week of departure and a set of acceptable sources and destinations. This may be convenient when the source and/or destination cities have multiple airports in the vicinity. Thus, the key effect of resource constraints is the restriction of

options for each activity.

It may be possible and desirable to define a function that compares one resource to another as options for a given activity. For example, one could define an ordering relation that states a consumer preference for flights of one airline company over the flights of another. This function could map specific flights to values, allowing different flights to be compared. In our model we incorporate these preferences in functions that reflect not only the preferences between resources within an activity, but the utility of resources in combinations. The next section describes these forms of relationships.

3.3 Relationships

A consumer is interested in identifying the resources that maximize the utility of his business process based on some subjective preferences. This utility and the preferences can be specified as 1) the resource constraints described earlier, as 2) utility relationships favoring the choice of specific configurations of resources among several activities and 3) they can also be given by an objective function, which we discuss in the next section.

The simplest type of relationship is a binary relationship between two activities. For instance, in the travel business process a flight activity can be related to the hotel activity in time and space. That is, the consumer may prefer the flight resource and hotel resource chosen as part of a solution to be no more than a few hours apart between the flight arrival and the start of the hotel, as well as the hotel to be as close to the airport as possible. Thus, we are interested in comparing specific pairs of flight and hotel resources to other such pairs, to determine which are better for the consumer.

We can model this type of relationship formally as a function $f_i : x_1 \times x_2 \longrightarrow \mathbb{R}^+$, where x_1 represents flights and x_2 stands for hotel room reservations, which takes the resources for each activity and gives a value of the relationship for each pair. From the optimization standpoint, it is irrelevant why a certain hotel and a certain flight give a higher value for f_i. Thus, we assume an existence of a tool that would map from the semantics of a given problem to the algebraic specification.

More generally, one can define a relationship f_j that takes as parameters any number of activities (including all of them). This allows the model to describe a wide range of physical and economic relationships between the activities of a business precess. For instance, in the travel example, it is possible to model a price discount if some specific configuration of flights and hotels were chosen (such as if they were operated by the same firm). Since cost is often a factor in the utility of a resource to the consumer, the values of such 'discount' relationships would be higher for 'same firm' resource configurations.

x_1	x_2	**utility**
1	1	5
1	2	10
1	3	2
2	1	20
2	2	16
2	3	8
3	1	1
3	2	9
3	3	11

Table 3.1: Functional representation of a binary relationship

3.4 Assumptions

The values for a given activity are discrete and the number of values is bounded. A function that relates two activities, x_1 and x_2, and assigns utilities for different resource configurations can be represented by a table as shown in Table 3.1. The function representing any relationship can be absolutely arbitrary – we make no assumptions about how they were derived and our algorithms do not make any assumptions about the function shapes.

The key concept in our model is that the consumer preferences for resources in business process activities can be described by such relationship functions. Since we allow the functions to be arbitrary and the relationships to be n-ary, these assumptions are not too restrictive.

We describe the formal models for the reservation (DSC) and the optimization (ARA) problems next.

3.5 Reservation Model

3.5.1 Properties of the Distributed System Model

A distributed system consists of physically independent computers connected by a communication network. It has three primary properties:

1. **Concurrent operation**

   All computers in the system execute concurrently.

2. **Communication infrastructure**

   Processes, running on the computers, can communicate with each other using messages. There are two primary modes of communication: synchronous and asynchronous. In synchronous communication, two processes exchange messages only if both processes are waiting synchronously for the exchange. In the asynchronous mode each computer can be thought of as having an *Inbox* and an *Outbox* where messages are queued until they are processed by the

receiving computer's logic (in the case of the Inbox) or delivered to their destination (in the case of the Outbox). The most common Internet protocol, TCP/IP, supports both synchronous and asynchronous modes of communication [11].

3. **No Global Knowledge**

   The key property of distributed systems is that no process has access to more than its own state. Specifically, no process at any time "knows" the exact state of its peers. This is a direct result of the communication delay between entities, their concurrent computation assumption and physical performance difference. What can be known is some earlier state of another process (e.g., from an earlier communication). Thus, all distributed computation is done using "old" knowledge about the states of the other computers in the system. (This will become important when we discuss the distributed atomic transactions in Chapter 8).

3.5.2 Dealing with Time

We assume that transactions between consumers and resources in our model can have temporal constraints and refer to future events. For example, a consumer can ask for the hotels available "during the next 2 weeks" and the Micro-Option right to reserve lasts for s minutes from "now." For this reason, both the resources and consumers should have access to a "universal" clock so that temporal references mean the same thing to everyone.

Earlier we have stated that entities in distributed systems do not share knowledge. A "universal" clock is knowledge shared by everyone. Though we cannot achieve a "true" universal clock, an approximate one is sufficient for our problem, as long as the clock skew does not exceed any time unit of significance. There are hardware and software solutions in existence that can guarantee very tight clock synchronization with a high degree of probability [13].

3.5.3 Instances of distributed systems

Currently, the most ubiquitous distributed system is the Internet. It consists of a myriad computers connected by a network and communicating with a specific network protocol: TCP/IP. There are numerous software packages that utilize the network and the protocol to communicate data in certain standardized formats such as *HTML*, *XML* and higher level protocols such as *HTTP* and *FTP*.

There are more advanced distributed systems that offer additional services and capabilities to many applications. One such is Object Management Group's CORBA (Common Object Request Broker Architecture) [26] that abstracts the network to client/server interactions between entities modeled using Object Oriented design principles. Another, is the Caltech Infospheres Infrastructure [11] that uses peer-to-peer communications between entities, and offers additional capabilities for designers to better compose, model and reason about distributed object systems.

3.6 Optimization Model

3.6.1 Informal Description

The search for optimal resources for a business process is an optimization problem. Posing it formally abstracts away the notions of resources and processes and allows us to concentrate on its key properties.

First, we assume that a business process consists of n activities, each of which requires one resource. Furthermore, we assume there are at most m different resources available for assignment to each activity. Thus, we model each activity as a variable x_i, $1 \leq i \leq n$ that takes values in the set $M = \{1, ..., m\}$. Namely, by $x_5 = 7$ we mean that activity #5 is assigned resource #7 out of the m available for it.

We represent the relationships between resources as functions f_i, $1 \leq i \leq k$ as described in the earlier section. A function f_i takes as parameters a set of activities and gives a value describing the user preference for each configuration of resources that could be chosen for the activities. For instance if function $f_1(x_1, x_2)$ is specified for the problem, it gives a positive real value for every assignment of resources for activities 1 and 2. It can be described as a table of size $m \times m$ rows. For example, $f_1(x_1 = 1, x_2 = 3) = 27$, $f_1(x_1 = 2, x_2 = 3) = 6$.

These relationship functions are given as input to the problem. We assume that they are generated from consumer preferences. In the business trip example, the airlines may offer cost savings to the consumer if he were to choose the same airline for all flights in the trip. This information can be translated into a function f_1 that relates all the flight resources in the business trip process. For example, if the business trip has two flights (to and from some destination) designated as activities x_1 and x_2, then the "same airline" discount can be represented as follows: $f_1(x_1 = \text{“}united\text{”}, x_2 = \text{“}united\text{”}) = 100$, $f_1(x_1 = \text{“}delta\text{”}, x_2 = \text{“}delta\text{”}) = 70$ otherwise $f_1 = 0$. This implies that $f_1(x_1 = \text{“}united\text{”}, x_2 = \text{“}delta\text{”}) = 0$.

The last part of specification is the objective function ϕ. It establishes the importance of various relationships to each other. While we do not restrict the shape of ϕ, its "topology" (the arrangement of relationship functions and operators on them) has major implications on the computational complexity of our solution method. We discuss this rigorously in section 4.1.1.

Finally, a solution to the problem is an assignment of a resource to each activity, and the optimal solution x_{opt} is an assignment that maximizes ϕ. It describes which resource needs to be assigned to which activity to get the most benefit to the consumer based on the specified preferences.

3.6.2 Formal Specification

Let $X = \{x_1, ..., x_n\}$ and $M = \{1, ..., m\}$, such that x_i takes values in M. Let $F = \{f_1, ..., f_k\}$ be a set of functions, where each $f_i(\prod_{x \in X_i}) : M^{|X_i|} \rightarrow \mathbb{R}^+$, $1 \leq i \leq k$ and $\emptyset \subset X_i \subseteq X$. Let $\phi : [\mathbb{R}^+]^k \rightarrow \mathbb{R} = \mathbf{f}[f_1, ..., f_k]$ be the global objective function.

Problem 3.1

ACTIVITY-RESOURCE ASSIGNMENT (ARA): Given X, F and ϕ find an n-vector $x_{opt} = (\dot{x}_1, ..., \dot{x}_n)$ that maximizes ϕ.

3.7 Complexity of the Problem

Our problem specification covers a subclass of the discrete non-linear optimization problems (DNLPs) most of which are NP-hard [3] [14][37]. In this section we show that our problem is at least as hard as 3-SAT. However, while it is intractable in the worst case, there are many instances of interest to us that can be solved quite quickly. When the objective function ϕ and the relationships f_i's of a problem have a certain 'topological structure,' we can use this structure to reduce the search time.

A method to do this is presented in the next chapter. In this section we show that the Activity Resource Assignment decision problem: "Is there a variable assignment that yields the value of ϕ of at least B?" is NP-complete by mapping 3-SAT into it.

3.7.1 ARA-decision is NP-complete

Theorem 3.1

The ARA decision problem is NP-complete.

Proof: First, ARA-decision $\in NP$ since given a variable assignment we can easily test in polynomial time whether $\phi \geq B$ by evaluating it. The cost to evaluate each f_i is constant and the algebraic operations on them are inexpensive.

We transform 3-SAT into ARA-decision. Let $U = \{u_1, u_2, ..., u_n\}$ be a set of variables and $C = \{c_1, c_2, ..., c_p\}$ a set of clauses where $|c_i| = 3$ for $1 \leq i \leq p$, making up an arbitrary instance of 3-SAT. The corresponding instance of ARA has $X = \{x_1, x_2, ..., x_n\}$ as variables where x_j relates to u_j and $|M| = 2$, such that each x_j takes values in the set $\{0, 1\}$. Furthermore, $\phi = \sum_{i=1}^{p} f_i$, where each $f_i : x_v \times x_w \times x_z \rightarrow \{0, 1\}$ has as parameters the variables corresponding to the variables in clause $c_i \equiv (u_v \vee u_w \vee u_z)$. The variables in c_i can appear literally or negated, where the negation is indicated by a bar above the variable name. Let each f_i be 1 if and only if at least one of its parameters x_q is 1, if the corresponding variable u_q in c_i appears literally, or x_q is 0 if u_q appears in c_i as negated. We need to show that $\phi \geq p$ if and only if the 3-SAT instance is satisfiable.

" $\Rightarrow$ " : There are p clauses in the 3-SAT problem and therefore p functions in the corresponding ARA problem. Each f_i is at most 1, thus the only way ϕ could become larger or equal to p is when all f_i's evaluate to 1. Now, the rule for the value of f_i makes it 1 only when the corresponding clause c_i evaluates to truth. When all the clauses are satisfied, their conjunction is satisfied. Thus, the problem instance is satisfiable when $\phi \geq p$.

" $\Leftarrow$ " : When a 3-SAT problem with p clauses is satisfiable, there exists an assignment of variables U such that each clause c_i evaluates to truth. By the above rule for f_i, the function becomes 1 only when its corresponding clause c_i is satisfied. Since there is a single function f_i for each clause c_i and p of them are satisfied, then $\sum_{i=1}^{p} f_i = p$. Hence $\phi = p$ when the 3-SAT instance is satisfiable.

Finally, the transformation from 3-SAT to ARA-decision is clearly polynomial, since the function corresponding to each clause is computed independently. Thus, 3-SAT can be transformed to ARA-decision. Therefore ARA-decision is NP-complete. ■

The ARA optimization is, of course, at least as hard as the decision problem. Nevertheless, some instances of ARA are easier than others and we show how to solve those relevant to our applications quickly.

Chapter 4

Optimization (Intuition)

A problem instance is a triple (X, F, ϕ), where the objective function ϕ maps all possible resource assignments for the activities of a business process, represented by variables $x_i \in X$, $1 \leq i \leq n$, into *real* utility values. For any assignment $x_{sol} = (\dot{x}_1, ..., \dot{x}_n)$, $\phi(x_{sol})$ returns a value that represents how well this configuration of resources satisfies the preferences of the consumer who specified the process.

The solution method proceeds in five stages (see Figure 4.1). First, we analyze the structure of the problem in step 1, represent it in a graph form in step 2, extract the topology by graph decomposition or a greedy heuristic in step 3, find a roadmap to the solution in step 4 and finally apply the roadmap to solve the original problem in step 5.

4.1 The Main Operator

Our goal is to find a solution that maximizes the objective function ϕ. ϕ is defined over a discrete n-space of variables in X, and computed through a set of component functions f's that take the variables as parameters. Now, we make an observation that it is possible to create a new function ϕ' with only $n-1$ variables that will share at least one maximum with ϕ if we replace the missing variable, say x_k, by a special function ρ_k that always returns the best value for x_k in ϕ for any assignment of the other variables.

For example, let $\phi = f(x_1, x_2) = \cos x_1 * \cos x_2$ for discrete x_1, x_2 (see Figure 4.2). Then we can form a $\phi' = f(x_1, \rho_{x_2}(x_1))$ such that $\max \phi = \max \phi'$, if $\rho_{x_2}(x_1)$ always returns a value for x_2 where for any x_1, $\phi(x_1, \rho_{x_2}(x_1)) \geq \phi(x_1, \dot{x}_2)$ for all assignments $\dot{x}_2$ of x_2. Then we can find a maximum of ϕ by simply iterating over all values of x_1.

Thus, we can reduce the 'complexity' of ϕ from being a function of n variables to just a function of one by systematically replacing all the variables by the appropriate ρ functions. We call ρ a *relaxation operator* and a recursive application of ρ to a problem to reduce it to its base case a *relaxation process.*

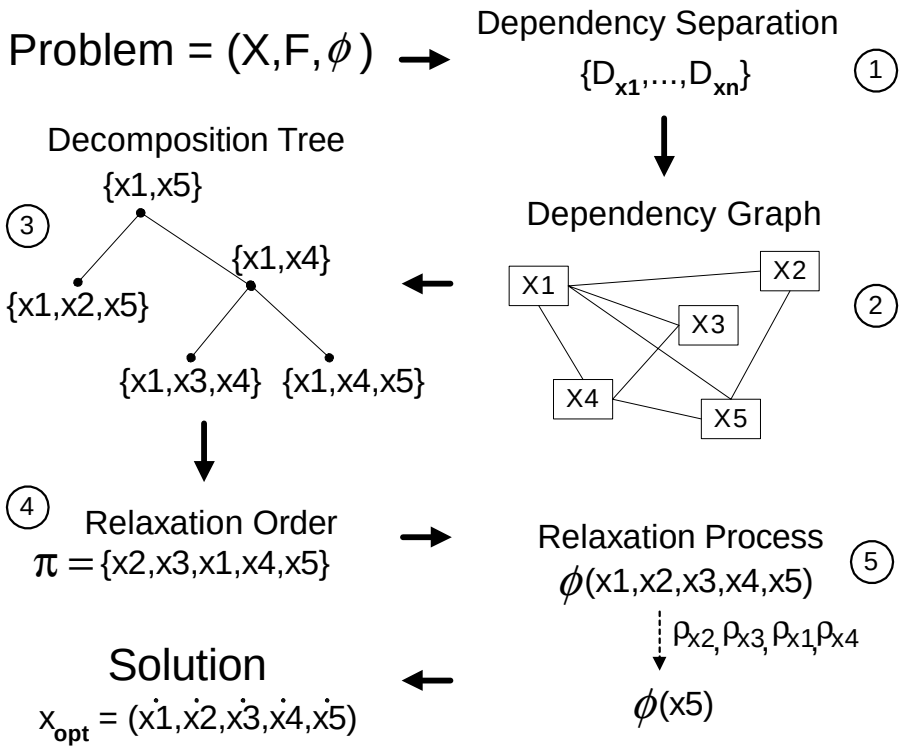


Figure 4.1: The Solution Process

Formally, the operator takes a variable $x_i \in X$ and represents it in terms of the other variables by creating a new function $\rho_{x_i} : m^{X-\{x_i\}} \rightarrow m$, where m is the number of values that each variable in X can take. Then applying it to ϕ can be written as $\phi(x_1, ..., x_i = \rho_{x_i}, ..., x_n)$, where ρ_{x_i} replaces all instances of x_i in the syntactic form of ϕ.

4.1.1 Dependency Separation

Now, it is possible to compute such ρ_{x_i} functions for our problems because all the variables in X are discrete. However, the brute-force approach would take $m^{|X|}$ time. If we make no assumptions about the shapes of the component functions f's an exhaustive search seems like the only option. Yet, if ϕ has a certain structure, for instance being a sum of products of f's, then we could do better by identifying for any given x_i a set of *dependent variables* $D_{x_i} \subseteq X$, with the property that $\rho_{x_i}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = \rho_{x_i}(\{x \,|\, x \in D_{x_i}\})$. This implies, that all the *independent variables*, $\bar{D}_{x_i} = X - D_{x_i}$, are irrelevant in finding the maxima of ϕ with respect to x_i and can therefore be 'omitted' in the search by being set to any legal values.

For the fastest computation of ρ_{x_i} we want $|D_{x_i}|$ to be as small as possible, as the cost of computing ρ_{x_i} is then $m^{|D_{x_i}|}$. To this end, we have found a set of rules to identify the dependent variables for common structures for ϕ (see Section 5.2). While, this is not always possible, for many

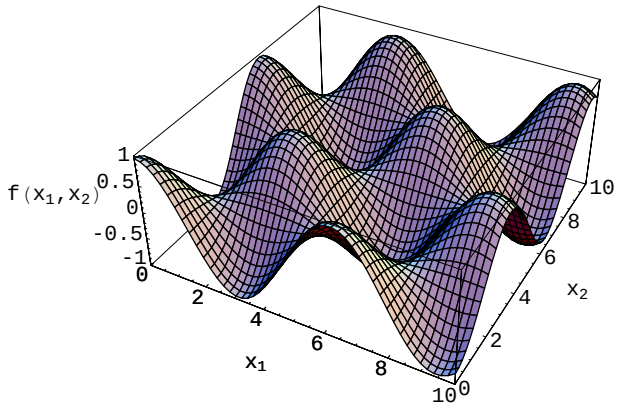


Figure 4.2: Preserving a Maximum

applications the sizes of the dependent sets are substantially smaller than n. Thus, there is a clear value in finding a good dependency separation for all variables in X and it is the first step in our optimization solution (Figure 4.1).

4.2 Dependency Graphs

Not only can the dependency separation help in reducing the computation time for the relaxation operators, there is even more important information to be found in the interaction between the dependent sets. This is interesting to us because the relaxation process can take a substantially different computation time depending on the order in which we choose to relax the variables. The difference can range from an exponential time to a low-degree polynomial. To help decide which x_i to relax when we can represent the dependencies among the variables in a *dependency graph*.

An undirected, simple graph $G = (V, E)$ is a dependency graph for an optimization problem $R = (X, F, \phi)$ if there exists a bijection between the variables X and the vertices V and there is an edge $(v_i, v_j) \in E$ if and only if x_i depends on x_j (corresponding to the vertices v_i and v_j via the bijection) or visa versa.

The purpose of the graph is to help us analyze the dependencies among variables and in particular what happens to them during a relaxation process. To do this we define a function η_{v_i} that transforms a graph G into G_{v_i}, the dependency graph after the relaxation of variable x_i. η_{v_i} is a graph dual of the relaxation operator ρ_{x_i} and it adjusts the vertices and edges in the dependency graph G after a relaxation to maintain the problem-graph correspondence. Specifically, relaxing a variable x_i removes it from X and creates a new function ρ_{x_i} which makes all the variables in D_{x_i} dependent on each other if they were not so before. Similarly, the transform η_{v_i} removes the vertex v_i from V and

connects by edges all the neighbors of v_i that were not already connected. We call this operation a *vertex elimination*. Since the neighbors of v_i always correspond to D_{x_i}, when x_i is relaxed all of its neighbors appear in a newly formed ρ_{x_i} as parameters, which creates new dependencies and therefore new edges. From this we establish an equivalence relation between the relaxation process on a problem and a vertex elimination process on its dependency graph.

Now, as stated earlier the computational complexity of generating the function ρ_{x_i} is $O(m^{|D_{x_i}|+1})$. Since each vertex in the dependency graph corresponds to a variable in X and since there is an equivalence between the relaxation and elimination processes, we can also express the cost of relaxing x_i in terms of the neighbors of v_i, N_{v_i}, namely $m^{|D_{x_i}|} = m^{|N_{v_i}|}$.

This is significant because for certain large classes of graphs it is possible to find good bounds on the total cost of a vertex elimination process, and therefore on the cost of the relaxation process on ϕ. Thus, if we eliminate vertices in the order that minimizes the cost, then the same relaxation order for variables in X would give us a bounded cost on carrying out the relaxation process (i.e. the time complexity to solve the optimization problem).

4.3 Graph Decomposition

In general, finding a polynomial relaxation order – an ordering of variables in X for a relaxation process to solve a given problem in polynomial time – for an arbitrary graph is difficult. We believe that it is likely to be NP-hard. However, we can often get a good upper bound on the cost of the relaxation process by performing a tree-decomposition on the dependency graph. A tree-decomposition yields a constant tw, called tree-width, which is a good metric for graph complexity. Tree-decomposition theory has been worked on by Tarjan[35] in the context of speeding up Gaussian elimination and later generalized by Robertson[29], Seymour[30] and others to show that many NP-hard graph problems can be solved in polynomial time for graphs with known tree-width. Here we show how to use tree-decomposition (step 3 in Figure 4.1) to guide our search for an optimal assignment.

A *graph decomposition* for a dependency graph G constructs a *decomposition tree* $\mathcal{D} = (T, \mathcal{V})$, where T is a tree and $\mathcal{V} = \{V_t\}_{t \in T}$ is a family of vertex sets $V_t \subseteq V(G)$, indexed by the vertices $t \in T$. Each node $t \in T$ corresponds to a set of vertices $V_t \equiv \mathcal{V}(t)$ in G. The tree-width tw of $\mathcal{D}$ is the number of vertices in the largest V_t.

For example, in Figure 4.1 the tree in step 3 corresponds to the graph in step 2. Its leaves contain all the vertices of G, and the internal nodes, the sets $\{x_1, x_5\}$ and $\{x_1, x_4\}$, separate the graph into cells such that there are no edges in G between the vertices from the separate cells. In this way each leaf of T corresponds to a cell in G.

This 'vertex isolation' property is exactly what is important to us for an efficient time complexity

relaxation. Because the vertices in separate cells of the graph share no edges, we can eliminate all the vertices within a cell without creating any edges other than to the other vertices in the same cell. In relation to relaxation, this means that if we relax the variables corresponding to the vertices in a cell, then all the new dependencies created will be to other variables within the same cell.

This property allows us to bound the cost of eliminating the vertices and correspondingly relaxing the variables to solve our optimization problem to at most $O(\phi_c * n * m^{tw})$, where ϕ_c is the number of algebraic operations in ϕ, and the relaxation of each of n variables takes at most m^{tw}.

There are a number of algorithms to decompose graphs into cells with this property [15],[27]. A clique-decomposition method due to Tarjan[35] with a later improvement by Leimer[22] can separate certain types of graphs (for example chordal graphs) in polynomial time or report that they are not separable. Also, Matausek[23] gave a fast method to identify and separate graphs with tree widths ≤ 3. A summary of known decomposition algorithms and graphs they can separate can be found in Table 6.1.

Graph decomposition remains a hot area of research in the graph complexity community, however we have found that the existing algorithms work well to decompose the dependency graphs of many cases that we simulated that we consider will be typical in practice.

Now, the last step to do before carrying out the relaxation process is translating a tree decomposition into a relaxation order that can give a polynomial solution to the optimization problem.

4.4 Relaxation Order

Once we have a tree decomposition of the dependency graph, we can immediately find a relaxation order with the following algorithm:

Input: Let $\mathcal{D} =(T, \mathcal{V})$ be a tree decomposition of G. Then,

1. Find a vertex v that appears in exactly one leaf V_t and label it as next (starting at 1).
2. Remove v from V_t; set $V_t := V_t - \{v\}$ and if $V_t - \{v\}$ exactly equals to its parent in T (the separator of V_t in the decomposition) then remove the leaf V_t from $\mathcal{V}$.
3. If $\mathcal{V}$ is not empty, repeat step 1.

Output: a vertex ordering π.

Intuitively, the algorithm constructs an ordering such that when a vertex is chosen for elimination (its corresponding variable for relaxation), it appears in exactly one leaf of the decomposition tree. In this way, we effectively "dissolve" all the cells of a graph (corresponding to the leaves) from outside in, by eliminating all the vertices inside each cell.

4.5 Order by Heuristic

An alternative to tree-decomposition to find a relaxation order is to use a vertex selection heuristic that somehow chooses the order of vertices for elimination. We experimented with several heuristics (see chapter 7) that rely on vertex degrees to choose the vertices for elimination at each step of the relaxation process. The results are surprisingly good, giving computation costs similar to those from tree-decomposition for simpler problem instances and even beating them for many complex instances.

4.6 Carrying out the Optimization – Applying the Relaxation Process

Given a variable ordering π a relaxation process can directly compute the ρ functions in the specified order. The variable $x \in X$ with largest assigned number in π is the last variable in ϕ. To find a max of ϕ with respect to its only remaining variable x can be done by iterating over all m of its values and computing the value of ϕ for each. The assignment of x that yields that largest value of ϕ determines the rest of the variables. This is an optimal solution to the optimization problem because each ρ_x preserves a maximum of the objective function.

4.7 A Complete Solution to the Example Problem

The following is a solution to the Example problem 1.1 using the outlined method. For convenience we restate the problem:

Problem 4.1

Given X, F *and* ϕ *where* $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$,

$$F = \begin{cases} f_1 : x_1 \times x_2 \times x_3 \to \mathbb{R}^+ \\ f_2 : x_1 \times x_3 \to \mathbb{R}^+ \\ f_3 : x_4 \times x_5 \to \mathbb{R}^+ \\ f_4 : x_1 \times x_2 \times x_6 \to \mathbb{R}^+ \\ f_5 : x_3 \times x_4 \times x_5 \to \mathbb{R}^+ \end{cases}$$

and $\phi = f_1^{f_2} + f_5 f_3 + f_4$

Find an n-vector $x_{opt} = (\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{x}_4, \dot{x}_5, \dot{x}_6)$ *that maximizes* ϕ.

Solution: We solve the problem using the five steps outlined above:

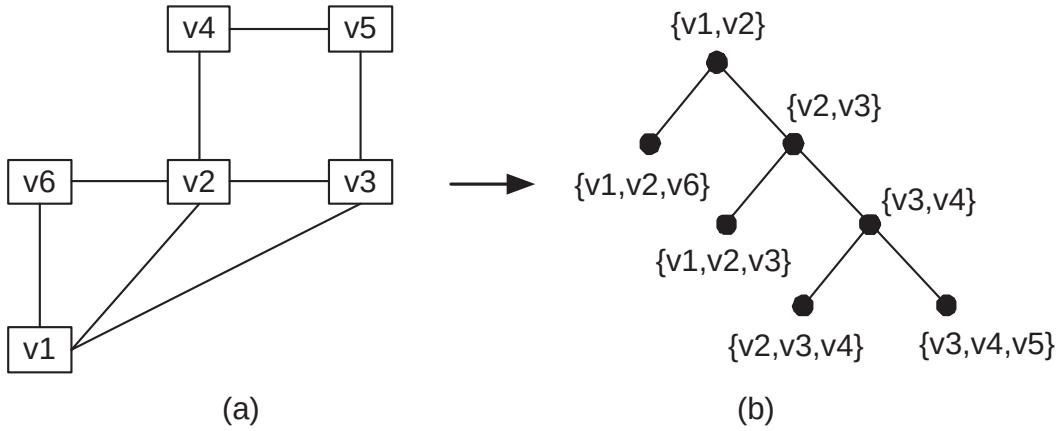


Figure 4.3: Example Problem Dependency Graph

1. Dependency Separation

   $D_{x_1} = \{x_2, x_3, x_6\}$,

   $D_{x_2} = \{x_1, x_3, x_4, x_6\}$,

   $D_{x_3} = \{x_1, x_2, x_5\}$,

   $D_{x_4} = \{x_2, x_5\}$,

   $D_{x_5} = \{x_3, x_4\}$,

   $D_{x_6} = \{x_1, x_2\}$

   For example, the dependent set of x_1 is $\{x_2, x_3, x_6\}$ because f_1 has x_1 as parameter along with x_2 and x_3; and f_4 has it with x_6.

2. Dependency Graph

   See Figure 4.3 (a). Here the corresponding vertex v_1 has edges to v_2, v_3 and v_6.

3. Graph Decomposition

   The graph in (a) can be decomposed into (b) such that the leaves are the cells of the graph and the internal nodes are the separators. For example, $\{v_1, v_2\}$ separates v_6 from the rest of the graph. There are no edges from v_6 to any vertices outside of its parent separator.

4. Relaxation Order

   For the graph from Step 3, we can assign the order as follows: 1) v_5 appears only in one leaf, the $\{v_3, v_4, v_5\}$, so we can assign the #1 to v_5. After the assignment, we remove v_5 from $\{v_3, v_4, v_5\}$ and observe the new leaf $\{v_3, v_4\}$ is the same as its parent separator. From step 2 in the algorithm we then remove it from the tree.

   Next, v_6 appears only in one leaf, $\{v_1, v_2, v_6\}$, so we assign it the #2. And so on, to get the final solution sequence $\pi = \{v_5, v_6, v_1, v_4, v_2, v_3\}$.

5. Applying the relaxation order π to ϕ

$\phi_0 = f_1(x_1, x_2, x_3)^{f_2(x_1,x_3)} + f_5(x_3, x_4, x_5) f_3(x_4, x_5) + f_4(x_1, x_2, x_6)$

relaxing $x_5 \Longrightarrow \max \phi_1 = \max \phi_0 \mid x_5 = \rho_{x_5}(x_3, x_4)$ with cost m^3

$\phi_1 = f_1(x_1, x_2, x_3)^{f_2(x_1,x_3)} + f_5(x_3, x_4, \rho_{x_5}(x_3, x_4)) f_3(x_4, \rho_{x_5}(x_3, x_4)) + f_4(x_1, x_2, x_6)$

relaxing $x_6 \Longrightarrow \max \phi_2 = \max \phi_1 \mid x_6 = \rho_{x_6}(x_1, x_3)$ with cost m^3

$\phi_2 = f_1(x_1, x_2, x_3)^{f_2(x_1,x_3)} + f_5(x_3, x_4, \rho_{x_5}(x_3, x_4)) f_3(x_4, \rho_{x_5}(x_3, x_4)) + f_4(x_1, x_2, \rho_{x_6}(x_1, x_2))$

relaxing $x_1 \Longrightarrow \max \phi_3 = \max \phi_2 \mid x_1 = \rho_{x_1}(x_2, x_3)$ with cost m^3

$\phi_3 = f_1(\rho_{x_1}(x_2, x_3), x_2, x_3)^{f_2(\rho_{x_1}(x_2,x_3),x_3)} +$

$+ f_5(x_3, x_4, \rho_{x_5}(x_3, x_4)) f_3(x_4, \rho_{x_5}(x_3, x_4)) + f_4(\rho_{x_1}(x_2, x_3), x_2, \rho_{x_6}(\rho_{x_1}(x_2, x_3), x_2))$

relaxing $x_4 \Longrightarrow \max \phi_4 = \max \phi_3 \mid x_4 = \rho_{x_4}(x_2, x_3)$ with cost m^3

$\phi_4 = f_1(\rho_{x_1}(x_2, x_3), x_2, x_3)^{f_2(\rho_{x_1}(x_2,x_3),x_3)} +$

$+ f_5(x_3, \rho_{x_4}(x_2, x_3), \rho_{x_5}(x_3, \rho_{x_4}(x_2, x_3))) f_3(\rho_{x_4}(x_2, x_3), \rho_{x_5}(x_3, \rho_{x_4}(x_2, x_3)))$

$+ f_4(\rho_{x_1}(x_2, x_3), x_2, \rho_{x_6}(\rho_{x_1}(x_2, x_3), x_2))$

relaxing $x_2 \Longrightarrow \max \phi_5 = \max \phi_4 \mid x_2 = \rho_{x_4}(x_3)$ with cost m^2

$\phi_5 : x_3 \to \mathbb{R}^+$: base case with cost m. done.

Total computing: $4 * (4 * m^3 + m^2 + m)$ time, because $\phi_c = 4$.

Now, $x_{opt} = (\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{x}_4, \dot{x}_5, \dot{x}_6)$ where

$\dot{x}_1 = \rho_{x_1}(\rho_{x_4}(\rho_{x_3}()), \rho_{x_3}())$

$\dot{x}_2 = \rho_{x_4}(\rho_{x_3}())$

$\dot{x}_3 = \rho_{x_3}()$

$\dot{x}_4 = \rho_{x_4}(\rho_{x_4}(\rho_{x_3}()), \rho_{x_3}())$

$\dot{x}_5 = \rho_{x_5}(\rho_{x_3}(), \rho_{x_4}(\rho_{x_4}(\rho_{x_3}()), \rho_{x_3}()))$

$\dot{x}_6 = \rho_{x_6}(\rho_{x_1}(\rho_{x_4}(\rho_{x_3}()), \rho_{x_3}()), \rho_{x_3}())$

computed from the above derivation.

Chapter 5

Optimization via Relaxation

5.1 The Main Operator

Let $X = \{x_1, ..., x_n\}$ and $M = \{1, ..., m\}$, such that x_i takes values in M for all $1 \leq i \leq n$. Let $F = \{f_1, ..., f_k\}$ be a set of functions, where each $f_i(\prod_{x \in X_i}) : M^{|X_i|} \rightarrow \mathbb{R}^+$, $1 \leq i \leq k$ and $\emptyset \subset X_i \subseteq X$. Let $\phi : [\mathbb{R}^+]^k \rightarrow \mathbb{R} = \mathbf{f}[f_1, ..., f_k]$ be the global objective function.

Definition 5.1 (Variable Assignment)
*Let $S \subseteq X$. We define $\alpha(S) : \prod S \rightarrow M^{|S|}$ to be an **assignment** of S, which chooses a value for each variable of S. We also use the notation, $s = \alpha(S, x_i = \dot{x}_i)$, where $\dot{x}_i$ is a value of x_i, to represent an assignment of variables in S with x_i necessarily set to $\dot{x}_i$.*

Definition 5.2 (Relaxation Function)
Let $x \in X$ and $S = X - \{x\}$. We define a relaxation function

$$\rho_x(S) : M^{n-1} \rightarrow M \triangleq \dot{x} \text{ such that } \forall \ddot{x} \in M : \phi(\ddot{x}, S) \leq \phi(\dot{x}, S) \tag{5.1}$$

that returns an optimal value of x (that maximizes ϕ) for any assignment of S. When the parameter S is $X - \{x\}$ we omit it from the argument, using simply ρ_x. We also use $\rho_x(\dot{x}_i \cup s)$ to mean an assignment of $s = \alpha(S, x_i = \dot{x}_i)$ as parameter to ρ_x.

Notation 5.1
We write $\phi(x_i = \rho_{x_i})$ to represent the function ϕ where every appearance of x_i was replaced by the expression ρ_{x_i}.

Definition 5.3 (Independence)
*Let $x_i \neq x_k \in X$ and $S = X - \{x_i, x_k\}$. Then, x_i is **independent** of x_k if and only if*

$$\forall s = \alpha(S) : \forall \dot{x}_k, \ddot{x}_k \in M : \phi(x_i = \rho_{x_i}(\dot{x}_k \cup s), \dot{x}_k, s) = \phi(x_i = \rho_{x_i}(\ddot{x}_k \cup s), \dot{x}_k, s) \tag{5.2}$$

otherwise, it is said to be ***dependent*** *on* x_k.

The definition of independence states that the two values $\dot{x}_i = \rho_{x_i}(\dot{x}_k \cup s)$ and $\ddot{x}_i = \rho_{x_i}(\ddot{x}_k \cup s)$ can be different (which can happen when ϕ has more than one maximum), but as assignments for x_i they both produce the same value of ϕ. Also, given an $x \in X$, all the variables in $X - \{x\}$ are divided into dependent and independent sets with respect to x.

Definition 5.4 (Independence Property)
We denote the set of dependent variables of $x \in X$ *as* D_x *and the set of independent ones* $\bar{D}_x$. *Then the following is an invariant:*

$$X = D_x \cup \bar{D}_x \cup \{x\} \wedge D_x \cap \bar{D}_x = \emptyset \wedge x \notin D_x \wedge x \notin \bar{D}_x. \tag{5.3}$$

5.1.1 The Property of ρ

The main property of the relaxation operator ρ is that regardless of which variable $x \in X$ is chosen, replacing x by ρ_x in ϕ preserves at least one of its maximal solutions (albeit at the expense of the other maxima).

Theorem 5.1
$\max \phi = \max \phi(x_i = \rho_{x_i}, s)$, *where* s *is an arbitrary assignment of all other variables* $X - \{x_i\}$.

Proof: Suppose not. Then there exists an $x_{opt} = (\dot{x}_1, ..., \dot{x}_i, ..., \dot{x}_n)$ such that $\phi(x_{opt}) = \max \phi$ and a corresponding $x_{sol} = (\dot{x}_1, ..., x_i = \rho_{x_i}(s), ..., \dot{x}_n)$ with $s = (\dot{x}_1, ..., \dot{x}_{i-1}, \dot{x}_{i+1}, ..., \dot{x}_n)$ where $\phi(x_i = \rho_{x_i}) \equiv \phi(x_{sol})$. Then, $\phi(x_{opt}) > \phi(x_{sol})$. Now, from the definition of ρ in 5.1 we have $\forall \ddot{x} \in M : \phi(\dot{x}_1, ..., x_i = \ddot{x}, ..., \dot{x}_n) \leq \phi(\dot{x}_1, ..., x_i = \rho_{x_i}(s), ..., \dot{x}_n)$. In particular, this implies that $\phi(\dot{x}_1, ..., \dot{x}_i, ..., \dot{x}_n) \leq \phi(\dot{x}_1, ..., x_i = \rho_{x_i}(s), ..., \dot{x}_n) \equiv \phi(x_{opt}) \leq \phi(x_{sol})$. Contradiction. ■

5.1.2 The Relaxation Process

Definition 5.5 (Ordering, Ordered Structure)
For a structure $R = (X, F, \phi)$ *an* ***ordering*** *of* X *is a bijection* $\pi : \{1, 2, ..., n\} \leftrightarrow X$ *and we call* R_π *an* ***ordered structure***. *Thus,* $x_i \in X$ *in* R_π *is the* i^{th} *variable with respect to ordering* π.

Definition 5.6 (X-Relaxation)
The relational structure R_x *obtained from* R *by replacing every instance of variable* x *in* ϕ *with the expression* ρ_x *is an* ***x-relaxation*** *of* R *represented by*

$$R_x = (X - \{x\},\ F \cup \rho_x,\ \phi(x = \rho_x)) \tag{5.4}$$

Definition 5.7 (Relaxation Process)

For an ordered relational structure R_π we define a relaxation process

$$P(R_\pi) = [R = R_0, R_1, ..., R_n] \tag{5.5}$$

as a sequence of relaxation structures defined recursively by $R_0 = R,\ R_i = (R_{i-1})_{x_i}$ for $1 \leq i \leq n$.

Theorem 5.2

A $P(R_\pi)$ finds an optimal solution to the optimization problem.

Proof: Induction on i, applying x-relaxation to each x in order π. ■

Now we have a deterministic process for finding an optimal solution. However, we also want to be able to find it quickly. Next we show how to compute the relaxation functions ρ fast for certain common problem topologies.

5.1.3 Computing ρ

The main implication of the dependency property 5.2 for a given x is that the value returned by ρ_x, the "best" value of x_i for each assignment of the other variables, is the same regardless of the values of **all** the independent variables $\bar{D}_x$. This is shown in the following theorem:

Theorem 5.3

Let $x \in X$ and $u = \alpha(D_x)$ be some assignment of the dependent variables of x. Then,

$$\forall v = \alpha(\bar{D}_x) \wedge v' = \alpha(\bar{D}_x), v \neq v' : \phi(x = \rho_x(u \cup v), u, v) = \phi(x = \rho_x(u \cup v'), u, v) \tag{5.6}$$

Proof: Let $x_i, x_j \in \bar{D}_x$ be two variables independent with respect to x. First, we show that these variables are together independent of x. That is, the independence property 5.2 holds for each one with respect to x regardless of the value chosen for the other. Namely, from the definition of independence, we have

$$\begin{aligned} &\forall s = \alpha(X - \{x, x_i\}) : \forall \dot{x}_i, \ddot{x}_i \in M : \\ &\phi(x = \rho_x(\dot{x}_i \cup s), \dot{x}_i, s) = \phi(x = \rho_x(\ddot{x}_i \cup s), \dot{x}_i, s) \end{aligned} \tag{5.7a}$$

and

$$\begin{aligned} &\forall s = \alpha(X - \{x, x_j\}) : \forall \dot{x}_j, \ddot{x}_j \in M : \\ &\phi(x = \rho_x(\dot{x}_j \cup s), \dot{x}_j, s) = \phi(x = \rho_x(\ddot{x}_j \cup s), \dot{x}_j, s) \end{aligned} \tag{5.7b}$$

We want to show that:

$$\begin{aligned}&\forall s = \alpha(X - \{x, x_i, x_j\}) : \forall \dot{x}_i, \ddot{x}_i, \dot{x}_j, \ddot{x}_j \in M : \\ &\phi(x = \rho_x(\dot{x}_i, \dot{x}_j \cup s), \dot{x}_i, \dot{x}_j, s) = \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j \cup s), \dot{x}_i, \dot{x}_j, s)\end{aligned} \tag{5.7c}$$

Suppose not. Let $s' = \alpha(X - \{x, x_i, x_j\})$ be an assignment and $\dot{x}_i, \ddot{x}_i, \dot{x}_j, \ddot{x}_j$ be the four values such that $\phi(x = \rho_x(\dot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \neq \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s')$ (negation of 5.7c). For this assignment, we have

$$\phi(x = \rho_x(\dot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') = \phi(x = \rho_x(\ddot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \tag{5.7d}$$

and

$$\phi(x = \rho_x(\dot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') = \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \tag{5.7e}$$

from 5.7a. This is directly from the definition of independence 5.2, where $s = \alpha(X - \{x, x_i\}) = \{\dot{x}_j \cup s'\}$ and $s = \{\ddot{x}_j \cup s'\}$ respectively. Similarly, we also have

$$\phi(x = \rho_x(\dot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') = \phi(x = \rho_x(\dot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \tag{5.7f}$$

and

$$\phi(x = \rho_x(\ddot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') = \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \tag{5.7g}$$

from 5.7b. Also directly from independence, where $s = \alpha(X - \{x, x_j\}) = \{\dot{x}_i \cup s'\}$ and $s = \{\ddot{x}_i \cup s'\}$ respectively. Then,

$$\begin{aligned}& \phi(x = \rho_x(\dot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \\ = \;& \phi(x = \rho_x(\ddot{x}_i, \dot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \\ = \;& \phi(x = \rho_x(\dot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s') \\ = \;& \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j \cup s'), \dot{x}_i, \dot{x}_j, s')\end{aligned} \tag{5.7h}$$

Contradiction. And so, we have 5.7c. Now, we simply induct on the number of variables in $\bar{D}_x$. For example, for three variables $x_i, x_j, x_k \in \bar{D}_x$ and $s' = \alpha(X - \{x, x_i, x_j, x_k\})$ the equality is derived as follows:

$$\begin{aligned}& \phi(x = \rho_x(\dot{x}_i, \dot{x}_j, \dot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \\ = \;& \phi(x = \rho_x(\ddot{x}_i, \dot{x}_j, \dot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \\ = \;& \phi(x = \rho_x(\dot{x}_i, \ddot{x}_j, \dot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \\ = \;& \phi(x = \rho_x(\dot{x}_i, \dot{x}_j, \ddot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s')\end{aligned} \tag{5.7i}$$

and

$$\begin{aligned} & \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j, \ddot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \\ = \; & \phi(x = \rho_x(\dot{x}_i, \ddot{x}_j, \ddot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \\ = \; & \phi(x = \rho_x(\ddot{x}_i, \dot{x}_j, \ddot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \\ = \; & \phi(x = \rho_x(\ddot{x}_i, \ddot{x}_j, \dot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \end{aligned} \tag{5.7j}$$

immediately from the independence properties for each variable. Finally, we also have

$$\phi(x = \rho_x(\dot{x}_i, \ddot{x}_j, \ddot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') = \phi(x = \rho_x(\dot{x}_i, \ddot{x}_j, \dot{x}_k \cup s'), \dot{x}_i, \dot{x}_j, \dot{x}_k, s') \tag{5.7k}$$

from the 2-variable case, which links 5.7i and 5.7j. Thus, they are all equal, contradicting the dependence assumption and by induction the theorem holds.

■

Now we are allowed to use the ρ functions computed only from the dependent variables instead of the ρ over all the variables everywhere in the relaxation process above. This is what allows us to gain the first type of substantial savings in solving the optimization problem via the relaxation process.

5.2 Dependency Separation

The relaxation process is predicated on our ability to correctly separate X into the dependent and independent sets with respect to a given variable. We now give some exact methods for separating the variables for the following forms of the objective function ϕ:

1. $\phi =$ a sum of f's or a product of f's.
2. $\phi =$ sum of products, product of sums
3. $\phi =$ sum of exponents, exponent of sums
4. $\phi =$ product of exponents, exponent of products

To prove that each separation is correct, we need to show that the results satisfy the independence property 5.2. First, some notation:

Notation 5.2

We use the syntax $f_i(A)[B]$ for some $A, B \subseteq X$ to represent a function in F that has variables in A as parameters and may have any variables in B as additional parameters. For example, $f_i(x_j)[x_k, S]$ means that f_i has at least x_j as parameter and may also have any variables in $\{x_k\} \cup S$ as additional parameters. The statement $f_i(\dot{x}_j)[\ddot{x}_k, s]$ denotes a function evaluation for the corresponding

$f_i(x_j)[x_k, S]$ where $\dot{x}_j$ and $\ddot{x}_k$ are assignments of x_j and x_k respectively and s is some assignment of S.

Notation 5.3

$\sum_{f_i \in C} f_i(A)[B]$ where $A, B \subseteq X$ and $C \subseteq F$ denotes a sum of functions each having at least the variables in A as parameters. However, each function may have different subsets of B as additional parameters. For example, $\sum_{f_i \in C} f_i(x_1)[x_2, x_3, x_4]$ can represent the expression $f_1(x_1) + f_2(x_1, x_3) + f_3(x_1, x_4)$. Similarly, the expression $\sum \prod_{i=1}^{n} f_i(A)[B]$ represents a sum of products of such functions.

Lemma 5.1

Let $\phi = \sum_{i=1}^{n} f_i$ and let $x_i \neq x_j \in X$. Then x_i is independent of x_j if there does not exist an $X_t, 1 \leq t \leq k$, such that $x_i, x_j \in X_t$.

Proof: Let $x \in X$ and let $\bar{D}_x$ be formed by the above rule. That is,

$$\forall x_{ind} \in \bar{D}_x : \nexists X_t : x, x_{ind} \in X_t \text{ for } 1 \leq t \leq k. \tag{5.8a}$$

Suppose the opposite is true. Then there exists an $x_{ind} \in \bar{D}_x$ that fails to satisfy the independence property 5.2. Namely, the following is true (negation of independence): Let $S = X - \{x, x_{ind}\}$

$$\exists\, s = \alpha(S) : \exists\, \dot{x}_{ind}, \ddot{x}_{ind} \in M : \phi(x = \rho_x(\dot{x}_{ind} \cup s), \dot{x}_{ind}, s) < \phi(x = \rho_x(\ddot{x}_{ind} \cup s), \dot{x}_{ind}, s) \tag{5.8b}$$

which in this case implies that for these s, $\dot{x}_{ind}$ and $\ddot{x}_{ind}$, if we let $\dot{x} = \rho_x(\dot{x}_{ind} \cup s)$, $\ddot{x} = \rho_x(\ddot{x}_{ind} \cup s)$ then we have

$$\sum_{i=1}^{n} f_i[\dot{x}, \dot{x}_{ind}, s] < \sum_{i=1}^{n} f_i[\ddot{x}, \dot{x}_{ind}, s] \tag{5.8c}$$

directly from 5.8b, replacing the general ϕ by the assumed structure $\phi = \sum_{i=1}^{n} f_i$. Now, the independent set formation rule 5.8a allows us to separate 5.8c as follows:

$$\sum_{f_i \in E_x} f_i[\dot{x}, s] + \sum_{f_i \in \bar{E}_x} f_i[\dot{x}_{ind}, s] < \sum_{f_i \in E_x} f_i[\ddot{x}, s] + \sum_{f_i \in \bar{E}_x} f_i[\dot{x}_{ind}, s] \tag{5.8d}$$

where $f_i \in E_x$ iff $x \in X_i$; otherwise $f_i \in \bar{E}_x$. With some algebraic manipulation this simplifies to

$$\sum_{f_i \in E_x} f_i[\dot{x}, s] < \sum_{f_i \in E_x} f_i[\ddot{x}, s] \tag{5.8e}$$

On the other hand, from the definition of ρ_x we have

$$\forall x' \in M : \phi(x', \dot{x}_{ind}, s) \leq \phi(\dot{x}, \dot{x}_{ind}, s) \text{ and } \forall x' \in M : \phi(x', \ddot{x}_{ind}, s) \leq \phi(\ddot{x}, \ddot{x}_{ind}, s) \tag{5.8f}$$

which can be expanded and separated as earlier from 5.8a and the structure of ϕ into:

$$\begin{aligned}&\forall x' \in M : \textstyle\sum_{f_i \in E_x} f_i[x', s] + \sum_{f_i \in \bar{E}_x} f_i[\dot{x}_{ind}, s] \leq \sum_{f_i \in E_x} f_i[\dot{x}, s] + \sum_{f_i \in \bar{E}_x} f_i[\dot{x}_{ind}, s] \\ &\equiv \forall x' \in M : \textstyle\sum_{f_i \in E_x} f_i[x', s] \leq \sum_{f_i \in E_x} f_i[\dot{x}, s]\end{aligned} \tag{5.8g}$$

and

$$\begin{aligned}&\forall x' \in M : \textstyle\sum_{f_i \in E_x} f_i[x', s] + \sum_{f_i \in \bar{E}_x} f_i[\ddot{x}_{ind}, s] \leq \sum_{f_i \in E_x} f_i[\ddot{x}, s] + \sum_{f_i \in \bar{E}_x} f_i[\ddot{x}_{ind}, s] \\ &\equiv \forall x' \in M : \textstyle\sum_{f_i \in E_x} f_i[x', s] \leq \sum_{f_i \in E_x} f_i[\ddot{x}, s]\end{aligned} \tag{5.8h}$$

Combining 5.8g and 5.8h we have:

$$\sum_{f_i \in E_x} f_i[\dot{x}, s] = \sum_{f_i \in E_x} f_i[\ddot{x}, s] \tag{5.8i}$$

And, from 5.8e and 5.8i we have a contradiction. Thus, the independent variables of x identified for $\phi = \sum_{i=1}^{n} f_i$ via the parameter overlap rule are indeed independent.

■

Lemma 5.2

Let $\phi = \prod_{i=1}^{n} f_i$ and let $x_i \neq x_j \in X$. Then x_i is independent of x_j if there does not exist an $X_t, 1 \leq t \leq k$, such that $x_i, x_j \in X_t$.

Proof: The proof has the same structure as for lemma 5.1, except equations 5.8d, 5.8g and 5.8h have multiplication instead of addition. ■

Lemma 5.3

Let $\phi = \sum \prod_{i=1}^{n} f_i$ and let $x \neq x_j \in X$. Then x is independent of x_j if there does not exist an $X_t, 1 \leq t \leq k$, such that $x, x_j \in X_t$ ***AND*** *ϕ cannot be written as*

$$\phi = \prod_{f_i \in B} f_i(x)[S] * \prod_{f_i \in A} f_i(x_j)[S] + \prod_{f_i \in D} f_i(x)[S] * \sum \prod_{f_i \in C} f_i(x_j)[S] + \sum \prod_{f_i \in E} f_i[x, x_j, S] \tag{5.9}$$

where $A, B, C, D, E \subseteq F$ and $S = X - \{x, x_j\}$ and sets C and E could be empty.

Proof: If there is no function in F that has both x and x_j as parameters, then a ϕ of the form $\sum \prod_{i=1}^{n} f_i$ can take only be of the following three forms:

a) $\prod_{f_i \in A} f_i(x)[S] + \sum \prod_{f_i \in B} f_i(x_j)[S]$ for some $A, B \subseteq F$

b) $\prod_{f_i \in A} f_i(x)[S] * \prod_{f_i \in B} f_i(x_j)[S] + \sum \prod_{f_i \in E} f_i[x_j, S]$

c) and the form given in equation 5.9.

We will show that if ϕ can be written as the types a) or b) then x is independent of x_j. As previously, the argument is by contradiction. Suppose the opposite is true. Then there exists some ϕ which is not of the form 5.9 and an $x \in X$ with $\bar{D}_x$ formed by the above rule. That is,

$$\forall x_{ind} \in \bar{D}_x : \nexists X_t : x, x_{ind} \in X_t \text{ for } 1 \leq t \leq k. \wedge \text{ NOT eq. 5.9 with } x_{ind} \text{ replacing } x_j \tag{5.10a}$$

Where there exists an $x_{ind} \in \bar{D}_x$ that fails to satisfy the independence property 5.2. Namely, the following is true (negation of independence): Let $S = X - \{x, x_{ind}\}$

$$\exists\, s = \alpha(S) : \exists\, \dot{x}_{ind}, \ddot{x}_{ind} \in M : \phi(x = \rho_x(\dot{x}_{ind} \cup s), \dot{x}_{ind}, s) < \phi(x = \rho_x(\ddot{x}_{ind} \cup s), \dot{x}_{ind}, s) \tag{5.10b}$$

which in this case implies that for these s, $\dot{x}_{ind}$ and $\ddot{x}_{ind}$, if we let $\dot{x} = \rho_x(\dot{x}_{ind} \cup s)$, $\ddot{x} = \rho_x(\ddot{x}_{ind} \cup s)$ then we have

$$\sum \prod f_i[\dot{x}, \dot{x}_{ind}, s] < \sum \prod f_i[\ddot{x}, \dot{x}_{ind}, s] \tag{5.10c}$$

Now, if ϕ is of type a) then we can rewrite 5.10c as:

$$\prod_{f_i \in A} f_i(\dot{x})[s] + \sum \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] < \prod_{f_i \in A} f_i(\ddot{x})[s] + \sum \prod_{f_i \in B} f_i(\dot{x}_{ind})[s]$$

for some $A, B \subseteq F$, leading to

$$\prod_{f_i \in A} f_i(\dot{x})[s] < \prod_{f_i \in A} f_i(\ddot{x})[s] \tag{5.10d}$$

or if ϕ is of type b) as:

$$\prod_{f_i \in A} f_i(\dot{x})[s] * \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] + \sum \prod_{f_i \in E} f_i[\dot{x}_{ind}, s] < \prod_{f_i \in A} f_i(\ddot{x})[s] * \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] + \sum \prod_{f_i \in E} f_i[\dot{x}_{ind}, s]$$

for some $A, B, E \subseteq F$ leading, again to equation 5.10d via an algebraic reduction.

On the other hand, from the definition of ρ_x we have

$$\forall x' \in M : \phi(x', \dot{x}_{ind}, s) \leq \phi(\dot{x}, \dot{x}_{ind}, s) \text{ and } \forall x' \in M : \phi(x', \ddot{x}_{ind}, s) \leq \phi(\ddot{x}, \ddot{x}_{ind}, s) \tag{5.10e}$$

which can be expanded and separated as follows: if the structure of ϕ is a) then we have

$$\begin{aligned} &\forall x' \in M : \textstyle\prod_{f_i \in A} f_i(x')[s] + \sum \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] \leq \prod_{f_i \in A} f_i(\dot{x})[s] + \sum \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] \\ &\equiv \forall x' \in M : \textstyle\prod_{f_i \in A} f_i(x')[s] \leq \prod_{f_i \in A} f_i(\dot{x})[s] \end{aligned}$$

and

$$\forall x' \in M : \prod_{f_i \in A} f_i(x')[s] + \sum \prod_{f_i \in B} f_i(\ddot{x}_{ind})[s] \leq \prod_{f_i \in A} f_i(\ddot{x})[s] + \sum \prod_{f_i \in B} f_i(\ddot{x}_{ind})[s]$$
$$\equiv \forall x' \in M : \prod_{f_i \in A} f_i(x')[s] \leq \prod_{f_i \in A} f_i(\ddot{x})[s]$$

and combining the two we obtain

$$\prod_{f_i \in A} f_i(\dot{x})[s] = \prod_{f_i \in A} f_i(\ddot{x})[s] \tag{5.10f}$$

which contradicts the equation 5.10d derived earlier.

Similarly, when ϕ is of type b) we have

$$\forall x' \in M : \prod_{f_i \in A} f_i(x')[s] * \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] + \sum \prod_{f_i \in E} f_i[\dot{x}_{ind}, s]$$
$$\leq \prod_{f_i \in A} f_i(\dot{x})[s] * \prod_{f_i \in B} f_i(\dot{x}_{ind})[s] + \sum \prod_{f_i \in E} f_i[\dot{x}_{ind}, s]$$
$$\equiv \forall x' \in M : \prod_{f_i \in A} f_i(x')[s] \leq \prod_{f_i \in A} f_i(\dot{x})[s]$$

and

$$\forall x' \in M : \prod_{f_i \in A} f_i(x')[s] * \prod_{f_i \in B} f_i(\ddot{x}_{ind})[s] + \sum \prod_{f_i \in E} f_i[\ddot{x}_{ind}, s]$$
$$\leq \prod_{f_i \in A} f_i(\ddot{x})[s] * \prod_{f_i \in B} f_i(\ddot{x}_{ind})[s] + \sum \prod_{f_i \in E} f_i[\ddot{x}_{ind}, s]$$
$$\equiv \forall x' \in M : \prod_{f_i \in A} f_i(x')[s] \leq \prod_{f_i \in A} f_i(\ddot{x})[s]$$

yielding the same contradiction via equation 5.10f.

Thus, we have a contradiction for types a) and b) of ϕ and the theorem holds.

■

The proofs for the sums of exponents and products of exponents follow the steps of the above lemma. For the product of exponents, we merely need to take the log function of both sides.

Now, with the dependency sets we can proceed to represent them in graph form for the next step in our optimization solution.

5.3 Dependency Graphs

Definition 5.8 (Dependency Graph)

Let $G = (V, E)$ be an undirected simple graph with V the set of vertices and E the set of edges. Let $R = (X, F, \phi)$ be an optimization problem instance. Then if

$$\begin{aligned} &1. \quad \textit{there exists a bijection } \beta : V \leftrightarrow X \\ &2. \quad \langle \forall x \in X : \forall x' \in D_x : \exists(\beta(x), \beta(x')) \in E \textit{ and not otherwise} \rangle \end{aligned} \tag{5.11}$$

*we call G a **dependency graph** for R.*

Definition 5.9 (Neighbors)

Let $x \in X$ and let $v = \beta(x)$. Then $N_v \triangleq \bigcup_{x' \in D_x} \{\beta(x')\}$ is the set of ***neighbors*** *of v in G.*

Note that $|N_v| = |D_x|$ because β is a bijection.

Definition 5.10 (Vertex Elimination Function)

Let $G = (V, E)$ be the dependency graph for a relational structure $R = (X, F, \phi)$. Let $x \in X$ and $v = \beta(x)$. Let $A = \bigcup_{v', v'' \in N_v} (v', v'')$ be the set of edges forming a clique among all neighbors of v and let $B = \bigcup_{v' \in N_v} (v, v')$ be the set of all edges incident on v in G. We define a ***vertex elimination function*** *η on G as*

$$\begin{aligned} G_v &\triangleq \eta_v(G) \quad \textit{where} \\ G_v &= (V - \{v\},\ E \cup A - B) \end{aligned} \tag{5.12}$$

and it is a graph dual of the relaxation function ρ_x on ϕ.

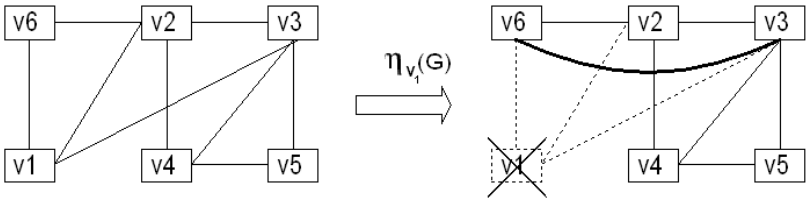


Figure 5.1: Vertex elimination of V_1

In Figure 5.1 the vertex V_1 and all its edges are removed and a new edge (V_3, V_6) is added to $E(G)$.

Similarly to the relaxation process we can define a vertex elimination process on G as follows:

Definition 5.11 (Ordered Graph)

Let $G = (V, E)$ be the dependency graph for a relational structure $R = (X, F, \phi)$. Then an ***ordering*** *of V is a bijection $\pi : \{1, 2, ..., n\} \leftrightarrow V$ and we call G_π an* ***ordered graph****. In this way, $v_i \in V$ in G_π is the i^{th} vertex with respect to ordering π.*

5.4 Vertex Elimination Process

Definition 5.12 (Elimination Process)

For an ordered graph G_π we define a vertex elimination process

$$P_G(G_\pi) = [G = G_0, G_1, ..., G_n] \tag{5.13}$$

as a sequence of derived graphs defined recursively by $G_0 = G$, $G_i = (G_{i-1})_{v_i}$ for $1 \leq i \leq n$.

Now, we need to make sure that as we apply η to G it corresponds exactly to the dependencies among variables after applying a relaxation ρ to ϕ. And, inductively, that the graph is always in correspondence with the transformations taking place on ϕ.

Lemma 5.4

Let G be the dependency graph for a problem instance $R = (X, F, \phi)$. Let $x \in X$ and $v = \beta(x)$. Then G_v is the dependency graph for R_x.

Proof: Recall that $G_v = \eta_v(G) = (V - \{v\},\ E \cup A - B)$ from definition 5.12 and $R_x = (X - \{x\},\ F \cup \rho_x,\ \phi(x = \rho_x))$ from definition 5.4. First, in R_x the variable x is removed from X and the corresponding v is removed from V. When v is removed, all the edges in G incident on v are removed via set B, which follows that no variable in X is dependent on x after x is relaxed. Second, in the relaxation of x a new function is created, ρ_x, which has all variables in D_x as parameters (definition 5.1). Now, all the variables present as parameters to a function in F as mutually dependent, based on the dependency separation rules in section 4.1.1. Thus, there must be a an edge in G_v for each pair (v, v') where $v, v' \in N_v$, which is insured with set A in the definition. ■

Corollary 5.5

Let π be an ordering for G_π and R_π such that $v_i = \beta(x_i)$. Then G_{v_i} is the dependency graph for R_{x_i} for all $1 \leq i \leq n$.

Proof: immediate from the definitions 5.5, 5.13 of P and P_G and lemma 5.4 by induction on i. ■

Corollary 5.6

If G_v is the dependency graph for R_x then $|D_x| = |N_v|$

Proof: Immediate from definition 5.11 and lemma 5.4. ■

5.5 Relaxation + Elimination Complexity Relationship

At this point, with the dependency graph and the relational structure in agreement, we can define a cost relationship between ρ and η. Now, here is a key insight

Theorem 5.4

Let π be an ordering for G_π and R_π such that $v_i = \beta(x_i)$. Then the computational complexity of a relaxation process R_π is $O(\sum_{1 \leq i < n} m^{|N_{v_i}|+1})$, when it follows the vertex elimination process G_π. Note, that N_{v_i} is the set of neighbors of v_i in G_i.

Proof: As stated in section 4.1.1 the computation cost to perform ρ_x for some $x \in X$ is $m^{|D_x|+1}$. From corollary 5.6 we have $m^{|D_{x_i}|+1} = m^{|N_{v_i}|+1}$, where $v_i = \beta(x_i)$. ■

$$\begin{array}{ccccccc} G_0 = (V, E) & \xrightarrow{\eta_{v_1}(G_0)} & G_1 = (V - \{v_1\}, E \cup A - B) & \xrightarrow{\eta_{v_2}(G_1)} & \cdots & \xrightarrow{\eta_{v_{n-1}}(G_{n-1})} & G_n \\ \updownarrow & & \updownarrow & & & & \updownarrow \\ R_0 = (X, F, \phi) & \xrightarrow{\phi(\rho_{x_1})} & R_1 = (X - \{x_1\}, F \cup \rho_x,\ \phi(x = \rho_{x_1})) & \xrightarrow{\phi(\rho_{x_2})} & \cdots & \xrightarrow{\phi(\rho_{x_{n-1}})} & R_n \end{array} \tag{5.14}$$

The value of the above theorem is that it allows us to express the computational complexity of the solution to our optimization problem purely in terms of neighbors in the corresponding dependency graph. The equation 5.14 shows the relationship between the optimization problem and its dependency graph lockstep transformation. Now, if we find a way to order the vertices to produce a small degree polynomial for the sum of complexities, $O(\sum_{1 \leq i < n} m^{|N_{v_i}|+1})$, in terms of neighbors, performing the relaxation process in same order would give us a low degree polynomial solution for the optimization problem.

To find good vertex orderings we propose two heuristics, one based on graph decomposition techniques and the other based on local properties of the dependency graph.

Chapter 6

Relaxation Order

In the previous chapter we showed a method for finding an exact optimal solution to any instance of ARA using the relaxation operation. However, the computational performance of the method depends on the order in which the relaxations are carried out. In this chapter we propose two methods for finding relaxation orders which seem to produce good results in practice.

The relationship between the relaxation operation and vertex elimination in graphs established in theorem 5.4 allows us to look for a relaxation order in the structure of the dependency graphs. We propose two approaches: a recursive graph decomposition via separators and a greedy search based on vertex degrees.

6.1 Graph Decomposition

Recent advances in graph complexity theory provide a set of tools to analyze graphs through tree decomposition. We first show the properties of tree decompositions and then refer to two of several known algorithms, one based on clique separators and originally due to Tarjan[35] with improvements by Leimer [22], and another based on minimal separators which can be identified with techniques by [4] and [23]. We then show how to use the decompositions produced with these techniques to extract good relaxation orders.

6.1.1 Notation

Let $G = (V, E)$ be an undirected, simple graph. Let $U \subseteq V$, then $G(U) \triangleq (U, E(U))$ is the *subgraph* of G *induced* by U, where $E(U) \triangleq \{\{v, w\} \in E : v, w \in U\}$. A path between v and v' in G is denoted by vGv' and if v'' is a vertex on the this path, then $v'' \in vGv'$. $\mathcal{C}(U)$ denotes the set of connected components of $G(V \setminus U)$.

$G(U)$ is a *clique*, if $\forall v, w \in U : \exists\{v, w\} \in E$. Also, the empty set $\varnothing$ is a clique. A graph is a clique if its vertex set is a clique. $U \subseteq V$ is a *separator* for $A, B \subseteq V - U$, if every path between $a \in A$ and $b \in B$ contains a vertex in U. U is a separator for G if there are non-empty sets $A, B \in V - U$

such that U is a separator for A and B. A graph G that has a clique separator is called *reducible*, otherwise it is *prime*.

6.1.2 Tree-decomposition properties

Definition 6.1 (Decomposition Tree)

*A **decomposition tree** of a graph G is a pair $\mathcal{D} = (T, \mathcal{V})$, where T is a tree and $\mathcal{V} = \{V_t\}_{t \in T}$ is a family of vertex sets $V_t \subseteq V(G)$, indexed by the vertices $t \in T$ that satisfy the following:*

$$\begin{aligned} &(T1) \quad V(G) = \textstyle\bigcup_{t \in T} V_t \\ &(T2) \quad \forall (v, w) \in E(G) : \exists\, t \in T : v \in V_t \wedge w \in V_t \\ &(T3) \quad \forall t_1, t_2, t_3 \in T : t_2 \in t_1 T\, t_3 \Longleftrightarrow V_{t_1} \cap V_{t_3} \subseteq V_{t_2} \end{aligned} \tag{6.1}$$

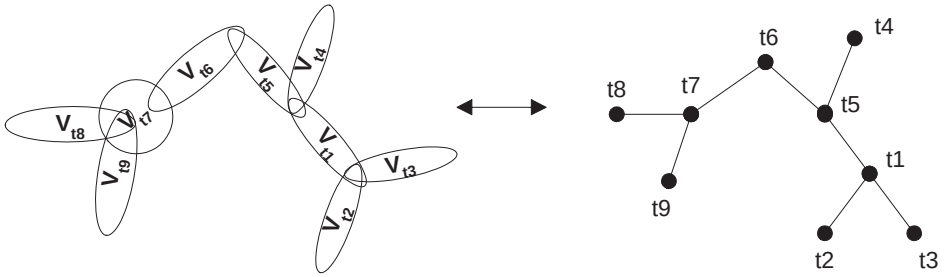


Figure 6.1: Tree Decomposition of a Graph

Definition 6.2 (Tree-Width)

*A **width** of a tree decomposition $(T, \mathcal{V})$ is*

$$\max\{|\mathcal{V}(t)| - 1 : t \in T\} \tag{6.2}$$

*and the **tree-width** of G, denoted by $tw(G)$, is the least integer w such that G admits a tree-decomposition of width w. Graphs of tree-width $\leq k$ are called **k-decomposable**.*

Remark 6.1

For example, $tw(G) \leq 1$ if and only if G is a forest; $tw(G) \leq 2$ if and only if it is series parallel; and the complete graph K_n has $tw(G) = n - 1$.

Some of what is known about tree decompositions is summarized in [23] as follows: 1) many NP-hard graph problems can be solved in polynomial time when a tree decomposition is given 2) if tw is part of input, the decision problem "is $tw(G) \leq w$?" is NP-complete [1]. 3) For every fixed w there exists an $O(|V(G)|^2)$ algorithm for deciding whether $tw(G) \leq w$, though the proof is merely existential. 4) There are linear algorithms to determine whether $tw(G) \leq w$, where $w = 1, 2, 3$

Class	Treewidth
Bounded degree	N[8]
Trees/Forests	C
Series-parallel graphs	C
Outerplanar graphs	C
Halin graphs	C[36]
k-Outerplanar graphs	C[7]
Planar graphs	O
Chordal graphs	P(1)
Starlike chordal graphs	P(1)
k-Starlike chordal graphs	P(1)
Co-chordal graphs	P[8]
Split graphs	P(1)
Bipartite graphs	N
Permutation graphs	P[5]
Circular permutation graphs	P[5]
Cocomparability graphs	N[2],[17]
Cographs	P[6]
Chordal bipartite graphs	P[20]
Interval graphs	P(2)
Circular arc graphs	P[34]
Circle graphs	P[19]

P = polynomial time solvable. C = constant (linear time solvable). N = NP-complete. O = Open Problem. (1) The treewidth of a chordal graph equals its maximum clique size minus one. (2) The treewidth of an interval graph equals its its maximum clique size minus one.

Table 6.1: Classes of graphs with known treewidth computation time, from Bodlaender[8]

and when the result is positive to construct the tree. Finally, there is a variety of approximate decomposition algorithms as well as some that find less than optimal decompositions, such as one returning at most $4 * tw$ [28]. Bodlaender in [8] provides a summary of classes of graphs for which there are known treewidth results (see Table 6.1).

Definition 6.3 (Minimal Separators)

*$U \subseteq V$ is called an **ab-separator** if there exist vertices $a \in V$ and $b \in V$ that are in different connected components of $\mathcal{C}(U)$. The set U is a **minimal ab-separator** if no proper subset of U is an ab-separator. Finally, U is a **minimal separator** if there is some pair $a, b \in V$ for which U is a minimal ab-separator. Separators that are cliques are called **clique separators**. The nodes of the decomposition tree derived via clique separators are called **simplical summands**.*

6.1.3 Decomposition Process

A decomposition of a simple connected graph $G = (V, E)$ works by finding a minimal separator U that breaks G into two or more connected components $\mathcal{C}(U)$. Then each component is augmented with the subgraph induced by the separator. We repeat the decomposition process on each component recursively until no longer possible. Namely, until all the resulting components are prime.

In general the decompositions are not unique, since often a graph can be split along several different separators. This yields different decomposition trees of different widths. If the separators are chosen to be minimal clique separators, then any decomposition gives the same prime components, though not necessarily of optimal width. Developing methods for choosing the separators to yield the lowest width decompositions is currently of primary interest for our algorithm to work efficiently, as well as for many other applications.

Tarjan[35] proposed a polynomial-time algorithm that decomposes a graph by systematically choosing minimal clique separators. Berry[4] showed a way to identify all the minimal separators of a graph at the cost of $O(n^3)$ per separator, any of which could be chosen for decomposition. We simulated a number of graphs choosing minimal separators at random and this approach has shown to give low width decompositions for many practical problem instances.

Example 6.1 (Graph Decomposition)

Figure 6.2 is an application of a clique decomposition of our example problem 4.1 with minimal separators.

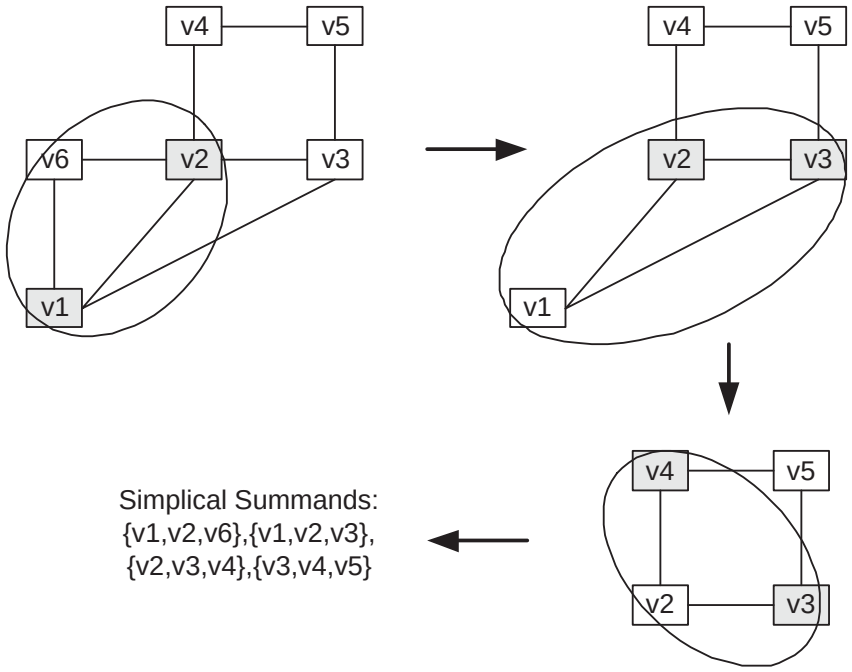


Figure 6.2: Example Problem Dependency Graph

In Figure 6.2, the first clique is $\{v_1, v_2\}$ which separates the graph into a prime subgraph $\{v_1, v_2, v_6\}$ and the remaining part $\{v_1, v_2, v_3, v_4, v_5\}$. The next clique separator is $\{v_2, v_3\}$ and so on. In this way, the decomposition produces the irreducible subgraphs indicated in the figure.

6.2 Relaxation Order

Now, the final step is to extract a good vertex elimination ordering from the prime subgraphs produced by a graph decomposition. We propose the following 2-step algorithm:

Algorithm 6.1 Input: Let $(T, \mathcal{V})$ be a tree decomposition of G. Then,
1) Find a vertex v that appears in exactly one prime subgraph V_t and label it as next (starting at 1).
2) Remove v from V_t; $V_t := V_t - \{v\}$ and if $V_t - \{v\}$ is the separator of V_t in the tree decomposition then remove the leaf V_t from $\mathcal{V}$.
3) If $\mathcal{V}$ is not empty, repeat step 1.
Output: a vertex ordering π.

Lemma 6.2

In step 1 of algorithm 6.1 a $v \in V$ that appears in exactly one prime subgraph always exists.

Proof: Suppose not. Then we have a tree where every vertex of G appears in at least two leaves, namely $\forall v \in V : \exists V_t, V_t' \in \mathcal{V} : v \in V_t \wedge v \in V_t'$. This is only possible when $v \in C$ and C is a separator for V_t and V_t'. Thus, all the vertices must be in some separators. Impossible, since initially a) by definition a separator C divides a graph into subgraphs A and B, both non-empty, such that $A \cap B = \varnothing$ and b) step 2 removes the V_t when all the separated vertices have been removed. ■

Lemma 6.3

The algorithm 6.1 enumerates all the vertices in G.

Proof: Clearly all the non-separator vertices are enumerated, since they satisfy the condition in step 1 directly. So, we only need to show this for all the vertices in the separators. Now let $v \in V$ be such that there exists a separator C where $v \in C$. Then there exists some set of prime subgraphs $V_t^1, ..., V_t^k$ where $v \in V_t^i$ for $1 \leq i \leq k$. From step 2 we know that a prime subgraph is removed from $\mathcal{V}$ when all its vertices but those in its separator have been removed. These subgraphs are going to be removed in some order, and since vertices are never added to any subgraphs, there will eventually be a the last one, say V_t^j, that is the only one containing v. But this would satisfy the condition in step 1 and v will then be enumerated. ■

6.3 Main Result

The following theorem completes the relationship between the optimization problems and their dependency graphs. It ties the complexity in terms of neighbors in Theorem 5.4 with a standard graph complexity metric of tree-width.

Theorem 6.1

Let $\mathcal{D}$ be a tree-decomposition of the dependency graph G for problem R of width w, producing an ordering π. Then, the computational complexity to find an optimal solution to R via the relaxation process in the order π is $O(\phi_c * n * m^{w+1})$.

Proof: From theorem 5.4 we have that for any ordering π, performing the relaxation process R_π has a computational complexity of $O(\sum_{1 \leq i < n} m^{|N_{v_i}|+1})$, where N_{v_i} is the set of neighbors of v_i in G_i of the corresponding vertex elimination process G_π. To show the bound in terms of tree width w, we only need to show that each vertex v_i being eliminated has $|N_{v_i}| \leq w$ in G_i.
Now, a tree decomposition $\mathcal{D}$ of G of width w has the property that for each leaf $t \in T$, $|\mathcal{V}(t)| \leq w$. Its other property (from definition T2 in 6.1) is that $\forall v \in \mathcal{V}(t) - C(t) : \forall v' \in N_v : v' \in \mathcal{V}(t)$, where $C(t)$ is the separator for t. Namely, all the neighbors of any v in $\mathcal{V}(t) - C(t)$ belong to the same prime subgraph $\mathcal{V}(t)$. Therefore, the cost to eliminate any vertex v in $\mathcal{V}(t) - C(t)$ is $O(m^{w+1})$.

Now, the ordering π produced by algorithm 6.1 is such that when any vertex v is removed, it is not a separator, i.e. it appears in only one leaf. Thus, in the worst case, the total cost to eliminate all the vertices in the order π is $O(\sum_{1 \leq i < n} m^{w+1}) = O(n * m^{w+1})$ and, the relaxation process R_π has the same computational complexity. QED. ■

6.4 Greedy Heuristic Search

A local ordering method is an algorithm that finds a relaxation order for a dependency graph, enumerating the vertices by computing some function of their degrees and those of their neighbors. An optimal ordering is one that produces the smallest total computational cost for the optimization problem when carrying out the relaxation process in that order.

We propose two 'immediate' greedy heuristics. Both algorithms take a graph generated previously via a dependency separation process and output a vertex ordering to guide the relaxation.

The algorithms follow this structure:

Algorithm 6.2 Greedy Elimination Ordering

1. Let $i = 0$.
2. Choose the next vertex k via a heuristic
3. Assign $i + 1$ to k.
4. Perform the vertex elimination on k – remove it from the graph and connect all of its neighbors
5. If there are any more vertices, then go to step 1; otherwise, stop.

Algorithm 6.3 Smallest Degree First

2. Find the vertex k with the least degree (if more than one, choose at random)

Algorithm 6.4 Least Edges Created

2. Find the vertex k such that the number of edges created after the relaxation of k is least. This value for any vertex j is $\frac{d*(d-1)}{2} - m$, where $d = Deg[j]$ and m is the number of edges that exist between the neighbors of j.

The greedy method makes a locally optimal decision at each step, where the "best" vertex is chosen depending on the heuristic. The basic problem with this approach is the following: when a vertex is removed its former neighbors are fully connected and as a result can make the graph more complex than it needs to become. In the worst case, a what looks like a good local relaxation can unnecessarily create a new largest clique in the graph, making the total complexity proportional to its size. Since finding the largest clique in a graph is NP-complete[14], testing for an increase is hard.

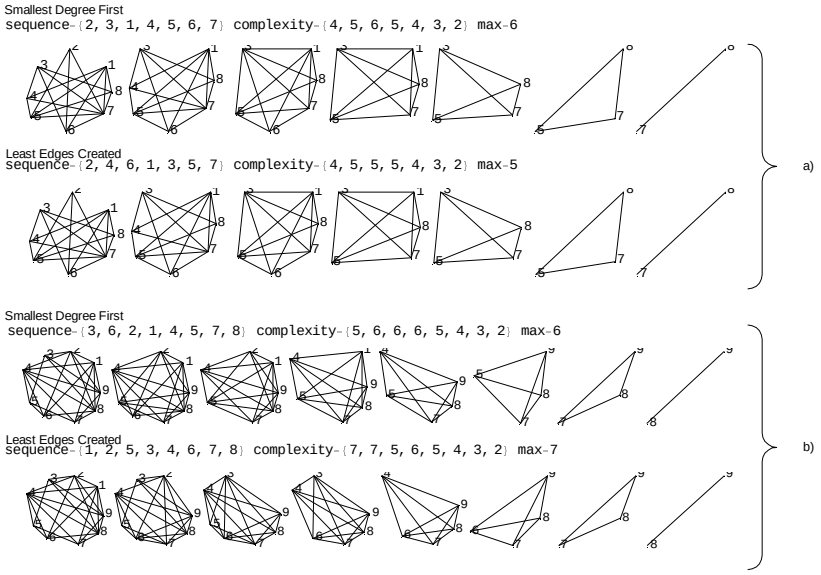


Figure 6.3: Non-optimality of the heuristics.

Neither heuristic is optimal and Figure 6.3 demonstrates two counterexamples. The sequence set indicates the order in which the vertices are removed and the complexity set the corresponding cost to remove the variables in that order. The part a) of the figure shows a graph where the Least Edges heuristic is better. The b) graph shows that the Smallest Degree First is better. The sequences of graphs show how the dependency graphs evolve in response to the relaxation processes.

6.5 Open Issues

There are two main directions open for further investigation. First, it is desirable to find efficient dependency separation rules for more objective function types. Second, we would like to find a way

to better choose minimal separators for the most common problem instances. Third, we conjecture that any selection heuristic for the greedy search, computed from vertex degrees alone, will likely yield about the same quality orderings as the two we showed above.

Chapter 7

Simulations

To see how well our relaxation method works we simulated a number of problem instances which we believe are likely to appear in practice. We looked at a range of problems, including problems with as few variables as 8 and as many as 100 with a varying number of relationship functions. For each generated instance we followed the solution process in Figure 4.1, namely:

1. Carried out the dependency separation following the algorithms in Section 5.2 to produce a dependency graph
2. Applied the graph decomposition from Section 6.1 and the greedy search heuristics to the graph as described in Section 6.4
3. Generated the relaxation orderings as in Section 6.2
4. Extracted the computation costs for the optimal solution search via both methods.

The following three experiments present our findings.

7.1 Experiment Setup

We implemented a relaxation simulator in Mathematica 4.1 with the extended combinatorica.m package from Skiena [31]. It consists of a random problem generator, our relaxation algorithm and two vertex elimination heuristics: MinDegree and LeastEdgesCreated. For decomposition we used a MSVS simulator developed by Koster [21].

A random problem is generated from three parameters: number of variables, number of relations and the number of arguments for each relation. For each relationship we randomly select a set of parameters, avoiding duplicates and duplicate relationships. Though a real-world problem would likely have relationships with different arity, we believe we can generate a diverse set of problems of varying complexity by changing the number of relations and keeping the number of arguments fixed.

7.2 Experiment 1 (Small Problems with Known Optimal Cost)

For small problems (8 variables or less) it is possible to find the optimal (fastest) relaxation ordering via brute-force in reasonable time, trying all possible variable assignments and computing the objective function. We generated 420 problems of 8 variables, between 4 and 10 relationships of 3 arguments each. The number of random relationships, on average, roughly controls the density of the dependency graph and the complexity of the problem. For each problem we computed 1) the size of the maximum clique of the dependency graph, an optimal relaxation sequence by exhaustive-search, and the cost of the relaxation order produced by the Min-Degree heuristic. The results of the simulation, shown in Figures 7.1 and 7.2, give us a measure of how well the heuristic and max clique values predict the optimal.

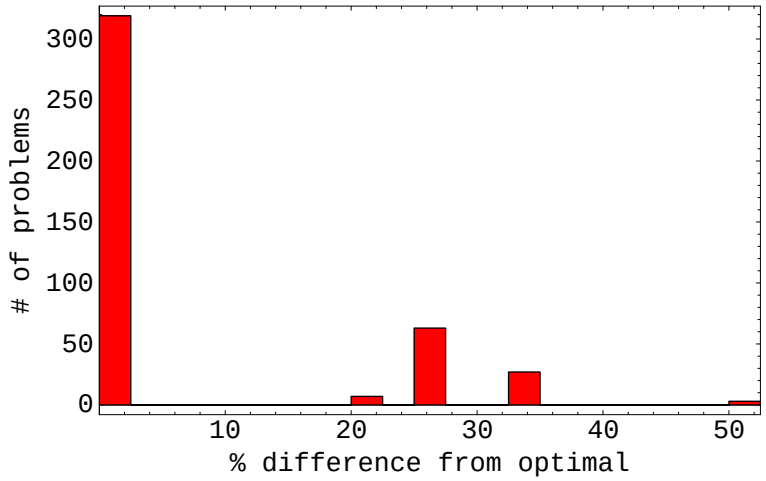


Figure 7.1: MaxClique and Optimal comparison for 8 variable problems.

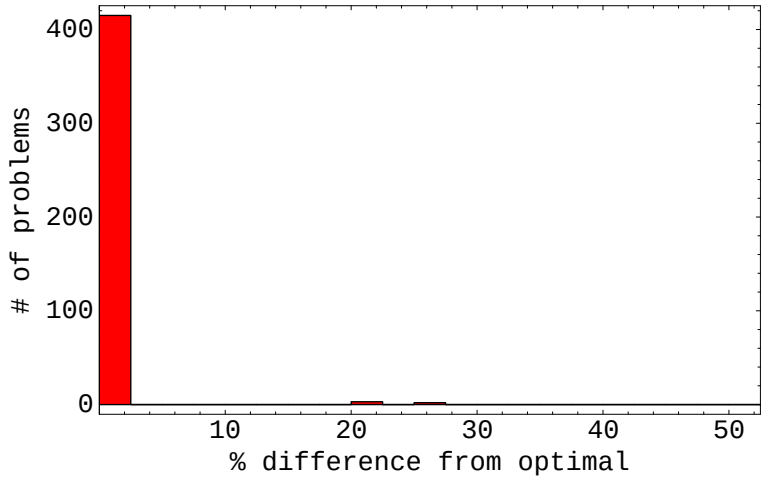


Figure 7.2: MinDegree and Optimal comparison for 8 variable problems.

The "% difference from optimal" was computed as $\frac{(Opt-MaxClique)*100}{Opt}$ for the maximum clique

in Figure 7.1 and similarly for the min-degree case. The variance for the $MaxClique$ difference is 155, standard deviation of 12.4 and the median of 0. The variance for the $MinDegree$ difference is 6, standard deviation 2.4 and the median of 0.

Figure 7.3 shows an example of a problem with MaxClique 3, an Optimal cost of 5 and MinDegree of 5.

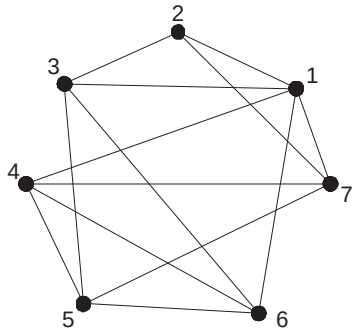


Figure 7.3: Example of problem with MaxClique=3 and optimal cost=5

The MinDegree heuristic gives a relaxation sequence of $\{2, 1, 3, 4, 5, 6\}$ with the corresponding cost sequence $\{4, 5, 5, 4, 3, 2\}$. Namely, it costs $O(m^4)$ to relax the first vertex (#2). The execution of the relaxation sequence is given in Figure 7.4.

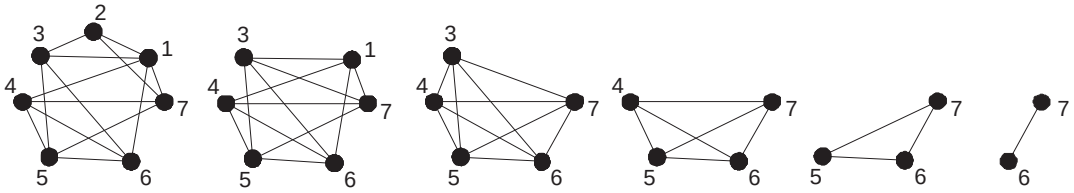


Figure 7.4: Relaxation of a problem with MaxClique=3 and Optimal cost=5

Figure 7.5 shows a problem with $MaxClique = 4$, $Optimal = 5$ and $MinDegree = 6$, with the Min Degree relaxation sequence $\{1, 2, 3, 4, 5, 6, 7\}$, cost sequence $\{5, 5, 6, 5, 4, 3, 2\}$ and the execution of the relaxation in Figure 7.6. The graph decomposition algorithm separates this graph first with separator $(4, 7, 5, 8, 1)$, isolating vertex 2 on one side of the separator and vertices 3 and 6 on the other. Then with separator $(4, 6, 1, 5, 7)$, with vertex 3 on one side and vertex 8 on the other. This gives a relaxation order of $(2, 3, 8, 1, 4, 5, 6, 7)$ and gives the relaxation cost of 6.

The data shows that the MinDegree heuristic is a much better predictor of the optimal than MaxClique for this set of problems. However, more importantly, we do not think there is a phase transition as the size of the graph increases, therefore the same should hold for larger graphs as well.

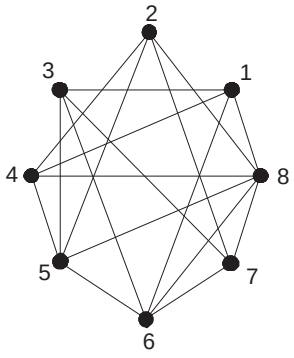


Figure 7.5: Example of problem where all three measurements differ

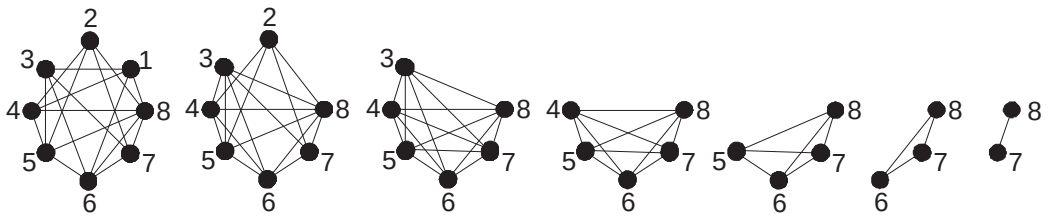


Figure 7.6: Relaxation of the problem with $MaxClique = 4$, $Optimal = 5$ and $MinDegree = 6$

7.3 Experiment 2 (Large Problems of Variable Complexity)

A lower bound on the computation cost of a problem instance, using the relaxation method, is the size of the largest clique in the dependency graph. For graphs of less than 100 nodes and moderate density it is possible to search for the size of the maximum clique within a few minutes,using the modern processors. This enables us to determine the "ball-park" of the performance of the greedy and graph decomposition heuristics as relaxation ordering selection strategies.

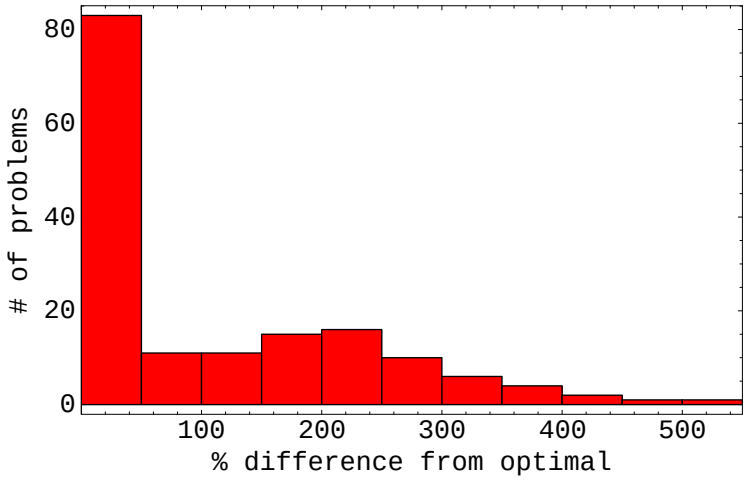
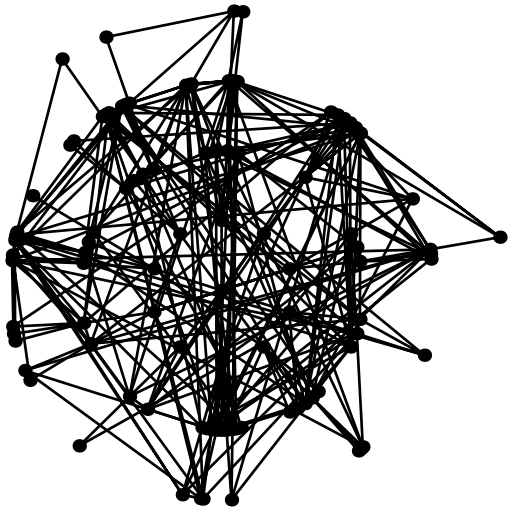


Figure 7.7: Histogram of the MinDegree % difference from MaxClique for large graphs

Problem size	**MaxClique**	**MinDegree**	**MinEdgesCreated**	**Decomposition**
100 nodes 20 relations	3	3	3	3
100 nodes 40 relations	3	4	3	5
100 nodes 60 relations	4	9	8	12
100 nodes 80 relations	4	13	12	19
100 nodes 100 relations	4	22	20	25

Table 7.1: Sample simulation results

The histogram in Figure 7.7 shows the result of 160 problems of 100 variables and 20 to 100 relationships. We generated an equal number of problems for each 10 relationship increment. The high spike showing 0% difference is primarily due to the "sparse" graphs, while the "Gaussian-like" tail is due to the more complex ones. Figure 7.8 shows the dependency graph for one of the more complex instances.



Degree sequence: {1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 11, 11, 12, 12, 12, 12, 16}.

Figure 7.8: Dependency graph for 100 variable problem with 100 relationships of 3 arguments each

Table 7.1 shows the simulation results for a sample set of problems. It shows that the two greedy heuristics and a minimal graph decomposition give approximately the same results, with the decomposition doing slightly worse.

7.4 Experiment 3 (Comparison of Greedy Heuristics)

In Section 6.4 we gave two greedy heuristics: Smallest Degree First (MinDegree) and Least Edges Created (MinEdges). In this experiment we compared the results of two heuristics on 200 of the

100-node problems discussed in the previous section. Figure 7.9 shows a histogram of the percent difference between the heuristics.

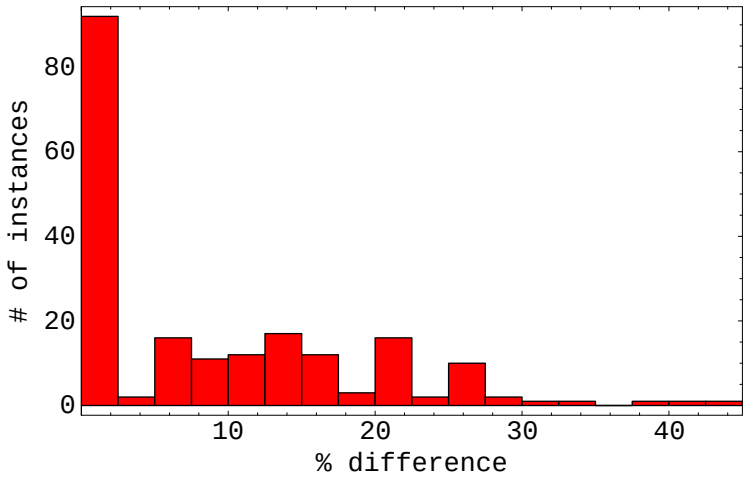


Figure 7.9: Histogram of % difference between MinEdgesCreated and MinDegree

The heuristics give practically identical results (relaxation orders of very similar quality), with a slight advantage to MinDegree (it is the better heuristic in almost all of the instances).

7.5 Observations

The heuristics show that the relaxation algorithm can be effective as a method for a number of randomly generated problem instances. We expect that in practice the results would be even better, because the business processes designed by humans will likely have more order. We also expect the problems to fall into one of the classes with low treewidth, as given in Table 6.1. Finally, other minimal decomposition algorithms may give better results. A comparison of them for this problem is needed.

Chapter 8

Atomic Resource Reservation using Call Options

For applications where resources are physically distributed, a consumer attempting to commit more than one of them into a simultaneous agreement, to participate in a business process, runs into the fundamental issue in network computing: the distributed agreement problem. The problem arises because resources are physically separated and though the consumers can communicate freely with each resource, there is a time delay in any communication which gives the resource an opportunity to change state while messages are in transit. This means that a consumer always has only the "old" information about the resource state (availability in particular). Yet, we need to establish a global predicate on the system – a collective agreement to execute a business process, and therefore need a mechanism to solve the problem reliably regardless of how and when the states may change.

Uncertainty about the states of the resources in a distributed system forces us to create protocols for communication and operation for all the participants to follow. Otherwise, if the protocols are violated or nonexistent, the system is said to be faulty and it can be shown that no agreement is then possible [16]. The traditional methods create protocols that involve special messages, called tokens, that the participants (in our case the consumers and resources) can exchange to help with synchronization. For example, in a common technique called two-phase-commit used to synchronize transactions among distributed databases, a 'coordinator' database first sends a 'preparation' token to the other databases asking them to 'prepare' to commit. It then waits to receive acknowledgement tokens from everyone, signifying that they are ready. In the second phase it sends the 'commit' tokens to signal the event, and then waits again for acknowledgments that everyone has indeed committed. If any of the participants have failed for whatever reason, all others are asked to roll-back (cancel) the transaction.

While such techniques work well from the correctness point of view, namely the databases go from one consistent state to another after a two-phase-commit, they cannot be applied directly to coordinating resources that operate both in a logical system and in an economy. The economic

aspect adds the extra requirement that is not accounted for in the standard algorithms. Therefore, we need to solve the distributed agreement problem in a new way such that the economics make sense as well. Specifically, since the resources are leased for time, any period that they stay unassigned has an 'unearned' value to each resource proprietor, which is what the economists call *opportunity cost*.

Traditional distributed algorithms are not concerned with time, focusing exclusively on the logical consistency. The specification of a distributed agreement algorithm typically states that an agreement will be reached *eventually*, if certain properties are true about the system, such as the system is not faulty and the communication channels deliver all messages. In our system, however, sending a token to a resource and asking it to wait until some other message arrives is reasonable only if the waiting time is somehow paid for. If the 'money constraint' is not satisfied, no resources will be offered for lease and the model breaks.

The opportunity cost must be absorbed by either the resource, as 'part of doing business,' or by the consumer who is attempting to reserve it for a business process. In the existing U.S. economy, different sectors of the industry have resolved this in either of the two directions. In the hotel industry one can place and cancel reservations for free if the cancellations take place a day or more before the service is rendered. Otherwise, the hotels charge a cancellation fee. In this way, the hotels absorb the opportunity cost of holding the room for the customer who requested it until a certain threshold, when this cost becomes too high. The airline industry sells several types of tickets. The fully-refundable ones have the opportunity cost built into the price. The non-refundable ones leave all the risk on the consumer.

Opportunity costs and risk management are subjects extensively studied in micro economics and finance. However, what makes the Distributed Service Commit problem new is the goal of achieving a commitment among *several* resources from a finite pool where time has value.

The solution we present here consists of algorithms for the consumer, resource and directory entities. In this thesis our focus is almost entirely on the consumer and the algorithms for the resource and the directory are little more than skeleton behaviors that a consumer needs to be able to interact with them. However, algorithms for the resources and the directories could be quite sophisticated. For example, designing distributed databases for the directories to efficiently store data about resources is a research field in its own right. One could also invent resource algorithms to perform elaborate computations to decide what prices to set on their resources. However, here we focus only on the *interaction* between consumers and resources towards achieving an atomic agreement. The algorithms are presented next.

8.1 Entities (operational overview)

8.1.1 Consumer

A consumer interacts with resources and a directory and its behavior consists of the following basic steps:

1. Ask the directory service for information about the resources that satisfy some given constraints
2. Request the availability information from each prospective resource
3. Run the optimization algorithm with the collected resource instance information.
4. Make a reservation of the optimal resources as an atomic transaction

In the first step the consumer seeks to locate all the resources that it could potentially use for its business process. To do this, it sends specifications of the resources it requires to a Directory. The directory returns a set of resource locators (URLs) that satisfy the consumer's constraints. For example, all the plumbers with at least 5 years experience within 20 miles of the construction site. If this set is empty (that is, the directory did not find any resources matching the query) then we assume that the client terminates (fails).

In step 2, the consumer directly contacts the resources whose locators it acquired in step 1, asking for their availability information. The resources respond by sending back to the consumer lists of their available time slots and pricing information. At this stage, the consumer has all the information it needs about each resource to be able to determine which ones it could use best. In step 3, the consumer algorithm uses this information to find an optimal solution. Finally, in step 4, the consumer entity negotiates with the resources chosen in step 3 to perform reservations according business process specification.

8.1.2 Directory

A directory entity has a message loop and responds to messages from consumers. The single type of message expected by the directory is a query for resources based on a set of constraints. This is similar to a database lookup, where the consumer specifies the constraints in arbitrary detail and expects the directory to identify all the resources matching them. When a directory receives such a request, it queries its own internal database and possibly the databases of other directories and returns a (possibly empty) set of resources as a result.

8.1.3 Resource

A resource in our model registers with a directory so that consumers could find and consider it for their business processes. It then proceeds to wait for a message from a consumer. There are two

categories of messages: information request and reservation/release messages. A resource receives the first type when a consumer in its step 2 asks for its availability. It replies to that by sending its availability time list. The second message type is invoked when a resource enters a negotiation for reservation. One type of reservation message requests a right for a consumer to reserve this resource some time in the future and to hold this right for some small amount of time (the Micro-Option request). The second is the actual reservation request (and an optional matching reservation release) if the consumer chooses to exercise their reservation option.

8.2 Distributed Transaction

We now present an algorithm for commencing an *atomic distributed reservation transaction* between a resource and a consumer. We assume that the consumer is at its final stage, where it has chosen a set of resources for reservation. By a set reservation we mean an atomic operation, such that either all the resources in the set become reserved or none of them. For instance, if a consumer has chosen the resources for a trip process and the return flight were not available when the consumer becomes ready to make a reservation, then he would not be interested in a partial reservation of just the unidirectional flight, a hotel and car. Only the reservation of the whole process is of interest.

The reason why such a reservation commitment is tricky is because the resource and the consumer entities are in a distributed system and the information they have about each other is always aged. In particular, the availability of a resource may have changed between the time a consumer has requested that information (in step 2) and the time of attempting the reservation. Thus, when the consumer attempts to make the reservation, it would fail for that resource. Because a process consists of a number of resources that all have to be reserved (or not reserved) together, it is not in a consumer's interest to start reserving a process and failing some time in the middle. To remedy this, one needs a mechanism that would give the consumer an insurance that its attempt at reserving a process would succeed with high probability (it is not possible to offer an absolute guarantee, since any entity in the system could, for example, loose its network connection, rendering it unreachable). We propose a solution for a distributed atomic transaction based on a financial instrument called **American Call Option** [24]. We define an option in our model as *a right, that can be purchased by a consumer, to reserve a resource for some specific time in the future.* An option has a fixed expiration date T (the date when the right expires), a strike price K (the cost of the actual reservation) and a premium U (the amount a consumer must pay to purchase the right to reserve). An option, when purchased, is binding to the resource for which it was purchased, but it is not binding to the consumer. By buying an option for a resource, the consumer has a right, not an obligation to reserve that resource.

Using this reservation tool, it now becomes possible for a consumer to purchase short-lived

tentative reservation commitments from resources by paying them to hold the time blocks that it is interested in. Now, the algorithm for the atomic reservation transaction can be outlined as follows.

Suppose a consumer C wants to make a reservation of resources for some process P. Then it performs the following steps:

1. send a message to each resource in P asking to purchase an option for a specific time block
2. wait for a confirmation from all the resources or stop the reservation if a negative response is received or the wait times out with some of the resource confirmations missing.
3. if the reservation stopped in step 2 then fail, otherwise send reservation messages to each resource in P.
4. wait for a confirmation from all the resources or roll-back (cancel all) the reservation if no response is received from any of the resources in the allotted time.

Note that no explicit releasing of resources is necessary, since all reservations are done for a fixed amount of time. Option prices are going to be determined by the market and, in a free-market, likely to be close the opportunity cost. If the option costs are significant, the algorithm can be easily modified to consider only the sets of resources for which the consumer can afford to pay the reservation cost, namely, buying the options.

This algorithm ensures an atomic transaction between a consumer and a resource if 1) the conditions for commencing the actual resource reservations in step 3 have been met, 2) the option expiration date for the resources was sufficiently far into to the future for the reservation requests to reach the resources, and finally 3) the infrastructure was able to deliver the messages successfully.

In the sections that follow we give a pseudo-code programs for the entities. The code is a high level implementation of the functionality for each entity of the distributed system

8.3 Program Notation

As stated in Chapter 2, we assume the presence of an infrastructure capable of delivering messages from one entity to the other. Our entities are entirely message-driven: each entity has an event loop constantly waiting for messages. When one arrives, an entity executes its program code to handle the message and immediately returns back to waiting. The messages received while an entity is handling an earlier message are queued in FIFO order and become available to the entity when it returns to the message loop.

We use **receive MessageName(M) time-out in X** expression to denote the readiness of an entity to receive a message of type *MessageName*. Each message can be though of as a structure capable of carrying arbitrary information. When a message is processed (or accepted), its payload

can be found in variable M. The message variable M also contains a special attribute, *sender*, referred to as *M.sender*, that identifies the source of the message. We also assume that a message can carry arbitrarily complex data – a list, a matrix, etc. If a **receive** statement has the **time-out** extension it signifies that the entity executing it should process messages of type MessageName until X units of time have expired. After that it should stop processing messages of this type and continue its computation. On the other hand, the **receive** statements without the time-out check the message queue of the calling entity and continue immediately if no messages matching their type are present.

We use **send M to E** expression to signify the entity's intent on sending a message M to some other entity E. The **send** command is asynchronous and returns immediately, allowing the sending entity to continue its computation. The **send M to E in X** is an instruction to deliver message M to entity E in X time units. An entity can use this to send alarm messages to itself.

8.4 Programs

The following are the programs for the three entities in our model:

8.4.1 Directory

```
1. wait for messages:
2.     receive ResourceLocationsRequest(M):
3.         R := resources satisfying M.constraints in local database
4.         send ResouceLocators(R.URL) to M.sender
```

8.4.2 Resource

```
1. Let A be a list of available time blocks
2. wait for messages:
3.     receive AvailabilityRequestMessage(M):
4.         send AvailabilityInformation(A) to M.sender;
5.     receive ReserveOptionRequest(M):
6.         if M.time block in A then // if block is available
7.             remove M.time block from A; mark M.time block as optioned;
8.             send OptionExpired(M.time block) to self in M.dt; // set option expiration alarm
9.             charge M.sender for the option
10.            send ReserveOptionCommit(M.time block) to M.sender;
11.        end if
12.    receive OptionExpired(M):
13.        // check if the consumer has reserved its block
14.        if(M.time block is not marked reserved) then
```

```
15.                 mark M.time block as free // option has expired
16.                 add M.time block to A // add it back to the free list
17.                 send OptionExpiredMessage(M) to M.sender
18.             end if
19.         receive ReservationRequestMessage(M):
20.             if (M.time block in A OR M.time block marked optioned) then
21.                 mark M.time block reserved
22.                 charge M.sender for the reservation
23.                 send ReservationSuccessfulMessage(M) to M.sender;
24.             else
25.                 send ReservationFailedMessage(M) to M.sender;
26.         receive ReservationCancellationMessage(M):
27.             if (M.time block marked as reserved) then
28.                 add M.time block to A
29.                 unmark M.time block
30.                 charge M.sender for the cancellation
31.             end if
```

8.4.3 Consumer

```
1. Let Resources be a list.
2. send ResourceLocationsRequest(graph, resource constraints) to Directory
3. wait for messages:
4.         receive ResourceLocators(M) timeout in X:
5.             for R in M.resources:
6.                 send AvailabilityRequestMessage to R;
7.             end for
8.         receive AvailabilityInformation (M) timeout in Y:
9.             add M.information to Resources;
10. solution = Find optimal resources for the process; // see Chapter 5
11. // atomic reservation
12. for resource in solution do:
13.         send ReserveOptionRequest(time block, dt) to resource;
14. end for
15. wait for messages:
16.         receive ReserveOptionCommit(M) timeout in X:
17.             mark resource as optioned
18. if all resources are optioned then:
19.         for resource in process do:
```

```
20.             send ReservationRequestMessage(time block) to resource;
21.         end for
22.     else
23.         break;
24.     end if
25.     wait for messages:
26.         receive ReservationSuccessfulMessage(M) timeout in X:
27.             mark resource as reserved
28.     if all resources are reserved then:
29.         SUCCESS
30.         exit;
31.     else
32.         FAILURE
33.         // cancel all reservations and pay penalties
34.         for resource in solution do:
35.             send ReservationCancellationMessage(time block) to resource;
36.         end for
37.     end if
```

Chapter 9

Applications

The application that inspired this work is Crisis Management. Our goal has been to automate information and resource management in crisis situations. In 1998 our example case was the crash of TWA flight 800 in New York, where dozens of agencies in the local, state and federal governments had to come together to deal with the disaster. Unfortunately, the September 11 event has redefined the meaning of disaster. The need for automating as many aspects of the crisis management as possible has never been greater.

In a crisis, government agencies, businesses and individuals have to communicate and do their respective jobs as efficiently as possible. From the resource management point of view, each job (process) needs numerous resources, and choosing the best is of paramount importance. For example, rushing a victim to the nearest available hospital on the closest ambulance can make the difference between life and death.

As we started to develop a model for the problem, we realized that businesses face their own kind of crisis in every-day operations. In a crisis time is the scarce resource; the same is true in business, with cost as an additional constraint. The difference in the specification is in semantics, not the formalism. Just as an efficient resource assignment can save a victim of a disaster, the same can save a corporation from closing its doors and laying off hundreds of employees. If businesses can find and use better resources, they become more efficient and productive. If these resources are the services provided by other companies, then there is a clear value for any business in optimizing the selection and management of services it uses for its business processes.

Today the American economy is more than 80% a service economy. Different sectors are more or less conducive to optimization via the model we propose. In order to use the model we require a communication infrastructure between the resources and the consumers. Both the resources and the consumers need to be "on-line" so that they can find and communicate with each other. If a network is in place, then we provide a method for the consumers to choose the best resources and then a way to reserve them all together.

We propose a way we envision this technology could be used for crisis management. We also

give two applications for different industries that are both very service oriented, however only one – Travel – currently satisfies the "connectedness" requirement. The other is on its way, offering some communication ability, but no automated infrastructure in place. However we believe all three applications can directly benefit from our model and the related optimization methods.

9.1 Crisis Management

In a crisis individuals and organizations who may have never had any business in common have to come together to deal with the disaster. For example, in the TWA crash NYNEX provided hard phone hookups for the mobile and temporary command vehicles, AT&T provided cell phones for site personnel and the Long Island Lighting Company provided generators for electrical power to the command vehicles.

The number one lesson learned from the Flight 800 investigation, according to the NY State Emergency Management Office, is "The need for implementation of the Incident Command System (ICS) when responding to a single agency or multi-agency/multi-jurisdictional incident"[25]. Under an Incident Command System the response would be organized in four sections: operations, planning, logistics and finance/administration, all geared towards efficient communication and management of resources.

We believe that software technology can play a key role in ICS if it offers at least the following three capabilities: 1) a reliable communication infrastructure between the parties who must talk to one another, 2) a condition detection and event notification system that identifies the important events and forwards them to the decision makers, and 3) an automatic system for assigning the optimal resources to tasks. The first capability can be provided by an instant wireless network connected to the Internet at the low level, with an object oriented or database middleware on top. The second capability is currently being developed by companies like iSpheres Corp., that creates products for enterprise command and control. This thesis offers a way to achieve the third capability – optimal resource assignment of agencies' resources, related in time, space, load and capability.

9.2 Travel

Online travel reservations is one of the few services that most consumers would identify as a truly useful application of the Internet technology. Companies that offer these services, such as Travelocity, Expedia and Orbitz are selling millions of tickets each year. On these sites the entire transactions take place online without direct participation of the seller. While this is not a new phenomenon for goods (e.g. automated Coke machines), it is a debut for services, especially on this scale.

The travel services have a means of electronic communication with the customers – via a web

browser over the Internet, and the resources offer their availability information upon request. Thus, the infrastructure is in place to apply our model. Specifically, the resources that most consumers are interested in leasing for different segments of their trips are flights, hotels and rental cars. The relationships between them are in time, location and cost incentives. For example, the airline industry usually gives substantial discounts for round-trip ticket fares compared to one way. Therefore all the flights of a trip can be linked by a "price discount (cost)" relationship. These relationships can be directly represented with our model as we have suggested throughout this document.

Figure 9.1 shows a dependency graph for an example trip to Europe from Los Angeles, where a sales person visits three cities, London, Paris and Rome, and returns back home. The dependencies are time, distance and cost that the person needs to consider when making the bundle selection and reservation of all the services.

Formally his problem can be specified as follows:

Problem 9.1

Given a business process "European trip" that has 13 activities and 18 relationships we formalize it as follows:

Let $X = \{x_1, ..., x_{13}\}$*, where the variables have the following meanings:*

Variable	*Activity*
x_1	*LA Airport Car Garage*
x_2	*Shuttle to LA Terminal*
x_3	*Shuttle from LA Terminal*
x_4	*Flight from LA to London*
x_5	*Flight from Rome to LA*
x_6	*Hotel in London*
x_7	*Rental Car in London*
x_8	*Chunnel Train from London to Paris*
x_9	*Hotel in Paris*
x_{10}	*Rental Car in Paris*
x_{11}	*Flight from Paris to Rome*
x_{12}	*Hotel in Rome*
x_{13}	*Rental Car in Rome*

and let the relationships F *be defined as follows:*

$$F = \left\{ \begin{array}{l|l} \textit{Relationship} & \textit{Meaning} \\ f_1 : x_1 \times x_2 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_2 : x_1 \times x_3 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_3 : x_2 \times x_3 \rightarrow \mathbb{R}^+ & \textit{cost (same company discount)} \\ f_4 : x_2 \times x_4 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_5 : x_3 \times x_5 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_6 : x_4 \times x_6 \rightarrow \mathbb{R}^+ & \textit{time, distance} \\ f_7 : x_6 \times x_8 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_8 : x_7 \times x_8 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_9 : x_8 \times x_{10} \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_{10} : x_8 \times x_9 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_{11} : x_6 \times x_9 \times x_{12} \rightarrow \mathbb{R}^+ & \textit{Hotel cost (same company discount)} \\ f_{12} : x_9 \times x_{11} \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_{13} : x_{10} \times x_{11} \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_{14} : x_{11} \times x_{12} \rightarrow \mathbb{R}^+ & \textit{time, distance} \\ f_{15} : x_{11} \times x_{13} \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_{16} : x_7 \times x_{10} \times x_{13} \rightarrow \mathbb{R}^+ & \textit{Car cost (same company discount)} \\ f_{17} : x_{13} \times x_5 \rightarrow \mathbb{R}^+ & \textit{time (start / end)} \\ f_{18} : x_4 \times x_{11} \times x_5 \rightarrow \mathbb{R}^+ & \textit{Flight cost (same company discount)} \end{array} \right.$$

and $\phi = \sum_{i=1}^{18} f_i$. *Find an n-vector* $x_{opt} = (\dot{x}_1, ..., \dot{x}_{13})$*that maximizes* ϕ.

The dependency graph of the problem has a maximum clique of 3 and the MinDegree greedy heuristic yields a complexity of $O(12 * m^4)$.

9.3 Building Construction

Building construction is a well-structured business process that relies both on material resources and services. Often the construction of a large office building involves a managing construction company that subcontracts certain jobs to other firms. Internally, it manages dozens of human and vehicle resources to get the job done. These include carpenters, architects, electricians, digging equipment and cranes that operate with time and space relationships to each other. For example, two cranes cannot occupy the same part of the construction site.

There are several online services that exist for the construction industry. For example, Build-Point.com helps find general contractors, subcontractors, suppliers, equipment and professionals of all trades. Another company, BuilderAct.com is an online procurement hub that attempts to connect contractors with suppliers.

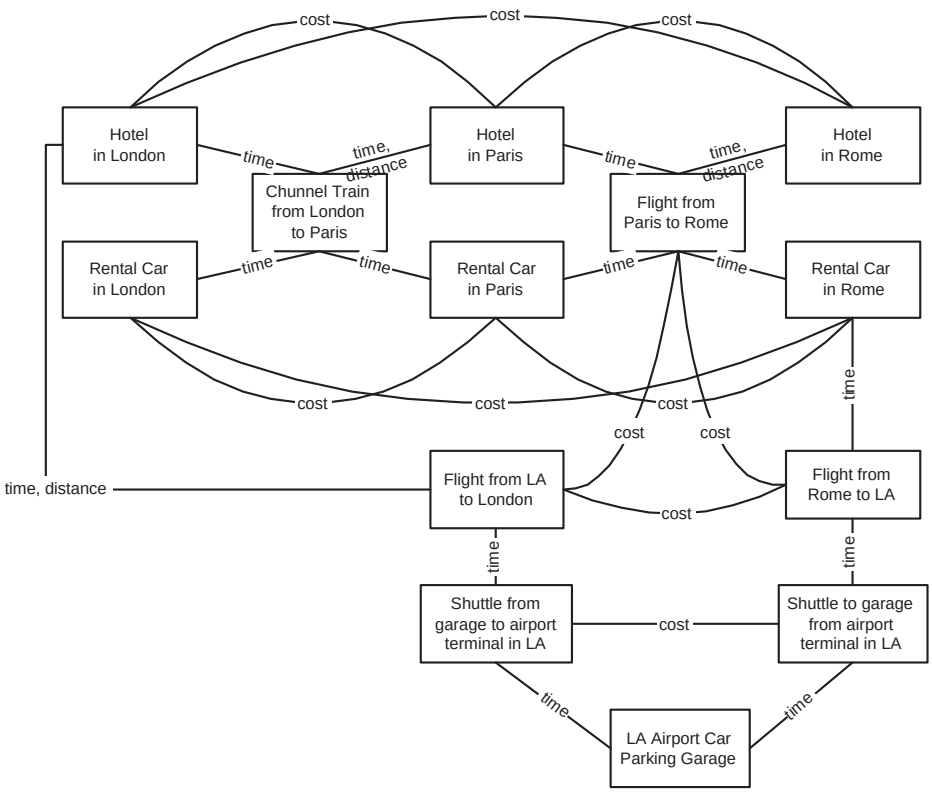


Figure 9.1: Dependency graph for European Trip Problem

These online services, among others, are moving the construction segment of the industry to an electronic infrastructure. Because of the structure of the business process and specialization of resources already present in the construction business, we believe it is a good candidate for optimization in the near future.

Chapter 10

Conclusion

10.1 Summary

In this thesis we identified two problems, the Activity Resource Assignment and the Distributed Service Commit, which we believe to be basic to all distributed applications in a networked service economy. We provided a formal model and a solution to each, making them immediately useful in practice.

10.1.1 Activity Resource Assignment

We specified the ARA problem, a type of Integer Programming, in a form that enabled us to find a special structure of the problem for many practical instances. We gave a method to search for an optimal by recursively reducing a problem instance without loosing at least one of the optima. We also gave an explicit way to represent the structure of a problem using the notion of variable independence. We showed two methods for finding solutions using the structure representation (dependency graph) and the reduction (relaxation) technique : two greedy search heuristics (MinDegree and MinEdgesCreated) and a method that builds on the existing work in graph complexity theory to guide the search via graph decomposition techniques. Both approaches can find exact solutions to many ARA instances in polynomial time.

We showed that the ARA-decision problem is NP-complete, which implies that the same solution method can be used to solve certain classes of other NP-complete problems in polynomial time. The degree of the polynomial depends on the 'complexity' of a given problem instance, which our method can compute quickly before proceeding to solve the problem. This two-stage approach to optimization allows a user to decide whether the method is acceptable for the given problem before carrying out the optimization.

10.1.2 Distributed Service Commit

The DSC problem addresses the issue of achieving an agreement among several parties in a networked environment when time has value. We described a new financial instrument, the Micro-Option, derived from the American Call Option adapted for services, which when used as part of a distributed agreement algorithm can secure a commitment from all the parties if one is possible. The need for this new approach arose because the resources required by the consumers in our model operate as services traded in a free-market economy. The time in this model has value, and our agreement algorithm explicitly addresses the resources' need to be compensated for their time during the reservation process. The traditional approaches do not take the time into account and therefore cannot be put to practical use in this environment.

Together the two solutions we offer can help automate resource management for many business processes that exist today and those we envision in the future. We believe they would pave way for a more efficient business operations and ultimately higher productivity for businesses and individuals alike.

10.2 Limitations

Specifying a business process may be hard without the help of semi-automated tools developed for individual industries. For the construction industry and crisis management, for example, only experts in the field can produce such a specification. For travel the situation may be somewhat easier. While this is a barrier to adoption, we believe the benefits of this form of automation would drive the development of 'template' business process specifications for many industries.

Our optimization method to solve the ARA problem works well for problems with relations defined on small subsets of X and with few interactions among relations in the objective function. The worst case is where all the variables are dependent, making the dependency graph a clique, which would happen if the specification were to contain a relation that expresses a global property that involves all the activities.

The reservation method requires that the economic infrastructure be sufficiently lean to make the costs of Micro-Options be close to the opportunity costs and virtually insignificant compared to the cost of the reserving the whole resource. This assumption is realistic, since call and put option prices on the standard market exchanges closely reflect the risks associated with their duration.

10.3 Future Work

First, in our optimization solution we identify four types of objective functions for which we can find the dependency sets efficiently. These functions are sufficient for our applications, however other

types should be explored as well. The larger the set of treatable objective functions the larger range of problems one could solve with this method. Thus, there is a clear value in doing further research in this direction.

Second, it would be useful to modify the first part of the optimization algorithm to not only give a complexity metric, but also suggest possible ways to change the constraints to reduce the complexity value. It may be possible to 'process' the relationships, combining them in certain ways that would reflect the user's desires almost as well as the original specification, but would yield a better complexity for our method.

Bibliography

[1] Arnborg, S., Corneil, D., and Proskurowski, A. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods 8* (1987), 277–284.

[2] Arnborg, S., Corneil, D., and Proskurowski, A. Complexity of finding embeddings in a k-tree, 1987.

[3] Belegundu, A. D., and Chandrupatla, T. R. *Optimization Concepts and Applications in Engineering*. Prentice-Hall, 1999.

[4] Berry, A., Bordat, J. P., and Cogis, O. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science 11*, 3 (2000), 397–403.

[5] Bodlaender, H., Kloks, T., and Kratsch, D. Treewidth and pathwidth of permutation graphs, 1995.

[6] Bodlaender, H., and Mohring, R. The pathwidth and treewidth of cographs, 1993.

[7] Bodlaender, H. L. Some classes of graphs with bounded treewidth. *Bulletin of the European Association for Theoretical Computer Science 36* (1988), 116–126.

[8] Bodlaender, H. L. A tourist guide through treewidth. *Acta Cybernetica 11* (1993), 1–21.

[9] Booz, Allen & Hamilton consulting firm. Program evaluation and review technique [pert], 1958.

[10] Clearwater, S. H., Ed. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[11] Compositional Systems Group. *Infospheres Infrastructure version 2.0 Users Guide*, 2 ed. California Institute of Technology, 1998.

[12] Dantzig, G. Linear programming and extensions, 1963.

[13] Fetzer, C., and Cristian, F. An optimal internal clock synchronization algorithm. In *Proceedings of the 10th Annual IEEE Conference on Computer Assurance* (Gaithersburg, MD, June 1995), pp. 187–196. http://www-cse.ucsd.edu/users/cfetzer/OCS/ocs.html.

[14] Garey, M. R., and Johnson, D. S. *Computers and Intractability A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, 1979.

[15] Gavril, F. Algorithms on clique separable graphs, 1977.

[16] Gray, J. N. *Operating Systems: An Advanced Course*, vol. 60 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978, ch. Notes on Database Operating Systems.

[17] Habib, M., and Mohring, R. Treewidth of cocomparability graphs and a new ordertheoretic parameter, 1992.

[18] Kelley, J. The critical–path method: Resources planning and scheduling, 1963.

[19] Kloks, T. Treewidth of circle graphs.

[20] Kloks, T., and Kratsch, D. Treewidth of chordal bipartite graphs. *J. Algorithms 19*, 2 (1995), 266–281.

[21] Koster, A., Bodlaender, H. L., and van Hoesel, S. P. *Treewidth: Computational Experiments*. ZIB Report 01-38. Konrad-Zuse-Zentrum fr Informationstechnik Berlin, 2001.

[22] Leimer. Optimal decomposition by clique separators. *DMATH: Discrete Mathematics 113* (1993).

[23] Matousek, J., and Thomas, R. Algorithms finding tree-decompositions of graphs. *Journal of Algorithms 12* (1991), 1–22.

[24] Neftci, S. N. *An Introduction to the Mathematics of Financial Derivatives*, 2 ed. Academic Press, April 2000.

[25] NYSEMO. After action report on the crash of the twa flight 800. Tech. rep., The New York State Emergency Management Office, http://www.nysemo.state.ny.us/TWA/LESSONS.HTM, 1998.

[26] Object Management Group(OMG). *The Common Object Request Broker: Architecture and Specification (CORBA), revision 2.0*, 1998.

[27] Ohtsuki, T., Cheung, L., and Fujisawa, T. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix, 1976.

[28] Robertson, N., and Seymour, P. Graph minors. xiii. the disjoint paths problem. *Journal of Combinatorial Theory Series B 63* (1995), 65–110.

[29] Robertson, N., and Seymour, P. D. Graph minors. ii. algorithmic aspect of tree-width. *Journal of Algorithms 7* (1986a), 309–322.

[30] Robertson, N., and Seymour, P. D. Graph minors. iv. tree-width and well-quasi-ordering. *Journal of Combinatorial Theory B 48* (1990a), 227–254.

[31] Skiena, S. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, 2003. combinatorica.m.

[32] Sprecher, A. *Resource-Constrained Project Scheduling*. Springer-Verlag, 1994.

[33] Sun Microsystems. *Java Message Service API*. http://java.sun.com/products/jms/index.html.

[34] Sundaram, R., Singh, K. S., and Rangan, C. P. Treewidth of circular-arc graphs. *SIAM Journal on Discrete Mathematics 7*, 4 (1994), 647–655.

[35] Tarjan, R. E. Decomposition by clique separators. *Discrete Mathematics 55* (1985), 221–232.

[36] Wimer, T. V. *Linear Algorithms on k-Terminal Graphs*. PhD thesis, Clemson University, Dept. of Computer Science, 1987.

[37] W.J. Cook, W.H.Cunningham, W., and A.Schrijver. *Combinatorial Optimization*. John Wiley and Sons, 1998.

Index