UNIVERSITY OF MINNESOTA

This is to certify that I have examined this bound copy of a doctoral thesis by

John E. Collins

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Maria Gini

Name of Faculty Advisor

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

Solving Combinatorial Auctions with Temporal Constraints in Economic Agents

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

John E. Collins

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Maria Gini, Advisor

June 2002

Dedication

To Geoff and Meg

who make it all worthwhile

Acknowledgments

I am deeply grateful to my advisor, Professor Maria Gini, who has a remarkable ability to see and articulate the core problem in a maze of confusion. She has been a good friend and a superb mentor to a student who is older than she is, and I'm sure it hasn't always been an easy role for her. I came back to school after many years in industry, thinking I already knew what I needed to know, with grandiose visions of what we could work on. I was quickly and gently disabused of these notions. Under her guidance, we backed off, articulated a vision, and started working out the specific areas where we could accomplish something that would be interesting. The result is not only this dissertation, but an ongoing research program that will keep a number of students busy for some years to come.

The many members of the MAGNET research team, past and present, have been good friends and have brought considerable enthusiasm and talent to our many long discussions. There are too many of them to list, but I would especially like to single out Bamshad Mobasher, now on the faculty at DePaul University, for help in refining our initial ideas, and for contributions and editing on several papers. Rashmi Sundareswara, Max Tsvetovat, Tim Lee, and Erik Steinmetz helped build the software that ran the experiments reported here, and Paula Greve, Dave Kohn, Len Josephs, and Chris Essex worked like mad to prepare the demo we presented at the Agents conference last year. Corey Bilot first conceived the idea of using Expected Utility Theory as an organizing principle in agent decision-making, was co-author on the first paper in which we began to develop that idea, and was kind enough to throw buckets of cold water on many dumb ideas. Alex Babanov has brought a strong background in Economics and a wealth of creative research ideas to the project, and has been able to take the expected-utility idea much further than I could. Wolf Ketter contributed his experience in wine making to the example in Chapter 4, and his thoughtful criticism and dry humor to many fruitful discussions.

Professor Mats Heimdahl, in his role as DGS of the Master of Science in Software Engineering program, always on the lookout for people with industry background who can teach, encouraged me to try my hand at teaching. I have found it challenging and highly rewarding, and the income has made it much easier to concentrate on my research. The students in the program are working professionals, and they are not afraid to challenge the professor! I am sure I have learned more from them than they have from me.

Karl Nilsson, an MSSE student of mine, took an interest in our project and introduced me to his boss, Kurt Hanson, president of Appian Bridge. Kurt introduced me to Ken Schumacher, president of North Star Import Export. Our work with them may have failed to produce the wealth of hard data we had hoped for, but it did give us a good view of the real-world complications that we would eventually have to deal with, and both Kurt and Ken have embraced our vision wholeheartedly.

Thanks to the members of my committee, Professors Mats Heimdahl, John Riedl, Art Hill, and Robert King, who have generously made time in their busy schedules to review my work.

Thanks to the many in the wider research community who have taken the time to discuss my work

and explain their work, including Holger Hoos, David Parkes, Michael Rothkopf, Jeff Kephardt, Juan Aguilar, Andy Williams, Brenda Dietrich, Katia Sycara, Nicola Muscettola, Michela Milano, Mark Boddy, Joachim Walser, and especially Tuomas Sandholm, who took time at the 2000 Agents conference to discuss and critique my work and my presentation, and at the 1997 AI conference to discuss the problems of moose interacting with automobiles and vegetable gardens. Thanks also to the many anonymous reviewers who have greatly improved the quality of my work.

The National Science Foundation has supported our work generously, under grants NSF/IIS-0084202 and NSF/EIA-9986042. Without this funding, I would not have had the freedom to concentrate on the completion of this work.

Bill Estrem repeatedly reminded me that the most important thing about a dissertation was having it finished, gave me a forum for testing some of my ideas with his students in his courses at the University of St. Thomas, and argued with me late into the night on several occasions over the role of technology in business. Bill Moen, my old friend who went from vagabond to trucker to merchant to librarian to professor, showed me that it's never too late. Elizabeth Sisley, who finished her degree while living an impossibly complicated and busy life, made me realize that I have nothing to complain about.

The majority of our work is done using open-source software, including gnu/Linux, LaTeX, ant, log4j, and other tools from the Apache project, octave, emacs, and many others. Thanks to the talented folks who make all this possible.

Finally, and most importantly, I appreciate and depend on the support and encouragement of my family. My wife Anne has put up with long hours, minimal income, and much distraction for several years, all with loving kindness and good humor. My son Geoff beat me in the Ph.D. race by two years, and doesn't let me forget.

Abstract

We consider the problem of rational, self-interested, economic agents who must negotiate with each other in order to carry out their plans. Customer agents express their plans in the form of task networks, which they offer in a marketplace in a request for quotations (RFQ). The market runs a combinatorial reverse auction, in which supplier agents may submit bids that specify prices for combinations of tasks, along with time windows and duration data that the customer may use to compose a work schedule. The presence of temporal and precedence constraints among the items at auction requires extensions to the standard winner-determination procedures for combinatorial auctions, and the use of the enhanced winner-determination procedure within the context of a real-time negotiation requires that we predict its runtime when planning the negotiation process.

We propose a set of requirements for a market infrastructure that can support this type of negotiation, and describe an architecture that can meet these requirements. It supports the interactions of a customer and a set of suppliers as they negotiate over a particular plan, helps customers find suppliers who can meet their needs, provides an ontology that defines the types of services that are available in the market, collects and publishes statistics in support of agent decision-making, and provides a security infrastructure.

We describe the high-level design of an agent that can act as a customer in this environment, and describe the decision behaviors such an agent must implement to maximize its utility. The design allows behaviors to be defined in terms of events, time, and responses to events. Events can represent external occurrences, or internal state changes. Decisions include creating a task network to represent a plan that will achieve a goal, creating a bid-process plan for carrying out negotiations that will gain the necessary commitments to execute the plan, possibly in multiple phases across multiple markets, allocating time to the various stages of the negotiation process, defining a request for quotations that will attract the most advantageous set of bids, analyzing the bids to determine auction winners, building a work schedule from the winning bids, and monitoring plan execution and responding to unexpected events.

Methods for the determination of auction winners are explored in detail, and three different algorithms are presented. One is an Integer Programming (IP) model, the second is a queue-based Simulated Annealing (SA) design, and the third is an extension of the bidtree-based Iterative-Deepening A* (IDA*) formulation proposed by Sandholm. We examine a wide range of tuning options for the simulated annealing algorithm, and we propose a particular way to structure the bidtree to optimize the performance of the IDA* algorithm.

The winner-determination algorithm must be run in the context of a real-time negotiation, and the time available to run it must be published to suppliers as part of a request for quotations. This is because suppliers need to know when they may expire their bids, and because shorter bid-expiration times are expected to lead to lower prices and higher willingness to bid on the part of suppliers. The result is that we must be able to predict the runtime of these $\mathcal{NP}$-complete algorithms based on information that can be measured or estimated before bids are available. We report on a series of experiments that provide us with runtime probability distributions across a range of problem

size parameters. We also show detailed profiles of the behavior of the SA algorithm, following the method of Hoos and Stützle. We show that there is little or no correlation between the IP and SA methods or between the IP and IDA* methods when compared on a problem-by-problem basis over large sets of problems with similar statistics. In other words, individual problems that are hard for one method are not necessarily hard for another method. This gives us some hope that variability could be managed by combining search methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation is about Autonomous Agents in Electronic Commerce. We believe that much of the commercial potential of the Internet will remain unrealized until a new generation of autonomous systems is developed and deployed. A major problem is that the global connectivity and rapid communication capabilities of the Internet can present an organization with vast numbers of new opportunities, to the point that users are overwhelmed, and conventional automation offers little help.

Much has been done to enable simple buying and selling over the Internet, including systems like EBay, CommerceOne, and tools for enabling online business transactions from vendors like IBM, Microsoft, ATG, Broadvision, and many others. Much is also being done to help customers and suppliers find each other, from search engines like Google to vertical industry portals like Covisint to personalization systems and recommender engines like Net Perceptions. However, many business operations are much more complex than the simple buying and selling of individual items. We are particularly interested in situations that require coordinated combinations of goods and services, where there is often some sort of constraint-satisfaction or combinatorial optimization problem that needs to be solved in order to assemble a "deal." Commonly, these extra complications are related to constraints among task and services, and to time limitations. The combinatorics of such situations are not a major problem when an organization is working with small numbers of partners, but can easily become nearly insurmountable when "opened up" to the public Internet.

We envision a new generation of systems that will help organizations and individuals find and exploit opportunities that are otherwise invisible or too complex to seriously evaluate. These systems will help potential partners find each other (matchmaking), negotiate mutually beneficial deals (negotiation, evaluation, commitment), and help them monitor the progress of distributed activities (monitoring, dispute resolution). They will operate with variable levels of autonomy, allowing users to delegate or reserve authority as needed, and they will provide users with a market presence and power that is far beyond what is currently achievable with today's telephone, fax, web, and email-based methods. We assume that the primary negotiation paradigm among these systems will be

market-based combinatorial auctions, with added precedence and temporal constraints.

The Multi-AGent NEgotiation Testbed (MAGNET) project represents a first step in the process of developing this vision into reality. MAGNET provides a unique capability that allows self-interested agents to negotiate over complex coordinated tasks, with precedence and time constraints, in an auction-based market environment. This dissertation introduces many of the problems a customer agent must solve in the MAGNET environment, explores in detail a set of algorithms for solving the extended combinatorial-auction winner determination problem, and characterizes their performance.

1.1 The evolution of e-commerce

Recently, the complexity of logistics involved in manufacturing and in many other areas of commerce has been increasing nearly exponentially. Many processes are being outsourced to outside contractors, making supply chains longer and more convoluted. The increased complexity is often compounded by reduced inventories and accelerated production schedules which demand tight integration of processes across multiple self-interested organizations. At the same time, much commerce is moving on line, were firms can cut costs and improve efficiency by taking advantage of reduced transaction costs, expedited order cycles, and dynamic pricing available in the network environment. Instead of fulfilling orders from warehouses, companies are looking for partners that can provide coordinated components and services in a timely manner to meet customer demand for made-to-order products. Thus, the field is ripe for the introduction of systems that automate logistics planning among multiple entities such as manufacturers, part suppliers and specialized subcontractors.

Current e-commerce systems typically rely on either fixed-price catalogs or simple auctions to set prices, and either industry portals or general search engines to find potential suppliers and customers. Companies often work with pre-qualified suppliers in order to manage risk and complexity, and buyer-supplier relationships depend on factors such as quality, delivery performance, and flexibility, in addition to cost [Helper, 1991]. In addition, most current e-commerce systems do not have any notion of time (although some can deal with delivery time or lead time), and only the simplest of constraints can be expressed. Exceptions include domain specific systems such as SABRE used in the travel industry, where one may search for connecting flights. Time and precedence constraints play a fundamental role in supply-chain formation and management, since many products are made up of different parts and require multiple suppliers who have to coordinate their work.

To see what is happening at the current "cutting edge" of commercial e-commerce in 2002, let us examine two companies who are focused on business-to-business commerce automation, Ariba and i2. Ariba systems help manage purchasing functions, including "direct" spending that supports production (raw materials, parts, etc.) and "indirect" spending that supports administrative operations (pencils, computers, etc.). Their systems integrate closely with ERP and manufacturing applications at multiple stages in a supply chain, allowing the sharing of forecasts, production volume data, and other information to help "downstream" organizations adapt to "upstream" conditions, and allowing the upstream organizations to anticipate the downstream needs. A high proportion of the paperwork involved in a typical purchasing cycle is eliminated, along with the accompanying delays, errors, and

much of the cost. Ariba systems can support forward and reverse auctions, using standard auction models.

i2 started with high-end production scheduling systems. They have extended their reach from the manufacturing floor into the supply chain, with many similar capabilities to Ariba's offerings. A major piece of their current offering is a capability for close coordination of production processes at multiple levels in a supply chain. This is typically done between companies that have entered into a long-term strategic partnership, and the implementation and integration costs can be substantial. Our approach envisions a more arms-length relationship among organizations who must coordinate their activities.

1.2 Motivating examples

To clarify our ideas a bit, here are several example scenarios from different industries that we think might be possible if the vision behind MAGNET were to be further developed and commercialized.

Flexible manufacturing – Much small-to-medium scale manufacturing is being done on a contract basis, and many of the products manufactured this way are seasonal (Christmas decorations) or have short product lifetimes (fad toys). There are "virtual" manufacturing organizations that have no plants, but build all their products on contract. We envision streamlining this type of business, and allowing many more organizations to participate, by greatly simplifying the process of building a supply chain to produce a product "on demand." Armed with a product design, a process plan to produce the product, and a budget, a product manager could tap into a network of automated agents to find the necessary resources, optimize prices and schedules, and let the contracts to get his product produced.

Mass customization – Driven by competition and rising customer expectations, many manufacturing industries are introducing the concept of "mass customization" to manufactured products. These are mass-produced products that are customized for each sale. Examples include Dell Computer, where desktop and server computer systems are assembled to order, with a wide variety of configurations and options available. Much of the output of the domestic auto industry is customized to order as well. Currently, most customization is handled at final assembly, by simply adding or not adding various options, or choosing colors and finishes. If a manufacturer had access to a rich on-line market populated by automated agents to handle the negotiations, a much deeper level of customization would be possible, with individualized operations extending back into the supply chain.

Travel arrangements – Making complex travel arrangements involving heavily-booked destinations and seasons can be risky. Travelers and their (human) agents must make guesses about which resources are most scarce, and plan around them. If they guess wrong, they can be stuck with reservations they can't use. The problem is that they generally can't reserve "bundles" in a single operation, because the various resources are under control of different organizations. This is one of the drivers behind consolidation in the industry, but an agent-mediated online

market for travel services would facilitate the evaluation of large numbers of alternative resources and schedules before making a commitment to a complete package. It would also help put smaller providers on a more equal footing with their larger competitors.

International shipping – When a manufacturer wishes to export goods by the shipping container, there is a complex maze of carriers, brokers, and local and international laws and conventions to navigate. Containers may traverse five or more transport modes (truck, rail, ocean vessel, etc.) and be subject to multiple inspections and quarantines, which must often be mediated by local brokers. Transfer of funds through the international banking system requires that various documents be prepared and transferred at the correct times to the correct agencies.

Currently, there are businesses who employ large numbers of brokers using telephones and fax machines who do nothing but coordinate these arrangements for clients. An example is North Star Import/Export in Minneapolis, a company we studied as part of a data collection effort for the MAGNET project[1]. The real value these businesses add is in knowing the regulations and having contacts with the various carriers, customs agents, and financial institutions. They typically are able to evaluate only a small fraction of the available resource combinations for any given shipment, and it is clear that shipping costs are higher than necessary because of this. They would benefit greatly from an on-line agent-mediated market that would help them find more optimal combinations of resources and schedules for their clients.

Health care – Many large health-care organizations operate networks of clinics and other primary, secondary, and tertiary health-care facilities within a geographical region. When patients are referred from primary to secondary or tertiary facilities, there are often several options available. The driving factors in choosing facilities are often patient convenience and availability of specific diagnostic or therapeutic resources. Both patients and the health-care organizations could benefit from being able to optimize convenience, schedule, and cost considerations when arranging such resources, and to be useful, the alternatives need to be generated and evaluated while the patient is standing at the desk of the primary care facility discussing the options with the reservations clerk. An agent-mediated market environment could accomplish this, and could allow different organizations to share resources much more effectively than is currently possible.

Systems management – Large-scale managed hosting and application-service providers (ASPs) must balance the needs for interactive response with the needs of large batch jobs, such as payroll runs, with a continuously-shifting mix of available computing and communication resources. Economies of scale are often not fully realized because of the difficulty of discovering and exploiting available resources. IBM has initiated a project using its ABLE agent toolkit to develop a set of more autonomous system management capabilities, and they have expressed interest in using an internal "market economy" to manage resource allocations [Bigus, 2002]. For complex processing scenarios, the optimization capabilities of MAGNET agents could be very useful in this domain, but the market infrastructure would probably be less useful.

[1] The data itself turned out to be less useful than we anticipated, but the experience was enlightening, and North Star management was very supportive of our work and vision.

Many other examples could be worked out. In Chapter 4, we use an example from the wine-making industry.

1.3 Agents for Electronic Commerce

The term "Agent" has been used to mean many things in recent industry and academic literature, and has been generally degraded by marketing hype. According to Webster's Third New International Dictionary, an agent is "one that acts or exerts power... a means or instrument by which a guiding intelligence achieves a result... one that acts for or in the place of another by authority from him." Russell and Norvig say that "An agent is just something that perceives and acts" [Russell and Norvig, 1995]. Bradshaw reviews in detail the various meanings of the term as it has been used in the research community [Bradshaw, 1997]. The meaning we intend focuses primarily on the "agency" and "intelligence" dimensions as used by Bradshaw. By "agency", we refer to the notion that MAGNET agents have persistent existence and identity within an environment in which they can perceive, act, and observe the effects of their actions. We also assume that each of our agents acts on behalf of some human or organization, its "principal." By "intelligence", we mean that our agents are *rational*, to the limits of their computational capabilities. We use the term "rational" in the decision-theoretic sense, to mean that a MAGNET agent acts to maximize its own *utility* or the utility of its principal. A MAGNET agent exists in a market environment, and its utility is measured in economic terms. Therefore, we consider a MAGNET agent to be a *self-interested, rational, economic agent.*

MAGNET agents are "autonomous" in the sense that they are independent, and not merely components in a larger system. They may also exhibit a high degree of autonomous behavior, or they may be deployed as decision-support tools for human decision-makers. They are "self-interested" in that they are expected to behave in a way that maximizes their own utility functions, without regard to the utilities of other agents or of the society as a whole. They are "heterogeneous" in two senses: first, they do not have the same capabilities, and in general must find other agents to supply resources to satisfy their own goals; second, they may be completely different agent implementations in the same network environment, as long as they satisfy the same high-level APIs and follow the rules of the markets. This means that one can make no assumptions about the behaviors and strategies of other agents other than that they will follow the rules.

1.4 The MAGNET system

The MAGNET system was conceived as an environment that could support a variety of studies into automated negotiation among self-interested, autonomous agents. Experimental research in this area requires a simulation environment that is sufficiently rich to be easily adapted to a variety of experimental purposes, while being sufficiently straightforward to support clear conclusions. MAGNET is not a complete simulation of a working market environment. Instead, it is focused on the process of determining the form and content of Requests for Quotations (RFQs), on the management of the bidding process, and on the evaluation of bids submitted by potential suppliers. It has the ability

to generate plans with well-defined statistics, or to accept hand-built plans or plans extracted from real-world data. Bids are generated by a community of abstract suppliers, again with well-defined statistics. All the major decision processes are driven by plug-in components, with documented APIs and a great wealth of configuration parameters. Data collection capabilities are well-suited to statistical studies.

The MAGNET customer agent with its experimental framework, which was used for the experiments reported in Chapter 6, has been released under an Open Source license[2].

1.5 Research contributions

This work advances the state of the art in multi-agent automated contracting in a number of ways.

1. We have identified a need for market-based negotiation over complex sets of coordinated tasks, and articulated a vision for addressing that need, with autonomous or semi-autonomous agents that negotiate among themselves using an *extended combinatorial auction* that adds precedence and temporal constraints to the standard formulation.

2. We have developed a general *market architecture* to support negotiations among heterogeneous, self-interested parties. The market publishes an ontology that defines the terms of discourse among agents, acts as a trusted third party in negotiations, helps customers to find qualified suppliers, and gathers and publishes statistics that support various agent decision processes.

3. We present an *architecture* for a MAGNET Customer Agent that organizes and supports the agents principal activities. It includes a status hierarchy and a time map that makes it easy to predicate behaviors on internal state changes and external events, and the passage of time.

4. We outline the *decision processes* that a MAGNET Customer Agent must implement in order to perform its functions, and we enumerate external *events* that a MAGNET Customer Agent must sense and react to in order to supervise the execution of its plans.

5. We show how the *combinatorial auction winner determination* process must be extended to support the evaluation of bids that specify time windows along with prices for bundles of tasks that may have precedence constraints among them.

6. We present three different *algorithms* for solving the extended winner determination problem.

7. We explore the *deliberation scheduling* problem that MAGNET customer agents must solve, and we *characterize the runtime performance of our winner-determination search algorithms* on a variety of problems. The results of this analysis are needed by a customer agent to drive its deliberation scheduling process.

[2]see `http://www.cs.umn.edu/magnet`.

1.6 Guide to this thesis

Chapter 2 places this work in context with other work in the field. In particular, we draw on work in multi-agent negotiation, auction mechanism design, and combinatorial auction winner-determination, which has been a very active field in recent years.

Chapter 3 describes the MAGNET system in some detail, focusing on the market architecture and the design and functions of a customer agent.

Chapter 4 briefly describes a branch of economics called "Expected Utility Theory," and then works through a complete interaction scenario with an example problem, describing each of the decision processes a customer agent must implement in order to maximize the expected utility of its principal. For some of them, we have worked out how to implement the decisions, while for the remainder we only describe the problems.

Chapter 5 focuses on one specific decision problem, that of deciding deciding the winners in a MAGNET auction. We develop three different winner-determination algorithms, one with two variations. Two of them produce provably optimal results, while the third is a stochastic algorithm based on Simulated Annealing.

Chapter 6 describes the results of an experimental program to characterize the performance of our search algorithms. This is important because it is a difficult combinatorial problem, and the negotiation process requires that time be allocated to it before the details of the problem can be known. We show that we can use a combination of factors that can be measured or estimated at the appropriate time in the negotiation cycle to predict search time requirements with a specified probability of success.

Finally, Chapter 7 wraps up the discussion and points out a set of additional research topics that must be addressed to further realize the MAGNET vision.

Chapter 2

Related Work

This work draws from several fields. In Computer Science, it is related to work in artificial intelligence and autonomous agents, more specifically to work in automated negotiation and coordination, and search methods. In Economics, it draws from auction theory and expected utility theory. From Operations Research, we draw from work in combinatorial optimization.

2.1 Multi-agent negotiation

At a very abstract level, MAGNET proposes using an auction paradigm to support problem-solving interactions among autonomous, self-interested, heterogeneous agents. Several other approaches to multi-agent problem-solving have been proposed. Some of them use a "market" abstraction, and some do not.

Rosenschein and Zlotkin [Rosenschein and Zlotkin, 1994] show how the behavior of agents can be influenced by the set of rules system designers choose for their agents' environment. In their study the agents are homogeneous and there are no side payments. In other words, the goal is to share the work, in a more or less "equitable" fashion, but not to have agents pay other agents for work. They also assume that each agent has sufficient resources to handle all the tasks, while we assume the contrary. Sycara [Sycara, 1988] describes a system called PERSUADER in which agents negotiate over conflicting goals using an iterative combination of persuasion and compromise. The domain of the system was labor conflict mediation.

In Sandholm's TRACONET system [Sandholm and Lesser, 1995, Sandholm, 1996], agents redistribute work among themselves using a contracting mechanism. Sandholm considers agreements involving explicit payments, but he also assumes that the agents are homogeneous – they have equivalent capabilities, and any agent can handle any task. MAGNET agents are heterogeneous, and in the general case do not have the resources or capabilities to carry out the tasks necessary to meet their own goals without assistance from others.

A general model for negotiation among multiple parties over multiple issues is presented in [Faratin *et al.*, 1997]. This model could potentially be applied to the MAGNET domain to deal with schedule adjustments and failure recovery, but since it assumes a fixed set of negotiators and multiple rounds of offers and counteroffers, it is not easily adapted to the auction phase of negotiation, where the set of suppliers may not be known until they have submitted bids. It is also not capable of finding optimal solutions, and places a fairly heavy computational burden on all the parties, which would not be appropriate in a market situation until after some initial agreement was reached. In [Klein *et al.*, 2002], agents negotiate over contracts with multiple components with the help of a mediator. If the issues in dispute have interdependencies (the value of one issue depends on the presence or absence of another issue), then a situation arises that is similar to the Prisoner's Dilemma, in that the strategy that maximizes welfare for the two parties is not individually rational for either one.

The ultimate goal of MAGNET is to automate the scheduling/execution cycle of an autonomous agent that needs the services of other agents to accomplish its task. Both Pollack's DIPART system [Pollack, 1996] and the Multiagent Planning architecture (MPA) [Wilkins and Myers, 1998] assume multiple agents that operate independently. However, in both of those systems the agents are explicitly cooperative, and all work toward the achievement of a shared goal. MAGNET agents are trying to achieve their own goals and to maximize their own profits; there is no global or shared goal.

2.1.1 Solving problems using markets and auctions

MAGNET uses an auction-based negotiation style because auctions have the right economic and motivational properties to support "reasonable" resource allocations among heterogeneous, self-interested agents. However, MAGNET uses the auction approach not only to allocate resources, but also to solve constrained scheduling problems.

The WALRAS framework described in [Wellman, 1993] is a programming environment for distributed decision-making, in which auctions are used to solve transportation problems and other distributed optimization problems. Quantities and flows are determined by a centralized auctioneer that solves a general-equilibrium model. Agents submit bids consisting of supply and demand curves.

An iterative auction-based mechanism for the scheduling of railroad resources is described in [Parkes and Ungar, 2001]. In this system, individual train agents bid on the rights to use sections of track from multiple dispatchers. A solution for an individual train involves acquiring a string of such rights that extend from an origin to a destination, and respect sequence and speed constraints. For a dispatcher, a solution maximizes revenue while preserving safety constraints. The solution procedure is quite specific to the problem domain, using a matrix of entry/exit times containing current prices. The updated matrix is communicated to bidders at each round; bidders must use the matrices from a set of dispatchers to solve a shortest-weighted-path problem to compose bids for subsequent rounds. If we were to adapt this approach to MAGNET, it is hard to see how we could guarantee coverage or allow bidders to give aggregate prices on sets of tasks.

A set of auction-based protocols for decentralized resource-allocation and scheduling problems is proposed in [Wellman *et al.*, 2001]. The analysis assumes that the items in the market are individual discrete time slots for a single resource, although there is a brief analysis of the use of the Generalized Vickrey Auctions [Varian and MacKie-Mason, 1995] to allow for combinatorial bidding. A combinatorial auction mechanism for dynamic creation of supply chains was proposed and analyzed in [Walsh *et al.*, 2000]. This system deals with the constraints that are represented by a multi-level supply-chain graph, but does not deal with temporal and precedence constraints among tasks. MAGNET agents must deal with multiple resources and continuous time, but we do not currently deal explicitly with multi-level supply chains[1].

Agents in the MASCOT system [Sadeh *et al.*, 1999] coordinate scheduling with a user, but there is no explicit notion of payments or contracts, and the criteria for accepting or rejecting a bid are not clear. The system is designed as a set of agents that "wrap" existing production scheduling components and allow communication and coordination among independent organizations in a supply chain. An auction-based mechanism that optimizes unit parameters in a large chemical plant is described in [Jose and Ungar, 1998]. In this case, the individual units are each run independently, and the problem is to optimize the temperature, quantity, and other parameters on the interfaces among units. The auction mechanism is shown to be correct and efficient, and it respects the fact that the various units are typically run in a fairly autonomous fashion.

In [Hunsberger and Grosz, 2000] combinatorial auctions are used for the initial commitment decision problem, which is the problem an agent has to solve when deciding whether to join a proposed collaboration. Their tasks have precedence and hard temporal constraints. However, to reduce search effort, they use domain-specific *roles*, a shorthand notation for collections of tasks. In their formulation, each task type can be associated with only a single role. MAGNET agents are self-interested, and there are no limits to the types of tasks they can decide to do. In [Glass and Grosz, 2000] scheduling decisions are made not by the agents, but instead by a central authority. The central authority has insight to the states and schedules of participating agents, and agents rely on the authority for supporting their decisions. The thrust of this research is about inducing cooperation among independent agents by simulating a "social consciousness."

Several proposed bidding languages for combinatorial auctions allow bidders to express constraints, for example [Nisan, 2000a, Boutilier and Hoos, 2001]. However, these approaches only allow bidders to communicate constraints to the bid-taker (suppliers to the customer, in the MAGNET scenario), while MAGNET needs to communicate constraints in both directions.

2.1.2 Automated support for auctions in electronic commerce

Auctions of various flavors are becoming a predominant mechanism for agent-mediated electronic commerce. An early survey is [Guttman *et al.*, 1998], and a more complete version is [Maes *et al.*, 1999]. Much of this is consumer-oriented, although Sandholm's sidebar argues strongly for the use

[1] Individual MAGNET agents can deal with multi-level supply chains by subcontracting, but this requires that the initial time allocation provide sufficient slack for the extra negotiation cycles. We will deal with this issue in Chapter 4

of combinatorial auctions in business-to-business scenarios. AuctionBot [Wurman *et al.*, 1998] and eMEDIATOR [Sandholm, 1999] are among the most well known examples of multi-agent auction systems. They use economics principles to model the interactions of multiple agents [Wellman and Wurman, 1998]. Simple auctions are not always the most appropriate mechanism for the business-to-business transactions we are interested in, where issues such as scheduling, reputation, and maintaining long term business relations are often more important than cost.

2.1.3 Infrastructure support for negotiation

Markets play an essential role in the economy [Bakos, 1998], and market-based architectures are a popular choice for multiple agents (see, for instance, [Chavez and Maes, 1996, Rodriguez *et al.*, 1997, Sycara and Pannu, 1998, Wellman and Wurman, 1998] and our own MAGMA architecture [Tsvetovatyy *et al.*, 1997]). Most market architectures limit the interactions of agents to manual negotiations, direct agent-to-agent negotiation [Sandholm, 1996, Faratin *et al.*, 1997], or some form of auction [Wurman *et al.*, 1998]. The Michigan Internet AuctionBot [Wurman *et al.*, 1998] is a very interesting system, in that it is highly configurable, able to handle a wide variety of auction rules. It is the basis for the ongoing Trading Agent Competition [TAC-02, 2002], which has stimulated interesting research on bidding behavior in autonomous agents, such as [Stone *et al.*, 2002].

Existing architectures for multi-agent virtual markets typically rely on the agents themselves to manage the details of the interaction between them, rather than providing explicit facilities and infrastructure for managing multiple negotiation protocols. In our work, agents interact with each other through a market. The independent market infrastructure provides a common vocabulary, collects statistical information that helps agents estimate costs, schedules, and risks, and acts as a trusted intermediary during the negotiation process. We believe there are many advantages to be gained by having a market-based intermediary for agent negotiations, as explained earlier in the paper.

Matchmaking, the process of making connections among agents that request services and agents that provide services, will be an important issue in a large community of MAGNET agents. The process is usually done using one or more intermediaries, called middle-agents [Sycara *et al.*, 1997]. Kuokka and Harada [Kuokka and Harrada, 1995] describe an agent application whereby potential producers and consumers of information send KQML messages describing their capabilities and needs to an intermediary called a matchmaker. Sycara et al. [Sycara *et al.*, 1999] present a language that can be used by agents to describe their capabilities and algorithms to use it for matching agents over the Web. Our system casts the Market in the role of matchmaker.

The MAGNET market infrastructure depends on an Ontology to describe services that can be traded and the terms of discourse among agents. There has been considerable attention to development of detailed ontologies for describing business and industrial domains [Fox, 1996, Gruninger and Fox, 1994, Schlenoff *et al.*, 1998], and for describing commitment in multiagent systems [Singh, 1999].

2.2 Combinatorial auctions

Determining the winners of a combinatorial auction [McAfee and McMillan, 1987] is an $\mathcal{NP}$-complete problem, equivalent to the weighted bin-packing problem. A good overview of the problem and approaches to solving it is [de Vries and Vohra, 2001]. Dynamic programming [Rothkopf *et al.*, 1998] works well for small sets of bids, but it does not scale well, and it imposes significant restrictions on the bids. Sandholm [Sandholm, 1999, Sandholm, 2002] relaxes some of the restrictions and presents an algorithm for optimal selection of combinatorial bids, but his bids specify only a price and a set of items. However, the "bidtree" structure introduced in [Sandholm, 2002] has been adopted as a fundamental data structure in the MAGNET A* and IDA* search methods (see Section 5.3). A set of optimal and approximate methods, along with a test set for algorithm evaluation, was published by Fujishjima et al. [Fujishjima *et al.*, 1999]. This method, called CASS (Combinatorial Auction Structured Search) is a branch-and-bound technique that gains acceptable performance by preprocessing and caching of partial solutions. It does not appear to perform as well as Sandholm's bidtree method. Hoos and Boutilier[Hoos and Boutilier, 2000] describe a stochastic local search approach to solving combinatorial auctions, and characterize its performance with a focus on time-limited situations. A key element of their approach involves ranking bids according to expected revenue; it's very hard to see how this could be adapted to the MAGNET domain with temporal and precedence constraints, and without free disposal. Andersson et al. [Andersson *et al.*, 2000] describe an Integer Programming approach to the winner determination problem in combinatorial auctions. Nisan [Nisan, 2000a] extends this model to handle richer bidding languages for combinatorial auctions. More recently, Sandholm [Sandholm *et al.*, 2001] has described an improved winner-determination algorithm called CABOB that uses a combination of linear programming and branch-and-bound techniques. It is not clear how this technique could be extended to deal with the temporal constraints in the MAGNET problem, although the bid-graph structure may be of value.

A number of extensions to the basic combinatorial auction have been proposed. Varian [Varian, 1995] showed how to apply the second-price Vickrey pricing mechanism [Vickrey, 1961] to a combinatorial auction setting. The resulting prices can be counterintuitive, the winner-determination procedure requires solving multiple reduced problems (at least one additional winner-determination problem per winning bid), and this approach is not widely used.

One of the problems with combinatorial auctions is that they are nearly always run in a single round sealed-bid format, and this is the format MAGNET uses. Parkes and Ungar [Parkes and Ungar, 2000] have shown how to organize multiple-round combinatorial auctions. Another problem is that the individual items in a combinatorial auction are individual items; there is no notion of quantity. MAGNET will eventually need to address this. This limitation is overcome in [Leyton-Brown *et al.*, 2000b] for simple items without side constraints. The addition of precedence constraints would seriously complicate their procedure, and it has not yet been attempted.

A test suite for testing combinatorial auctions is proposed in [Leyton-Brown *et al.*, 2000a]. This suite has been used in a number of more recent papers, including [Hoos and Boutilier, 2000, Sandholm, 2002, Sandholm *et al.*, 2001]. There would be considerable value to adapting this suite or building

a similar problem set for auctions with constraints. Perhaps our work in Chapter 6 could be the foundation of such an effort.

2.3 Search

Because of a desire to moderate the variability of the Integer Programming approach, and the need for agents to make decisions according to a strict timeline, we have evaluated a simulated annealing framework as an alternative method. Since the introduction of iterative sampling [Langley, 1992], a strategy that randomly explores different paths in a search tree, there have been numerous attempts to improve search performance by using randomization. Randomization has been shown to be useful in reducing the unpredictability in the running time of complete search algorithms [Gomes *et al.*, 1998], although in our application is that variability remains quite high.

A variety of methods that combine randomization with heuristics have been proposed, such as Least Discrepancy Search [Harvey and Ginsberg, 1995], heuristic-biased stochastic sampling [Bresina, 1996], and stochastic procedures for generating feasible schedules [Oddi and Smith, 1997], just to name a few. Our stochastic algorithm is based on simulated annealing [Kirkpatrick *et al.*, 1983], and as such combines the advantages of heuristically guided search with some random search. See [Reeves, 1993] for a thorough overview of simulated annealing and other heuristic search techniques.

Two other search methods have been suggested, but not tested. One is the combination of local search and pseudo-boolean constraints described in [Walser, 1999], and the other is the combination of constraint-satisfaction search with logic programming as developed in [Hooker, 2000].

2.4 Search performance and evaluation

An important element of this research is the need to characterize the performance of the winner-determination search. The inspiration and the basic approach we use was originally borrowed from [Hoos and Stützle, 1998]. We have applied this approach not only to the evaluation of the Simulated Annealing algorithm on individual problems, as originally described by Hoos and Stützle, but also to the evaluation of the Integer Programming and A* search performance on sets of randomly-generated problems.

2.5 Deliberation scheduling

The principal reason we are interested in search performance is because the search is embedded in a real-time negotiation scenario, and time must be allocated to it before bids are received, and therefore before the exact dimensions of the problem are known. In [Greenwald and Dean, 1995], deliberation scheduling is done with the aid of anytime and contract algorithms, and performance profiles. An anytime algorithm is one that produces a continuously-improving result given additional time, and a contract algorithm is one that produces a result of a given quality level in a given amount of time,

but may not improve given additional time. The best winner-determination algorithms we know of for the MAGNET problem have marginal anytime characteristics, and we know of no applicable contract-type algorithms. In fact, [Sandholm, 2002] presents an inapproximability result for the winner-determination problem, leading us to believe that there may not be an acceptable contract algorithm.

One way to think about deliberation scheduling is to assign the time required for deliberation a cost, and then to balance the cost of deliberation against the expected benefit to be gained by the results of the deliberation. This is the approach taken in [Boddy and Dean, 1994]. However, much of this analysis assumes that there is a "default" action or state that can be used or attained without spending the deliberation effort, and that there is a clear relationship between the time spent in deliberation and the quantifiable quality of the result. In the MAGNET case, the alternative to deliberation is to do nothing.

In [Ronén *et al.*, 1998], the problem addressed is how to deal with situations where the "offered load" would saturate the available computing resources in a real-time environment. Computing tasks arrive at random, have predictable runtimes, and produce some specific value for the agent if they are completed by their respective deadlines. The idea is to adapt to the load as it arrives, and decide what to handle and which pieces of work to defer, rather than trying to predict or control the load ahead of time. They examine several polynomial approximations to maximizing the "value density" of the work that is scheduled, where value density is the ratio of the value of the result to the time required to produce it.

Chapter 3

The MAGNET system

The name of our system, "MAGNET", is an acronym for Multi-AGent NEgotiation Testbed. MAGNET is a software system intended to support experimental work in multi-agent negotiation. It is not, however, a general environment for exploring arbitrary types of negotiation among arbitrary types of agents. We envision a particular type of auction-based negotiation scenario, in which agents are self-interested, economically motivated, and heterogeneous. Furthermore, we have focused on negotiation scenarios in which the object of the interaction is to gain agreement on the performance of a set of coordinated tasks that one of the agents desires to complete in order to maximize its own utility. We assume that agents will cooperate in such a scheme to the extent that they believe it will be profitable for them to do so.

The descriptions in this chapter are largely qualitative, and are intended to serve as a reference model and context for the more detailed analyses in the later chapters.

3.1 Agents and their environment

Agents may fulfill one or both of two roles with respect to the overall MAGNET architecture, as shown in Figure 3.1. A *Customer agent* pursues its goals by formulating and presenting *Requests for Quotations* (RFQs) to *Supplier agents* through a market infrastructure [Collins *et al.*, 1998]. An RFQ specifies a task network that includes task descriptions, a precedence network, and temporal constraints that limit task start and completion times. Customer agents attempt to satisfy their goals for the greatest expected profit, and so will in general accept bids at the least net cost, where cost factors can include not only bid prices, but also goal completion time, risk factors, and possibly other factors, such as preferences for strategic suppliers. More precisely, these agents are attempting to maximize the utility function of some user, as discussed in detail in [Collins *et al.*, 2000a].

A Supplier agent attempts to maximize the value of the resources under its control by submitting Bids in response to an RFQ, specifying what tasks it is able to undertake, when it is available to perform those tasks, how long they will take to complete, and a price for each bid, which may contain

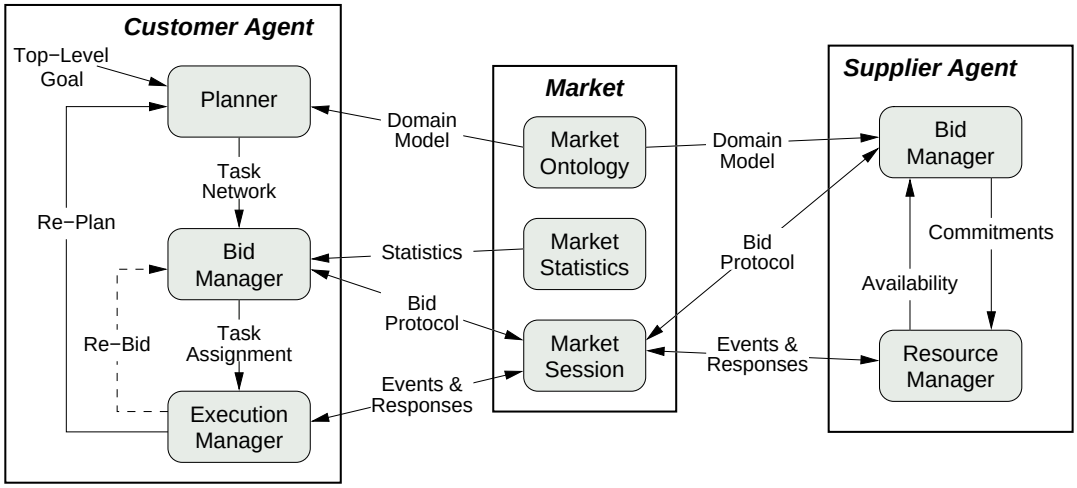

Figure 3.1: The MAGNET architecture

a bundle of one or more tasks. Suppliers may submit multiple bids to specify different combinations of tasks, or possibly different time constraints for different prices. For example, a supplier might specify a short duration for some task that requires use of high cost overtime labor, as well as a longer duration at a lower cost using straight-time labor. Currently, MAGNET supports simple disjunction semantics for bids from the same supplier. This means that if a supplier submits multiple bids, any non-conflicting subset can be accepted. Other bid semantics are possible [Nisan, 2000b, Boutilier and Hoos, 2001].

The MAGNET market infrastructure acts as an intermediary and information repository in support of participating agents. It helps Customers locate Suppliers that have the capabilities required to carry out a plan, and it keeps records of agent interactions. It serves an *ontology* of task types, along with market statistics on prices, durations, lead time, variability, and supplier availability. It also collects and serves statistics on supplier reliability, which customer agents can use to estimate risk during the bid-evaluation process.

3.2 Market Architecture

We now visit the market infrastructure in more detail. Although Multi-Agent negotiations can be carried out without an intermediary [Rosenschein and Zlotkin, 1994, Sandholm, 1996, Reeves *et al.*, 2001], the existence of the market greatly simplifies the decision processing tasks of the individual agents. In particular, the statistics-gathering function of the market allows agents to reason about temporal and risk factors using a body of data that is potentially much richer than any individual agent could hope to accumulate. Before describing the architecture of the MAGNET market infrastructure, we address the requirements we expect this infrastructure to satisfy.

3.2.1 Requirements

The fundamental requirement is that the architecture should provide a framework for secure and reliable commerce among self-interested agents. Existing architectures and proposals provide such a framework [Tennenbaum *et al.*, 1997, McConnell *et al.*, 1997], but they fail to address the information needs of MAGNET agents. We focus on several areas of requirements that have not been fully addressed in existing proposals, and that will be required to support the contract negotiations needed to formulate and carry out plans in a virtual organization. They are presented in approximate priority order, in terms of their importance to MAGNET agent design.

3.2.1.1 Market Statistics

As we shall see, many agent decision processes depend on market statistics of various types. At a minimum, a Customer agent must rely on market statistics to know what resources are likely to be available, with what lead times, and possibly at what costs. Suppliers would like to know how much competition they are likely to have in order to gauge the value of an opportunity.

At a minimum, the following statistical information is required:

- For each task type, the number of suppliers registered, and level of activity (number of requests and number of bids per request). Customers need to know whether bids are likely.
- For each task type, cost and lead time, with variability. The representation of lead time information and its relationship to cost remains under study. Customers need to know approximate costs for planning purposes, and they need to know whether their lead-time needs are realistic.
- For each task type, duration and performance to commitment across the market, and variability. Some tasks are intrinsically more variable than others, due to dependence on weather for example.
- For each task type, relationship between time-window size and bids received. Customers need to know how tightly they can schedule.
- For each supplier, performance to commitment.

In a real-world environment, we expect that in general the more recent data is more indicative than older data, so the statistics need to give more weight to the more recent data. In some markets, seasonal effects may be important. In addition to central tendencies and variability, distribution data may be important, when distributions are likely to be distinctly non-normal.

3.2.1.2 Market Ontology

Parties to an agreement need to first agree on the terms in which the agreement is expressed. In the MAGNET domain these terms include at a minimum the products and services that are to be

bought and sold in the marketplace, as well as time, duration, and cost. We expect the market itself to provide standard definitions for these terms. Others [Fox, 1996, Gruninger and Fox, 1994, Schlenoff *et al.*, 1998, Singh, 1999] have developed detailed ontologies and languages for expressing ontologies. For experimental purposes, much of this can be abstracted, although many features of business and manufacturing ontologies will require extensions to the current MAGNET models. These include notions of quantity, flow, location, calendars, and partial completion.

3.2.1.3 Common time reference

Many negotiation protocols involve time limits, such as a deadline for receipt of bids. All parties to a time-sensitive negotiation process must have a common time reference. In a research environment, this is more complex than simply using ntp[1] to synchronize the machines on which MAGNET components run. This is because many experimental scenarios require the same simulated time period to be replayed repeatedly, possibly at rates that differ from the standard clock rate. One way to accomplish this is to initialize the market with 3 parameters, known as "base" b, "offset" z, and "rate" r. The simulated time t_s can then be related to the "true" time t as maintained by ntp by

$$t_s = b + z + r(t - b).$$

3.2.1.4 Matchmaking

Customers need to find potential suppliers, and suppliers need to find potential customers [Sycara *et al.*, 1997]. Ideally, each supplier in a market will be able to maintain an up-to-date "registration" in the market that allows them to receive customer requests that have a good probability of leading to profitable interactions. This is a much larger issue in an automated marketplace than in "traditional" market environments, because of expectations for high utilization, short decision cycles, and low transaction costs. Suppliers want to see requests that have value to them. Customers want to be exposed to suppliers who might not otherwise be visible. Ideally this will minimize unproductive communication and evaluation of alternatives.

Matchmaking requires that suppliers register their capabilities in terms that are defined in the Ontology, and that customers communicate their needs in those same terms. The expectation is that the market will interpret the customer's RFQ to extract this information, and then advertise the RFQ to those suppliers whose registration information intersects the requirements in the RFQ.

3.2.1.5 Protection against fraud and misrepresentation

We must assume that participating agents will avail themselves of any opportunities that exist in the design of the market to gain advantage at the expense of other agents. The structure of the market must recognize and protect against situations that allow agents to gain unfair advantage. Strategies that can result in unfair gains include the following. Each of these issues must be addressed by the

[1] See IETF RFC 1305 (`http://www.ietf.org/rfc/rfc1305.txt`).

architecture and design of the market.

- Hiding one's identity or taking on the identity of another. The current industry standard for dealing this this issue is the use of digital certificates, usually in conjunction with a Public Key Infrastructure (PKI). The certificates must be traceable to a Certification Authority that can certify that the owner of the certificate has been verified according to some specified criteria, such as physical address, registered corporate name, etc.
- Dishonest auctioneer - In Vickrey-type auctions [Vickrey, 1961, Varian, 1995], the motivation for truth-telling on the part of participants is predicated on their belief in the honesty of the auctioneer.
- Miscommunication of the rules under which an auction is being conducted, or subversion of those rules. The market must implement and enforce the advertised rules.
- Failure to follow through on commitments. This requires that the market have visibility of commitments, and of notifications that commitments have been satisfied or not. We presume that the market cannot compel agents to follow through on commitments, but it can verify that commitments have been satisfied, at least in terms of the message exchanges (completion notifications, payment notifications) that are visible to the market.

The architecture must also have the ability to validate non-performance and assess any negotiated decommitment penalties.

3.2.1.6 Discouragement of counterspeculation

Opportunities for counterspeculation arise when the rules of negotiation allow agents to gain advantage by making use of factors other than their own capabilities and valuations, such as their estimates of the capabilities and valuations of the customers or other suppliers [Kreps, 1990]. We are concerned with two general types of counterspeculation. Value-based counterspeculation [Rosenschein and Zlotkin, 1994, Sandholm, 1996, Vickrey, 1961, Varian, 1995] occurs when agents use their own estimates of each other's valuations to set bid prices. In [Collins *et al.*, 1997], we identify two classes of time-based counterspeculation opportunities in a contracting domain that can be controlled by the settings of timing parameters in the RFQ. One of these situations occurs when supplier agents are allowed to expire their bids before the customer's call-for-bids expires, and the other situation occurs when customers are perceived by suppliers to be considering bids and formulating plans before bidding is closed.

In general, there is little the market can do to prevent value-based counterspeculation, other than to provide protocols that reduce the value of such strategic behavior [Vickrey, 1961]. Time-based counterspeculation can be limited by enforcing rules represented by the RFQ timing parameters. For example, a sealed-bid auction protocol requires that the market not return bids to the customer until after the bidding deadline has passed.

3.2.1.7 Context support for Complex Multi-agent Negotiation

Protocols that require agents to do complex marginal cost computations [Sandholm, 1996, Smith, 1980, Parkes and Ungar, 2001], or plan generation and composition [Collins *et al.*, 1997, Rosenschein and Zlotkin, 1994], may require extended periods of time to complete, during which a context must be maintained. If we extend the transaction to cover not only the initial negotiation but also the execution of tasks, then the time during which the negotiated transaction extends can span significant periods of time, at least in the range of weeks to months. In order to support this, the market infrastructure must maintain the state of each transaction over time. This is a prerequisite for many other functions we expect the market to perform, and it makes the system more robust in the face of hardware and communication failures.

For a given RFQ, the state information includes the RFQ and its state (bidding open, bidding complete, awards issued, work complete), the individual bids with state (received, forwarded to customer, accepted, not accepted), and the bid awards with state (work started, work complete, late start, late completion, abandoned, failed, paid).

3.2.2 Architectural Elements

We now turn to the abstract design of a market infrastructure that will have the capacity to satisfy these requirements. We don't address standard architectural attributes like scalability, security, manageability, standards compliance, etc. except in passing. These can largely be addressed by high-level design and technology choices and are beyond the scope of this discussion (but see [Josephs, 2001], where some of these issues are addressed). The fundamental elements of this architecture form a hierarchy consisting of the *exchange*, *markets*, and *market sessions*. We will start with the "innermost" element, the market session.

3.2.2.1 Market Session

A *market session* (or simply a session) is the vehicle through which market services are delivered dynamically to participating agents. It serves as an encapsulation for a transaction in the market, as well as a persistent repository for the current state of the transaction, as shown in Figure 3.2.

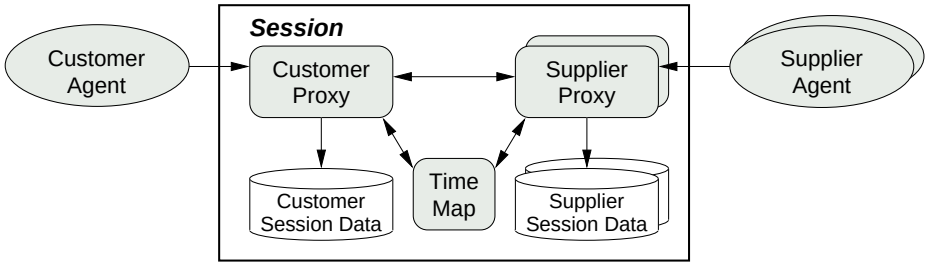

Figure 3.2: A Market Session keeps track of a single transaction for one Customer and a set of Suppliers.

We have chosen the term "session" to emphasize the temporally extended nature of many of these interactions. For example, in a construction market, if an agent wishes to build a new garage, it initiates a session and issues a Request for Quotations containing all or part of the plan for building the garage. The session extends from the initial RFQ through the bid submission, awards, possibly additional RFQs, construction work, interactions over failures or schedule adjustments, paying of bills, and final closing. In other words, the session encloses the full life of a contract or a set of related contracts within a given market. The session mechanism ensures continuity of partially-completed transactions even if agents themselves must be restarted. It also protects against fraud, supports non-repudiation, limits time-based counterspeculation, and relieves the participating agents from having to keep track of detailed negotiation status themselves.

The session uses a *time map* [Boddy, 1993] to keep track of events that are supposed to happen at particular times. Initially, it is derived by combining plan and timeline data from the RFQ with protocol specifications from the market that supports the session. When bids are awarded, the time windows and duration data from the bid awards is added. The time map can be used to trigger events at particular times, or to detect that scheduled events have not occurred at their correct times. For example, in a sealed-bid auction protocol, bids may be withheld from the customer until the bidding period has ended[2]. The time map would generate the event to close the bidding and send the accumulated bids to the customer. Another example of time map function is the detection of late delivery of a committed service. Because the session records such events independently of the agents, disputes among the agents themselves can be mitigated.

The data collected by the session is strictly partitioned to permit market rules to govern its visibility. For example, bids would typically be visible to the Customer and to the supplier who submitted them. Some markets, however, might make bids visible to all suppliers, or might make winning bids visible after bids are awarded. This data is also the foundation of the market statistics function discussed in the next section.

When an RFQ is received, the session must examine it to extract task-type data. It then requests a list of potential suppliers from its market, and advertises the new RFQ to suppliers in the list. Alternatively, the customer may specify a list of pre-approved suppliers for some or all tasks.

At any given time, a session can be *open* to new supplier participants, or *closed*. A session containing one or more RFQs that are currently accepting bids would typically be open to new participants, but would be closed once bids were awarded, unless the customer has specified limitations on the supplier list.

3.2.2.2 Market

A *market* is a forum for commerce in a particular commodity or business area. There could be markets devoted to banking, publishing and printing, construction, transportation, industrial equipment,

[2]This can prevent the type of temporal counterspeculation in which bidders attempt to take advantage of the limited computing capabilities of the customer by manipulating the timing of their bidding. See [Collins *et al.*, 1997].

etc. Each market includes a set of domain-specific services and facilities, as shown in Figure 3.3. The primary functions of the market are to act as a matchmaker between customers and suppliers, to define terms of discourse among participating agents, and to collect and publish statistics to support agent decision processes. In addition, each market encapsulates a set of Sessions that represent the active transactions within the Market.

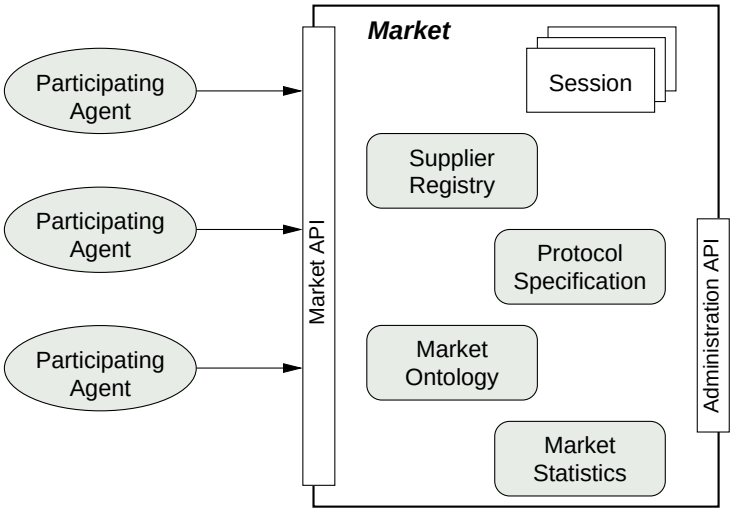

Figure 3.3: A Market supports commerce in a particular business area, and encapsulates a set of Sessions representing the active transactions in the Market.

Important elements of the market include:

Ontology – Each market provides to its customers and suppliers an Ontology, specific to the domain of the market, specifying the terms of discourse within that domain. In a commodity-oriented domain, it would include terms for the products or services within the domain, as well as terminology for quality, quantity, features, terms and conditions of business, etc. See, for example, [Ontology.org, 2002]. In a planning-oriented domain, the Ontology would include specifications of services in a form that supports planning, such as [Tate, 1996].

Protocol Specification – an operational specification defining the types of negotiation supported within the market. This specifications would include limits on parameters of the negotiation protocol, such as the size of deposit required when bids are awarded, whether bids can be awarded before the bid deadline, etc. For example, in a market where suppliers must frequently deal with subcontractors to prepare bids, there might be a lower limit on the time suppliers must be allowed to prepare their bids. An example of a market environment that uses such an operational specification is the Michigan Internet AuctionBot [Wurman *et al.*, 1998].

Supplier Registry – This is simply a list of supplier agents who have expressed interest in doing business in the market. Entries in this registry would include the identity of a supplier, a catalog [Keller, 1996] (or a method for accessing a catalog) of that supplier's interests, products or capabilities, which can be used to locate suppliers to submit bids for new RFQs, and the

location (a URI or similar specification) of a supplier agent that is empowered to negotiate contracts on behalf of the supplier. Supplier catalogs are required to express their interests and offerings in terms of the market's ontology. As described earlier in Section 3.2.2.1, a primary function of the supplier registry is to return a list of potential suppliers, given a list of task types extracted from an RFQ.

Note that here we distinguish between a Supplier Agent and its Principal, a Supplier. A given organization may have multiple supplier agents active in multiple markets, seeking out new business and managing ongoing interactions, but the Customer is primarily interested in the identity of the organization on whose behalf the agent is acting.

Market Statistics – As we shall see in Chapter 4, many of the agent's decision processes depend on statistical information from the Market. For example, when a customer is building its initial schedule, it needs to know whether there are active suppliers in the market for the tasks it wants performed, likely lead times for the resources it will need, average durations of tasks, etc., as described earlier in Section 3.2.1.1. Most of the data used to generate these statistics are expected to come from the Market's Sessions. Some statistics, such as supplier reliability records, would likely be shared among multiple markets. That is the function of the Better Business Bureau in the Exchange.

In a typical negotiation scenario, the Customer would first come to the market to access its ontology and statistics. With the ontology, it can specify its plans in terms that other agents in the market can understand. With the statistics it can add temporal detail that maximizes the probability of profitable completion of its plan. The customer would then acquire a session from the market, and issue one or more RFQs to request services. The session, in turn, would interpret the RFQ, request a supplier list from the market, and notify suppliers who are registered for the services requested in the new RFQ. Suppliers could then join the new Session, retrieve the RFQ, and submit bids. Because the market maintains a list of open sessions which may be accessed by supplier agents, suppliers who have not registered for the particular services specified in a specific RFQ would typically be able to see and retrieve the RFQ by inspecting this list.

3.2.2.3 Exchange

An *exchange* provides a set of common services for a collection of domain-specific markets in which goods and services are traded. Services include verifying identities of participants in a transaction, ability to search for markets that support trade in specific goods or services, and a Better Business Bureau that can provide information about the reliability of other agents based on past performance. Architecturally, an exchange is a network-accessible resource that supports a set of markets and common services, as depicted in Figure 3.4.

In a typical negotiation scenario, both Customers and Suppliers would use the exchange to establish their identities and to acquire the necessary security certificates to access the markets. Once they have established their credentials, Suppliers may query the Exchange for Markets in which to register

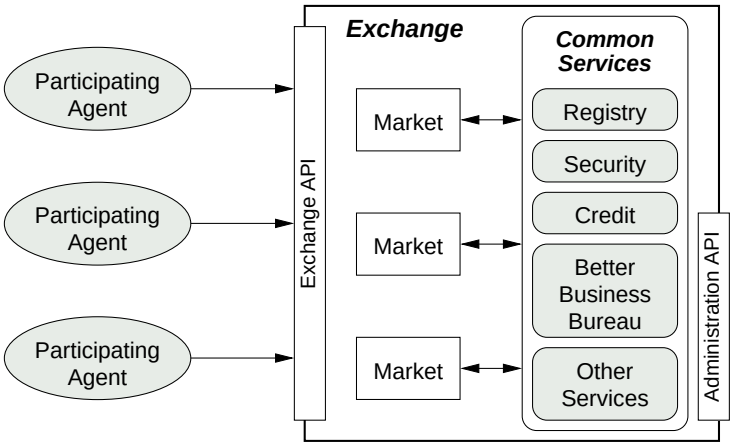

Figure 3.4: An Exchange aggregates multiple markets and provides common services.

their capabilities. Customers, in turn, would query the Exchange to find Markets that can provide the capabilities and resources needed to carry out their plans.

Although conceptually the Exchange encapsulates its markets, there are at least three ways the relationship between Exchange and Market can be implemented. The simplest is that there is only a single Market, and the functions of the Exchange and the Market are merged. Second is a literal encapsulation as shown in Figure 3.4. The third, and most general and scalable approach is to make the exchange a resource for multiple distributed markets. Prototypes of both the second and third alternatives have been implemented in experimental MAGNET market servers.

3.3 Magnet Customer Agents

We now focus on the structure and responsibilities of a customer agent in the MAGNET environment. We will put off a detailed examination of the decision processes an agent must implement until Chapter 4. As indicated in Figure 3.1, the basic operations are planning, bidding, and plan execution. Each of these functions can be implemented as fully autonomous functions, or as user interfaces that allow a user to perform them with varying degrees of assistance. For many organizational environments, we expect that a mixed-initiative approach [Collins *et al.*, 2000a] will be most acceptable, in which the agent handles some tasks autonomously and requests final decisions from its principal for others, providing analysis tools to support those decisions. We will focus on these decision in Chapter 4.

A more detailed schematic description of the customer agent is shown in Figure 3.5. Virtually all agent behaviors can be defined by which events they respond to, how they update the agent status hierarchy, and what communication and user interface actions they take.

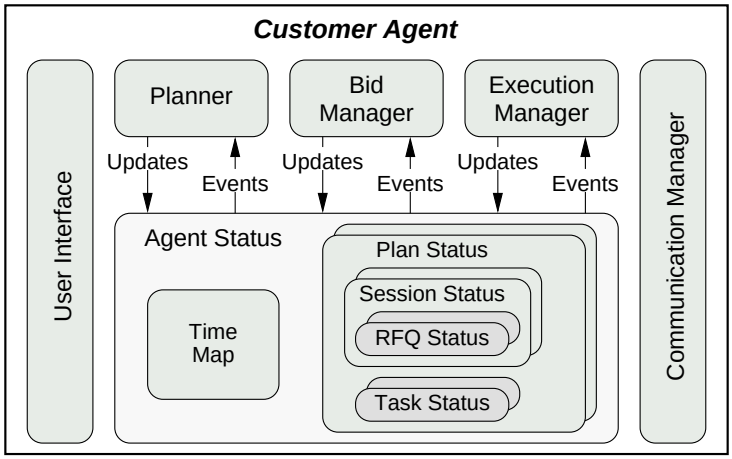

Figure 3.5: Schematic diagram of a MAGNET customer agent.

3.3.1 Status Hierarchy

This is the repository of agent state. Each change in state generates an event, and the time map allows events to be generated at specific times in the future, or at specified offsets from other events. Every active plan is represented by a Plan Status instance, which persists until the plan is completed or abandoned. Once a plan exists, the Agent can go into one or more markets and start negotiation sessions, which are represented by Session Status instances. Each RFQ in a session is represented by an RFQ Status instance, and each task by a Task Status instance.

This structure makes it easy to specify and implement behaviors that are time-limited, time-driven, or event-driven. For example, when an RFQ is issued, the significant points in its timeline are entered into the Time Map. Events generated by the Time Map can then cause the agent to retrieve bids from the market when the bid deadline arrives, or to terminate winner-determination search before bids must be awarded. It is also possible, without additional programming effort, to have multiple plans in progress, with negotiations proceeding in multiple markets simultaneously.

3.3.2 Planner

The Planner's task is to turn high-level goals into executable plans, represented as task networks. More formally, a task network $\mathcal{P} = (\mathcal{S}, \mathcal{V})$ contains a set $\mathcal{S}$ of task descriptions, and a set $\mathcal{V}$ of precedence constraints. For each task $s \in \mathcal{S}$, there is a possibly empty set $\mathcal{V}_s \subset \mathcal{V}$ of precedence constraints $\mathcal{V}_s = \{s' \prec s\}$, associating s with each of the tasks s' whose completion must precede the start of task s. Other types of precedence constraints are certainly possible, such as start-start and finish-finish constraints. Notice that the task network itself need not be situated in time. That will happen when we compose the RFQ. Also notice that the operations in the task network do not need to be linearized with respect to time, since operations can be executed in parallel by multiple agents.

The task network is a central data element in a customer agent. When a new task network is

generated by the Planner, it is embedded in a new Plan Status instance. The Bid Manager then uses it to generate RFQs and to evaluate and record resource commitments and timing data, and the Execution Manager uses it to monitor and repair the ongoing execution of the plan. Part or all of the task network is included in a RFQ, and the market session uses it to find potential suppliers and to keep track of the progress of the transaction. Once bids are awarded, both the customer and the session add each task to their respective time maps, along with committed start and completion times, to support tracking of completion status and performance to commitment.

Each of the tasks in a plan $\mathcal{P}$ is described by a "task type" and a set of parameters that are specific to the particular task type. It is a requirement of the design that all tasks that are included in an RFQ are of a type that is included in the Ontology of the market to which the RFQ is submitted. This does not mean that every task needs to be included in some market's ontology, just that the tasks that are to be put out for bid be so described. Other tasks may be handled by the customer somehow, or may be used to represent time delays or milestones.

3.3.3 Bid Manager

The Bid Manager is responsible for ensuring that resources are assigned to each of the tasks of a task network, that the assignments taken together form a feasible schedule, and that the expected profit is maximized. This normally requires that the cost and risk of executing the plan is minimized.

When the Bid Manager is invoked, some tasks in the task network may already be assigned. This can occur because the Execution Manager may request the Bid Manager to repair a partially-completed task network in which previously determined assignments have failed, because the agent will perform some of the tasks itself, or because bidding is being carried out in multiple markets or in multiple stages. For example, I may decide to wire my new garage myself, and so I might not contract for that task when seeking bids on the construction of my new garage.

The Bid Manager must plan the bidding process, construct and issue one or more RFQs, evaluate bids, and accept bids in order to carry out its responsibilities. The high-level structure of the Bid Manager is shown in Figure 3.6.

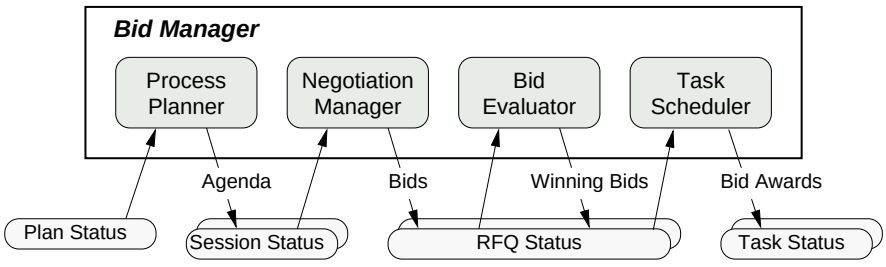

Figure 3.6: Bid Manager: high-level schematic

The Process Planner creates the high-level agenda for the Bid Manager. A primary responsibility is to allocate time to negotiation and plan execution. It decides which markets to use, when to

consult local catalog and timetable databases, and how to break up the task network accordingly. If the task network has alternative branches, it may also decide which alternatives to pursue and in what order. For example, it may decide to solicit bids on a high-value but risky approach, and if that fails to fall back on a lower-value but safer approach. It could even decide to defer taking bids on later tasks until earlier tasks were under way or even completed. For each market, it must initiate a session and create a Session Status instance, which will be used to track the progress of the negotiation in that market.

The Negotiation Manager (Figure 3.7) handles the actual bidding process. Its job is to acquire "good-quality" bids to cover the tasks in the plan. Once bids are available, the Bid Evaluator runs a winner-determination process to choose bids (we will describe this process in much more detail in Chapter 5), and the Task Scheduler updates the time windows in the accepted bids to prepare Bid Award messages. This is necessary because suppliers' bids typically specify resource-availability intervals that are larger than the task durations, but the customer must specify, in the Bid Award messages, a specific start time for each task. This is done to minimize the resource-reservation costs for suppliers.

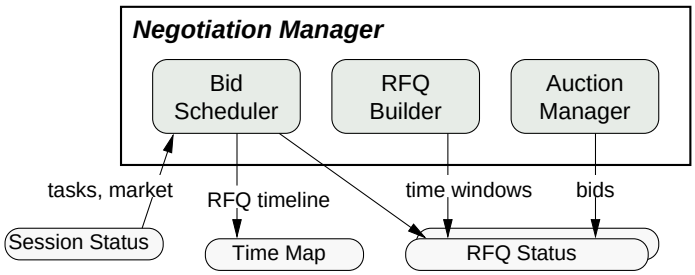

Figure 3.7: Negotiation Manager: high-level schematic

Before bids can be solicited in a market, the Bid Scheduler generates the bidding timeline, and the RFQ Builder constructs an RFQ. The RFQ is a structure that contains some portion of the task network data (tasks and precedence relations), along with a set of scheduling constraints. Information comes from several sources:

- From the Plan Status and Task Status instances, we have a set of tasks and their precedence constraints, and we know which tasks have already been scheduled or completed.
- From the Market, we have statistical information about duration and variability for the different task types. We also have information about resource availability and the number of vendors who are likely to bid on tasks of this type.
- From the Process Planner, we have the overall schedule for the execution of the task network, in particular the earliest time any task can start, and a completion deadline.

The Negotiation Manager attempts to produce an RFQ that will solicit the most advantageous set of bids possible. The bid evaluator cannot evaluate bids that are not received, nor can it make

successful combinations of bids that conflict with one another over precedence constraints. The principal job of the RFQ builder is to specify time windows, consisting of early-start and late-finish times, in the RFQ for each task. It must find a balance between giving maximum flexibility to suppliers, ensuring that the resulting bids will combine feasibly, and ensuring that the job will be completed by the deadline. This may involve specifying time windows that, taken together, define an infeasible schedule (the early start of a succeeding task may be earlier than the late finish of a preceding task). We will consider this issue in more detail in the next chapter.

3.3.4 Execution Manager

The Execution Manager is responsible for overseeing execution of the plan as contracted, and making decisions on how to respond when events do not proceed as expected. It receives the task assignments from the Bid Manager, and receives updates on plan execution from contracted suppliers. It maintains the time map with tasks, vendor commitments, and temporal constraints among tasks.

As time passes and the execution of the plan proceeds, the Execution Manager works in conjunction with the market session to drive the plan to completion. In the process, the Execution Manager receives notifications of task release and task completion events. It is then responsible for making decisions and taking appropriate action in response to those notifications.

In the next chapter we consider the events that the Execution Manager must respond to, but little further detail is available, since this component has not been implemented or even explored in detail.

3.3.5 Communication Manager

The function of the Communication Manager is easy to understand. It forms the connection between the agent and the market infrastructure, passing requests and notifications from the agent to the market, and receiving responses and notifications and turning them into events that the agent can process. To make the agent useful in a variety of test environments, multiple implementations must be available. For example, the search experiments in Chapter 6 were run with a version of the Communication Manager that actually encapsulates a module that behaves as a whole community of virtual suppliers. There is no market, and the only messages that are processed are RFQs and Bids. Another version, used for examining market functionality, sends and receives SOAP/http messages.

Chapter 4

Decision processes in a MAGNET customer agent

We now focus on the decision processes that must be implemented by a MAGNET customer agent. The ultimate goal of this work is that our agents exhibit rational economic behavior. In other words, the agent would always act to maximize the *expected utility* of its principal. Much work remains to be done before we have a clear understanding of how to maximize expected utility at all decision points. In this chapter, we simply walk through a prototypical transaction from the customer agent's viewpoint and observe the various decisions that such an agent must make. In Chapter 5 we will focus on the problem of evaluating bids in much more rigorous detail.

We will use an example to work through the agent's decisions. Imagine that you own a small vineyard, and that you need to get last autumn's batch of wine bottled and shipped[1]. During the peak bottling season, there is often a shortage of supplies and equipment, and your small operation must lease the equipment and bring on seasonal labor to complete the process. If the wine is to be sold immediately, then labels and cases must also be procured, and shipping resources must be booked. Experience shows that during the Christmas season, wine cases are often in short supply and shipping resources are overbooked.

4.1 Expected Utility

In the real world, the outcomes of our decisions always have an element of uncertainty. If our automated agent is to produce plans that are acceptable to a human decision-maker, the agent must have the ability to handle decision-making in an uncertain environment. To model this concept, we use Expected Utility Theory (EUT) [Biswas, 1997, Rabin, 2000, Cox and Sadiraj, 2001], which attempts to describe human economic decision-making. EUT models decision-making under uncertainty by using probabilities and a construct known as a utility curve, $u(x)$, a function that maps the value

[1] This example is taken from the operations of the Weingut W. Ketter winery, Kröv, Germany.

x of some (uncertain) payoff to a level of utility. According to EUT, a decision-maker who is faced with a "gamble" G consisting of a set of wealth-based outcomes will calculate the expected utility over the gamble as:

$$Eu(G) = \sum_{\{x_i, p_i\} \in G} p_i u(x_i)$$

where p_i is the probability of outcome i, and x_i is the resulting payoff of the decision-maker if outcome i is realized. Stated another way, the decision-maker weighs the utility of each outcome within the opportunity. This is an *income-based* utility model [Cox and Sadiraj, 2001], and we use it for convenience even though it lacks generality[2].

If we assume a constant *risk-aversion coefficient* [Mas-Colell *et al.*, 1995] $r = -u''/u'$, then the utility function u can be expressed as

$$u(x) = \begin{cases} -e^{-rx} & \text{for } r \neq 0 \\ x & \text{for } r = 0 \end{cases}$$

To make a decision, the decision-maker evaluates this expected utility, and accepts the gamble G just in case $Eu(G)$ is positive. Similarly, a decision-maker faced with multiple opportunities can decide which (if any) it will pursue by comparing their expected utilities. We will bypass the issue of modeling a user's risk preferences, but see [Jullien and Salanié, 2000] for an example.

4.2 Planning

A transaction (or possibly a series of transactions) starts when the agent or its principal acquires a goal that must be satisfied, or an opportunity arises that, if satisfied, would likely yield a positive payoff. Attributes of the goal might include a payoff and a deadline, or a payoff function that varies over time, either according to a discount rate or some other function.

While it would certainly be possible to integrate a general-purpose planning capability into a MAGNET agent, we expect that in many realistic situations the principal will already have a plan, perhaps based on standard industry practices. Figure 4.1 shows such a plan, for our winery bottling operation. We shall use this plan to illustrate the decision processes the agent must perform (or provide assistance to its principal in performing).

Every task in the plan must meet one of two criteria. Either it must be an instance of some product or service in the ontology of some market, or it must be handled outside of any market. For a task derived from a market ontology, the specifications of the task must be in accordance with that Ontology; otherwise the agent or its principal will be responsible for scheduling the task without the assistance of a market. For our example, assume that the tasks in our plan are associated with markets as specified in Table 4.1.

It appears that we will need to deal with 3 different markets, and we will pack the cases ourselves.

[2]The income-based utility model does not scale well over large ranges in the values of risks and payoffs.

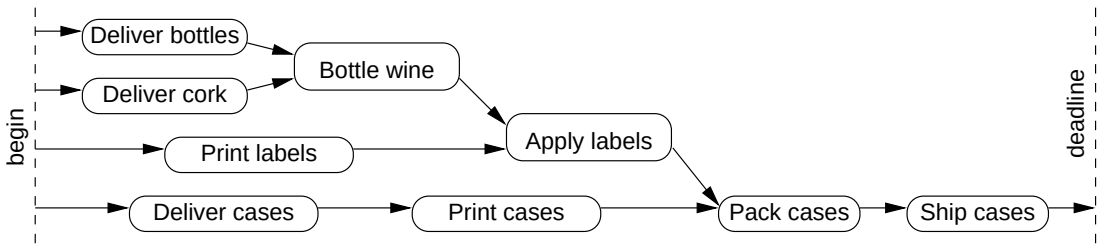

Figure 4.1: Plan for the wine-bottling example.

Table 4.1: Tasks and market associations for the wine-bottling example

Task	Description	Market
s_1	Deliver bottles	Vineyard Services
s_2	Deliver cork	Vineyard Services
s_3	Bottle wine	Vineyard Services
s_4	Print labels	Printing & Graphic Arts
s_5	Apply labels	Vineyard Services
s_6	Deliver cases	Vineyard Services
s_7	Print cases	Vineyard Services
s_8	Pack cases	(none)
s_9	Ship cases	Transport Services

Or perhaps we'll open a few bottles and invite the village to help out.

So far, our plan is not situated in time, and we have not discussed our expected payoff for completing this plan. In the wine business, the quality of the product depends strongly on time. The wine must be removed from the casks within a 2-week window, and the bottling must be done immediately. For some varieties, the price we can get for our wine is higher if we can ship earlier, given a certain quality level. All the small vineyards in the region are on roughly the same schedule, so competition for resources during the prime bottling period can be intense. Without specifying the exact functions, we assume that the payoff drops off dramatically if we miss the 2-week bottling window, and less dramatically as the shipment date recedes into the future.

This example is admittedly a bid contrived, and it is important not to stretch it too far. We are treating the bottling and labeling operations as atomic – the entire bottling operation must be finished before we can start labeling – even though common-sense would inform us that you would probably want to apply this constraint at the per-bottle level, not the per-batch level. On the other hand, some varieties of wine are aged in the bottles for 6 months or more before the labels are applied.

Formally, we define a plan $\mathcal{P} = (\mathcal{S}, \mathcal{V})$ as a task network containing a set of tasks $\mathcal{S}$, and a set of precedence relations $\mathcal{V}$. A precedence relation relates two tasks $s, s' \in \mathcal{S}$ as $s \prec s'$, interpreted as "task s must be completed before task s' can start."

4.3 Planning the bidding process

At this point, the agent has a plan, it knows which markets it must deal in to complete the plan, the value of completing the plan, and how that value depends on time. The next step is to decide how best to use the markets to maximize its utility. It will do this in two phases. First, the agent generates an overall plan for the bidding process, which may involve multiple RFQs in each of multiple markets. We call this the "bid-process plan" (in Figure 3.6, it appears as "agenda"). Then a detailed timeline is generated for each RFQ.

The simplest bid-process plan, from a conceptual standpoint, would be to issue a single RFQ in each of the markets, with the parts of the plan that are relevant to those markets. If all RFQs are issued simultaneously, and if they are all on the same timeline, then we can combine their bids and solve the combined winner-determination problem in a single step. However, there are many reasons why this might not be the optimum strategy. For example:

- We may not have space available to store the cases if we are not ready to pack them when they arrive.
- Our labor costs might be much lower if we can label as we bottle; otherwise, we will need to move the bottles into storage as we bottle, then take them back out to label them.
- Once cases are packed, it is easy for us to store them for a short period. This means that we can allow some slack between the packing and shipping tasks.
- There is a limit to what we are willing to pay to bottle our wine, and there is a limit to the premium we are willing to pay to have the bottling completed earlier.

The agent can represent these issues as additional constraints on the plan, or in some cases as alternative plan components. For example, we could constrain the interval between s_5 (labeling) and s_8 (packing) to a maximum of one day, or we could add an additional storage task between s_3 (bottling) and s_5 that must be performed just in case there is a non-zero delay between the end of s_3 and the start of s_5.

There are many possible alternative actions that the agent can take to deal with these issues. It need not issue RFQs in all markets simultaneously. It need not include all tasks for a given market in a single RFQ. Indeed, dividing the plan into multiple RFQs can be an important way to reduce scheduling uncertainty. For example, we might want to have a firm completion date for the bottling and labeling steps before we order the cases.

Data from the market's statistics service can be used to support these decisions. For example, if we knew that resources were readily available for the steps up through the labeling process (tasks $s_1 \ldots s_5$), we could include the case delivery and printing steps (tasks s_6 and s_7) in the same RFQ. This could be advantageous if suppliers were more likely to bid or likely to bid lower prices if they could bid on more of the business in a single interaction. In other words, some suppliers might

be willing to offer a discount if we agree to purchase both bottles and cases from them, but if we negotiate these two steps in separate RFQs, we eliminate the ability to find out about such discounts.

We should note that there are also actions suppliers could take to help the customer mitigate issues like the constraint between bottling and packing. For example, if a supplier knew about this constraint, it could offer both tasks at appropriate times, or it could give the customer the needed scheduling flexibility by offering the case delivery over a broad time window or with multiple bids with a range of time windows. In some domains this could result in significantly higher costs, due to the large speculative resource reservations the supplier would have to commit to in order to support its bids.

The issue of cost limits is especially important in cases where bidding will be done in multiple phases. For example, suppose we have a limit of $15,000 to complete the whole bottling plan. What limit should we place on the bids for tasks $s_1 \ldots s_5$? An obvious approach would be to sum the expected costs for the remainder of the plan, and the known costs for portions already committed, and subtract it from our limit.

The bid-process plan that results from this decision process is a network of negotiation tasks and decision points. Figure 4.2 shows a possible bid-process plan for our wine-bottling example.

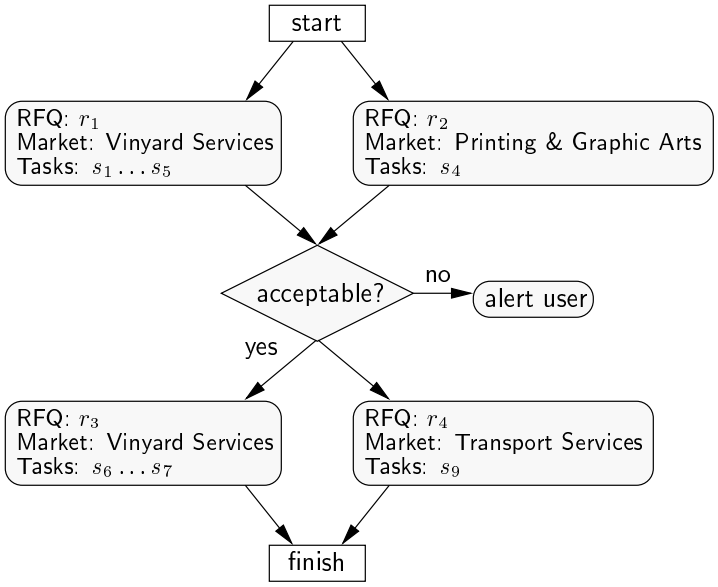

Figure 4.2: Bid-process plan for the wine-bottling example.

Once we have a bid-process plan, we know what markets we will enter, and how we want to divide up the bidding process. We must then schedule the bid-process plan, and allocate time within each RFQ/bidding interaction. These two scheduling problems may need to be solved together if the bid-process plan contains multiple steps and it is important to finish it in minimum time. Each RFQ step needs to start at a particular time, or when a particular event occurs or some condition becomes true. For example, if the rules of the market require deposits to be paid when bids are

awarded, the customer may be motivated to move RFQ steps as late as possible, other factors being equal. On the other hand, if resources such as our bottling and labeling steps are expected to be in short supply, the agent may wish to gain commitments for them as early as possible in order to optimize its own schedule and payoff. We assume these decisions can be supported by information from the market, the agent's own experience, and/or the agent's principal.

Each RFQ must also be allocated enough time to cover the necessary deliberation processes on both the customer and supplier sides. Some of these processes may be automated, and some may involve user interaction. The timeline in Figure 4.3 shows an abstract view of the progress of a single negotiation. At the beginning of the process, the customer agent must allocate deliberation time to itself to compose its RFQ (which may be a difficult combinatorial problem – see for example [Babanov *et al.*, 2002]), to the supplier for bid preparation, and to itself again for the bid evaluation process. Two of these time points, the Bid deadline and the Bid Award deadline, must be communicated to suppliers as part of the RFQ. The Bid deadline is the latest time a supplier may submit a bid, and the Bid Award deadline is the earliest time a supplier may expire a bid. The interval between these two time points is available to the customer to determine the winners of the auction.

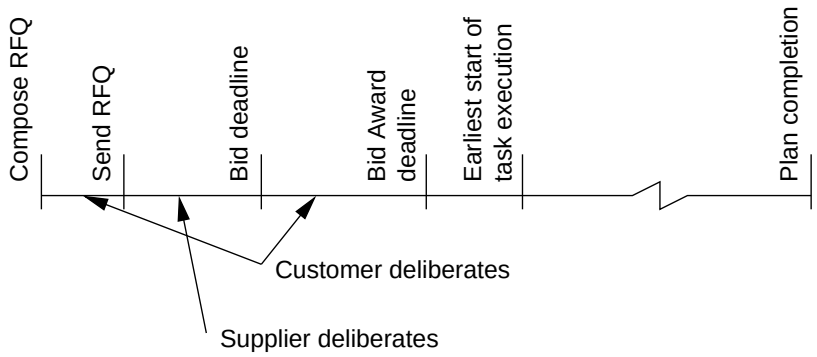

Figure 4.3: Typical timeline for a single RFQ

In general, it is expected that bid prices will be lower if suppliers have more time to prepare bids, and more time and schedule flexibility in the execution phase. Minimizing the delay between the bid deadline and the award deadline will also minimize the supplier's opportunity cost, and would therefore be expected to reduce bid prices. On the other hand, the customer's ability to find a good set of bids is dependent on the time allocated to bid evaluation, and if a user is making the final decision on bid awards, she may want to run multiple bid-evaluation cycles with some additional think time. We are interested in the performance of the winner determination process precisely because we realize that it takes place within a window of time that must be determined ahead of time, and because we expect better overall results, in terms of maximizing the Agent's utility, if we can maximize the amount of time available to suppliers while minimizing the time required

for customer deliberation. These time intervals can be overlapped to some extent, but doing so can create opportunities for strategic manipulation of the customer by the suppliers, as discussed in [Collins *et al.*, 1997].

The process for setting these time intervals could be handled as a non-linear optimization problem, although in general it seems that a more greedy approach would be adequate. This could consist of estimating the minimum time required for the customer's processes, and allocating the remainder of the available time to the suppliers, up to some reasonable limit.

4.4 Composing a Request for Quotes

At this point in the agent's decision process, we have the information needed to compose one or more RFQs, we know when to submit them, and we presumably know what to do if they fail (if we fail to receive a bid set that covers all the task in the RFQ, for example). The next step is to set the time windows for tasks in the individual RFQs, and submit them to their respective markets.

Formally, an RFQ $r = (\mathcal{S}_r, \mathcal{V}_r, \mathcal{W}_r, \tau)$ contains a subset $\mathcal{S}_r$ of the tasks in the task network $\mathcal{P}$, with their precedence relations $\mathcal{V}_r$, the task time windows $\mathcal{W}_r$ specifying constraints on when each task may be started and completed, and the RFQ timeline τ containing at least the bid deadline and bid award deadline. As pointed out earlier, there might be elements of the task network $\mathcal{P}$ that are not included in the RFQ. For each task $s \in \mathcal{S}_r$ in the RFQ the Bid Manager must specify a time window $w \in \mathcal{W}_r$, consisting of an earliest start time $t_{es}(s, r)$ and a latest finish time $t_{lf}(s, r)$, and a set of precedence relationships $\mathcal{V}_r = \{\text{for each } s, s' \in \mathcal{S}_r, s' \prec s\}$, associating s with each of the other tasks $s' \in \mathcal{S}_r$ whose completion must precede the start of s.

The principal outcome of the RFQ-generation process is a set of values for the early-start and late-finish times for the time windows $\mathcal{W}_r$ in the RFQ. We obtain a first approximation using the Critical Path (CPM) algorithm [Hillier and Lieberman, 1990], after making some assumptions about the durations of tasks, and about the earliest start time for tasks that have no predecessors in the RFQ (the *root tasks* $\mathcal{S}_R$) and the latest finish times for tasks that have no successors in the RFQ (the *leaf tasks* $\mathcal{S}_L$). As a first approximation, the customer can use market mean-duration statistics for the task durations. Overall start and finish times for the tasks in the RFQ may come from the bid-process plan, or we may already have commitments that constrain them as a result of other activities. For this discussion, we assume a continuous-time domain, although we realize that many real domains effectively work on a discrete-time basis. Indeed, it is very likely that some of our wine bottling activities would typically be quoted in whole-day increments. We also ignore calendar issues such as overtime/straight time, weekends, holidays, time zones, etc.

The Critical Path algorithm walks the directed graph of tasks and precedence constraints, forward from the early-start times of the root tasks to compute the earliest start $t_{es}(s)$ and finish $t_{ef}(s)$ times for each task $s \in \mathcal{S}_r$, and then backward from the late-finish times of the leaf tasks to compute the latest finish $t_{lf}(s)$ and start $t_{ls}(s)$ times for each task. The minimum duration of the entire task network specified by the RFQ, defined as $\max_{s' \in \mathcal{S}_L}(t_{ef}(s')) - \min_{s \in \mathcal{S}_R}(t_{es}(s))$, is called the

makespan of the task network. The smallest slack in any leaf task $min_{s \in \mathcal{S}_L}(t_{lf}(s) - t_{ef}(s))$ is called the *total slack* of the task network within the RFQ. All tasks s for which $t_{lf}(s) - t_{ef}(s) = \textit{total-slack}$ are called *critical* tasks. Paths in the graph through critical tasks are called *critical paths*.

Note that some situations will be more complex than this. This can happen when there are constraints that are not captured in the precedence network of the RFQ. For example, some non-leaf task may have successors that are already committed but are outside the RFQ. The CPM algorithm is still applicable, but the definition of critical tasks and critical paths becomes more complex.

Figure 4.4 shows the result of running the CPM algorithm on the tasks of RFQ r_1 from our bid-process plan. We are assuming task durations as given in the individual "task boxes." We observe several problems immediately. The most obvious is that it is likely that many bids returned in response to this RFQ would conflict with one another because they would fail to combine feasibly. For example, if I had a bid for the label printing task s_5 for days 5-7, then the only bids I could accept for the labeling task s_4 would be those that had a late start time at least as late as day 7. If the bids for s_4 were evenly distributed across the indicated time windows, and if all of them specified the same 4-day duration, then only 1/3 of those bids could be considered. In general, we want to allow time windows to overlap, but excessive overlap is almost certainly counterproductive. We will revisit this issue shortly.

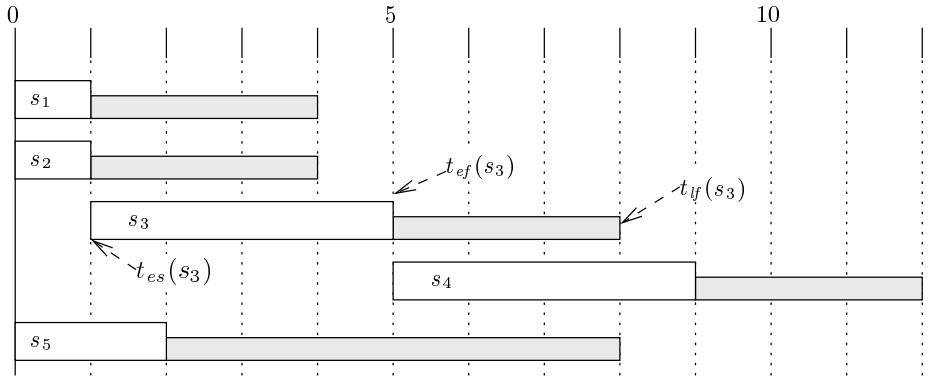

Figure 4.4: Initial time allocations for tasks in RFQ r_1. Only the $t_{es}(s)$ and $t_{lf}(s)$ times are actually specified in the RFQ.

Once we have initial estimates from the CPM algorithm, there are several issues to be resolved, as described in the following sections.

4.4.1 Estimating task durations

In order to run the CPM algorithm, we need to know the durations of the tasks. At this point in the process, the only information we have available regarding task durations is market statistics. We assume at minimum that this will give us mean and standard deviation data. The distribution may be unknown, but if the market is large enough it is likely to be well-approximated by a normal distribution. Therefore, as a first approximation, the use of mean task duration data from the market seems justified.

4.4.2 Setting the total slack

The plan may have a hard deadline, which may be set by a user or determined by existing commitments for tasks that cannot be started until tasks in the current RFQ are complete. Otherwise, in the normal case, the bid-process plan is expected to set the time limits for the RFQ.

It would be interesting to find a way to use the market to dynamically derive a schedule that maximizes the customer's payoff. This would require cooperation of bidders, and could be quite costly. Parkes and Ungar [Parkes and Ungar, 2001] have done something like this in a restricted domain, but it's hard to see how to apply it to the more generalized MAGNET domain.

4.4.3 Task ordering

For any pair of tasks in the plan that could potentially be executed in parallel, we may have a choice of handling them in parallel, or in either sequential order. For example, in our wine-bottling example, we could choose to acquire the bottles before buying the corks. This example is a bit contrived, perhaps, but if there is uncertainty over the ability to complete tasks which could cause the plan to be abandoned, then the agent's financial exposure can be affected by task ordering. If a risky task is scheduled ahead of a "safe" task, then if the risky task fails we can abandon the plan without having to pay for the safe task. Babanov [Babanov *et al.*, 2002] has worked out in detail how to use task completion probabilities and discount rates in an expected-utility framework to maximize the probabilistic "certain payoff" for an agent with a given risk-aversion coefficient.

For some tasks, linearizing the schedule will extend the plan's makespan, and this must be taken into account in terms of changes to the ultimate payoff. Note that in many cases the agent may have flexibility in both the start time and the completion time of the schedule. This would presumably be true of our wine-bottling example.

4.4.4 Allocating time to individual tasks

Once we have made decisions about the overall time available and about task ordering, the CPM algorithm gives us a set of preliminary time windows. In most cases, this will not produce the best results, for several reasons:

Resource availability – In most markets, services will vary in terms of availability and resource requirements. There may be only a few dozen portable bottling and labeling machines in the region, while corks may be stored in a warehouse ready for shipping. There is a high probability that one could receive several bids for delivery of corks on one specific day, but a much lower probability that one could find even one bid for a 6-day bottling job for a specific 6-day period. More likely one would have to allow some flexibility in the timing of the bottling operation in order to receive usable bids.

Lead-time effects – In many industries, suppliers have resources on the books and on the payroll that must be paid for whether their services are sold or not. In these cases, suppliers will

typically attempt to "book" commitments for their resources into the future. In our example, the chances of finding a print shop to produce our labels tomorrow is probably much lower than the chances of finding shops to print them next month. This means that, at least for some types of services, one must allow more scheduling flexibility to attract short lead time bids than for longer lead times. Typically, we should also expect to pay more for shorter lead times.

Task-duration variability – Some services are very standardized (delivering corks, printing 5000 labels), while others may be highly variable, either because they rely on human creativity (software development) or the weather (bridge construction), or because different suppliers use different processes, different equipment, or different staffing levels (wine bottling). These two types of variability can usually be differentiated by the level of predictability; suppliers that uses a predictable process with variable staffing levels are likely to be able to deliver on time on a regular basis, while services that are inherently unpredictable will tend to exhibit frequent deviations from the predictions specified in bids[3].

For services that exhibit a high variability in duration, as specified in bids, the customer's strategy may depend on whether a large number of bidders is expected, and whether there is a correlation between bid price and quoted task duration. If a large number of bidders is expected, then the customer may be able to allocate a below-average time window to the task, in the expectation that there will be some suppliers at the lower end of the distribution who will be able to perform within the specified window. On the other hand, if few bidders are expected, a larger than average time window may be required in order to achieve a reasonable probability of receiving at least one usable bid.

Excessive allocations to non-critical tasks – One obvious problem with the time allocations from the CPM algorithm as shown in Figure 4.4 is that non-critical tasks (tasks not on the critical path) are allocated too much time, causing unnecessary overlap in their time windows. All other things being equal, we are likely to be better off if the RFQ time windows do not overlap, because we will have fewer infeasible bid combinations.

4.4.5 Trading off feasibility for flexibility

In general we expect more bidders, and lower bid prices, if we offer suppliers more flexibility in scheduling their resources by specifying wider time windows. On the other hand, if we define RFQ time windows with excessive overlap, a significant proportion of bid combinations will be unusable due to schedule infeasibility. The intuition is that there will be some realistic market situations where the customer is better off allowing RFQ time windows to overlap to some degree, if we take into account price, plan completion time, and probability of successful plan completion (which requires at minimum a set of bids that covers the task set and can be composed into a feasible schedule). We have not developed the data or the analysis to prove this intuition or to guide the agent in setting

[3]Whether the market or customers would be able to observe these deviations may depend on market rules and incentives, such as whether a supplier can be paid early by delivering early.

its time windows, but we have designed the winner-determination procedure to handle schedule infeasibilities among bids.

Figure 4.5 shows a possible updated set of RFQ time windows for our wine-bottling example, taking into account the factors we have discussed. We have shortened the time windows for tasks s_1 and s_2, because we believe that bottles and corks are readily available, and can be delivered when needed. There is no advantage to allowing more time for these tasks. Market data tells us that bottling services are somewhat more difficult to schedule than labeling services, and so we have specified a wider time window for task s_3 than for s_4. Our deadline is such that the value of completing the work a day or two earlier is higher than the potential loss of having to reject some conflicting bids. We also know from market data that a large fraction of suppliers of the bottling crews can also provide the labeling service, and so the risk of schedule infeasibility will be reduced if we receive bids for both bottling and labeling. Finally, there is plenty of time available for the non-critical label-printing task s_5 without needing to overlap its time window with its successor task s_4.

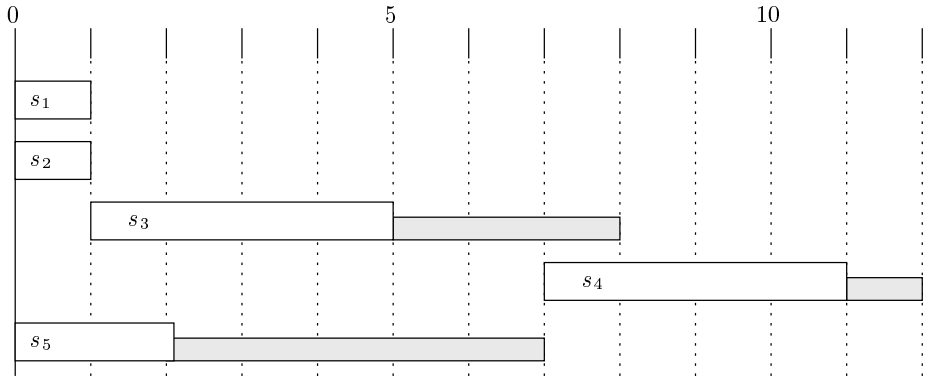

Figure 4.5: Revised time allocations for tasks in RFQ r_1.

4.5 Evaluating Bids

Once an RFQ is issued and the bids are returned, the agent must decide which bids to accept. The bidding process is an extended combinatorial auction, because bids can specify multiple tasks, and there are additional constraints the bids must meet (the precedence constraints) other than just covering the tasks. The winner-determination process must choose a set of bids that maximize the agent's utility, cover all tasks in the associated RFQ, and form a feasible schedule.

4.5.1 Formal description of the winner-determination problem

Each bid represents an offer to execute some subset of the tasks specified in the RFQ, for a specified price, within specified time windows. Formally, a bid $b = (r, \mathcal{S}_b, \mathcal{W}_b, c_b)$ consists of a subset $\mathcal{S}_b \in \mathcal{S}_r$ of the tasks specified in the corresponding RFQ r, a set of time windows $\mathcal{W}_b$, and an overall cost c_b. Each time window $w_s \in \mathcal{W}_b$ specifies for a task s an earliest start time $t_{es}(s, b)$, a latest start time $t_{ls}(s, b)$, and a task duration $d(s, b)$.

It is a requirement of the protocol that the time window parameters in a bid b are within the time windows specified in the RFQ, or $t_{es}(s,b) \geq t_{es}(s,r)$ and $(t_{ls}(s,b)+d(s,b)) \leq t_{lf}(s,r)$ for a given task s and RFQ r. This requirement may be relaxed, although it is not clear why a supplier agent would want to expose resource availability information beyond that required to respond to a particular bid. For bids that specify multiple tasks, it is also a requirement that the time windows in the bids be internally feasible. In other words, for any bid b, if for any two of its tasks $(s_1, s_2) \in \mathcal{S}_b$ there is a precedence relation $s_1 \prec s_2$ specified in the RFQ, then it is required that $t_{es}(s_1,b) + d(s_1,b) \leq t_{ls}(s_2,b)$.

A solution to the bid-evaluation problem is defined as a complete mapping $\mathcal{S} \rightarrow \mathcal{B}$ of tasks to bids in which each task in the corresponding RFQ is mapped to exactly one bid, and that is consistent with the temporal and precedence constraints on the tasks as expressed in the RFQ and the mapped bids.

Figure 4.6 shows a very small example of the problem the Bid Evaluator must solve. As noted before, there is scant availability of bottling equipment and crews, so we have provided an ample time window for that activity. At the same time, we have allowed some overlap between the bottling and labeling tasks, perhaps because we believed this would attract a large number of bidders with a wide variation in lead times and lower prices. Bid 1 indicates this bottling service is available from day 3 through day 7 only, and will take the full 5 days, but the price is very good. Similarly, bid 2 offers labeling from day 7 through day 10 only, again for a good price. Unfortunately, we can't use these two bids together because of the schedule infeasibility between them. Bid 3 offers bottling for any 3-day period from day 2 through day 7, at a higher price. We can use this bid with bid 2 if we start on day 4, but if we start earlier we will have to handle the unlabeled bottles somehow. Finally, bid 4 offers both the bottling and labeling services, but the price is higher and we would finish a day later than if we accepted bids 2 and 3.

4.5.2 Evaluation criteria

We have discussed the winner-determination problem in terms of price, task coverage, and schedule feasibility. In many situations, there are other factors that can be at least as important as price. For example, we might know (although the agent might not know) that the bottling machine being offered in bid 3 is prone to breakdown, or that it tends to spill a lot of wine. We might have a long-term contract with one of the suppliers, Hermann, that gives us a good price on fertilizer only if we buy a certain quantity of corks from him every year. We might also know that one of the the local printers tends to miss his time estimates on a regular basis, but his prices are often worth the hassle, as long as we build some slack into the schedule when we award a bid to him.

Many of these factors can be expressed as additional constraints on the winner-determination problem, and some can be expressed as cost factors. These constraints can be as simple as "don't use bid b_3" or more complex, as in "if Hermann bids on corks, and if a solution using his bid is no more than 10% more costly than a solution without his bid, then award the bid to Hermann." Some of them can be handled by preprocessing, some must be handled within the winner-determination process,

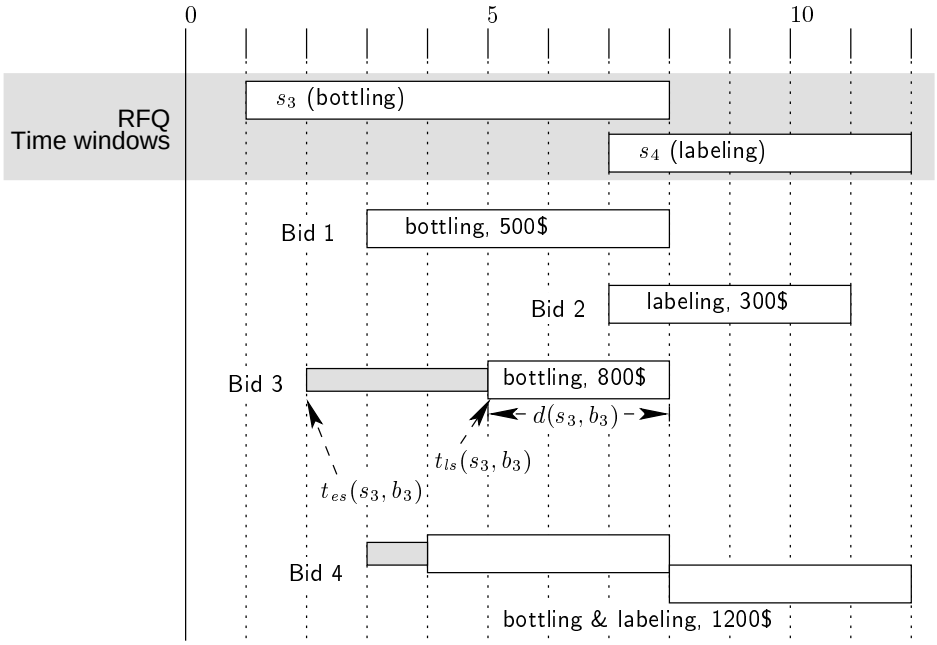

Figure 4.6: Bid Example

and some will require running it twice and comparing results.

4.5.3 Mixed-initiative approaches

As discussed in Section 3.3, there are many environments in which an automated agent is unlikely to be given the authority to make unsupervised commitments on behalf a person or organization. In these situations, we expect that many of the decision processes we discuss here will be used as decision-support tools for a human decision-maker, rather than as elements of a completely autonomous agent. The decision to award bids is one that directly creates commitment, and so it is a prime candidate for user interaction. We have constructed an early prototype of such an interface. It allows a user to view bids, add simple bid inclusion and exclusion constraints, and run one of the winner-determination search methods. Bids may be graphically overlaid on the RFQ, and both the RFQ and Bid time windows are displayed in contrasting colors on a Gantt-chart display.

Effective interactive use of the bid-evaluation functions of an agent require the ability to visualize the plan and bids, to visualize bids in groups with constraint violations highlighted, and to add and update constraints. The winner-determination solver must be accessible and its results presented in an understandable way, and there must be a capability to generate multiple alternative solutions and compare them.

4.6 Awarding Bids

The result of the winner-determination process is a (possibly empty) mapping $\mathcal{S} \rightarrow \mathcal{B}$ of tasks to bids. We assume that the bids in this mapping meet the criteria of the winner-determination process: they cover the tasks in the RFQ and can be composed into a feasible schedule, and they maximize the agent's or user's expected utility. However, we cannot just award the winning bids. In general, a bid b contains one or more offers of services for tasks s, each with a duration $d(s, b)$ within a time window $w(s, b) > d(s, b)$. The price assumes that the customer will specify, as part of the bid award, a specific start time for each activity. Otherwise, the supplier would have to maintain its resource reservation until some indefinite future time when the customer would specify a start time. This would create a disincentive for suppliers to specify large time windows, raise prices, and complicate the customer's scheduling problem.

This means that the customer must build a final work schedule before awarding bids. We will defer to the next section the issue of dealing with schedule changes as work progresses. This scheduling activity represents another opportunity to maximize the customer's expected utililty. In general, the customer's utility at this point is maximized by appropriate distribution of slack in the schedule, and possibly also by deferring task execution in order to defer payment for completion.

4.7 Execution Management

Once bids are awarded and work commences, the customer must manage progress of the plan and deal with unexpected events that could threaten the final payoff for completing the plan. Following are the classes of events to which the agent's Execution Manager must respond, and a brief outline of the agent's response options. Many of them involve negotiations between the customer agent and its suppliers that are outside the scope of the basic auction-style interactions we have explored so far. So far, none of this has been implemented, or even worked out in detail.

Nominal task completion – No action is required.

Early task completion – If a critical path is affected by early completion of a task s, and if the payoff of the plan could be improved by changing the start time for some successor s' of task s, then the agent could request the supplier of s' (and possibly other tasks in the transitive closure of the successors of task s) whether the schedule can be moved up, and for what cost. After evaluating the cost/benefit tradeoff, the agent may request schedule changes accordingly.

Supplier decommitment – If a supplier backs out of a commitment, the customer has three choices:

- The customer can abandon the plan. This would presumably mean abandoning deposits already paid, but could make sense if the completed work has significant residual value for the customer, or if a hard deadline thereby cannot be met.

- The customer may attempt to re-plan and re-bid unsatisfied subgoal(s). This may involve abandoning some deposits, but there may be cases where some other awarded bids could still be honored.
- The customer may attempt to re-bid the decommitted task(s), if there is sufficient slack in the schedule.

The customer agent will need to estimate its expected utility for these alternative courses of action, and we assume that the market will record the failure in its database, which will help customers in the future to judge the reliability of this supplier and apply appropriate discounts to its bids.

Missing event – Completion events are considered missing if their failure to arrive triggers violation of a temporal constraint. This is considered non-performance on the part of the supplier. The agent may respond by notifying the market session of supplier non-performance. It must also determine the impact of the failure on the remainder of its plan. Depending on (so far undefined) circumstances, the customer could treat this as supplier decommitment, or it could discover that the task will likely be completed later than expected. There may be an opportunity to negotiate with suppliers of tasks that are impacted by this schedule disturbance over a change in schedule, and with the late supplier about a discount to cover the extra costs that might be incurred thereby.

Late Completion – A late completion event is one that occurs later than promised but does not violate precedence constraints in the work schedule. We assume that the market session will record the schedule variance for the benefit of future customer decision-making. Market rules may require or create incentive for the supplier to offer a price adjustment in this case.

Notice of Late Completion – If a supplier wishes to extend a task deadline, it may initiate a negotiation with the customer, giving a new expected completion time and an updated bid. The customer must then decide, based on the impact of the resulting schedule variance, whether to accept the updated bid, with the new time commitment, or whether to treat this event as a customer decommitment.

Chapter 5

Solving the MAGNET winner-determination problem

We now focus on the MAGNET winner-determination problem, originally introduced in Section 4.5. Although we would prefer a complete method (one which will either produce an optimal solution or determine that no solution exists, in a bounded amount of time), we are willing to accept incomplete methods if they have good average characteristics. We may also be interested in methods that have good anytime characteristics, in which a usable solution appears early, and is continuously improved. In this chapter, we introduce two complete methods, one based on Integer Programming and one based on the A* method, and an incomplete method based on Simulated Annealing. In addition, the A* method comes in two versions, which trade off time against memory requirements. All of the algorithms are presented in a form that solves the winner-determination problem under assumption of a fixed payoff, and do not deal with a payoff that depends on completion time. We will discuss the necessary adaptations as we review the various methods.

The winner determination problem for combinatorial auctions has been shown to be $\mathcal{NP}$-complete and inapproximable [Sandholm, 1999]. This result clearly applies to the MAGNET winner determination problem, since we simply apply an additional set of (temporal) constraints to the basic combinatorial auction problem, and we don't allow free disposal. In fact, because the additional constraints create additional bid-to-bid dependencies, and because bids can vary in both price and in time specifications, the bid-domination and partitioning methods used by others to simplify the problem (for example, see [Sandholm, 2002]) cannot be applied to the MAGNET problem.

Because there can be no polynomial-time solution, nor even a polynomial-time bounded approximation, we must accept exponential complexity. We will see in Chapter 6 that we can determine probability distributions for search time, based on problem size metrics, and we can use those empirically-determined distributions in our deliberation scheduling process.

5.1 Integer Programming formulation

Andersson [Andersson *et al.*, 2000] introduced an Integer Programming (IP) formulation for the Combinatorial Auction Winner Determination problem. This is attractive, because it means we can solve the problem by building a model rather than by building software, and it takes advantage of investment in optimization amortized across a wide range of problems. Conceptually, Andersson's model is very simple. We assign a 0/1 variable x_i corresponding to each bid b_i, and interpret it such that if $x_i = 1$ then b_i is accepted. For m tasks and n bids, the formulation is

Maximize:

$$\sum_{i=1}^{n} c_i x_i$$

Subject to the m constraints:

$$\sum_{i:s_j \in \mathcal{S}_i} x_i \leq 1, \text{for each } s_j \in \mathcal{S}_r$$

where c_i is the bid price for bid b_i, $\mathcal{S}_i$ is the set of items specified in bid b_i, and the constraints ensure that at most one bid is accepted for each item. The notation $i : s_j \in \mathcal{S}_i$ denotes the set of all i such that the task s_j is in the set of tasks $\mathcal{S}_i$ specified in bid b_i. This formulation will produce maximum revenue in a standard (forward) auction, but it may leave items unallocated. This is known as the *free disposal assumption*, which is valid if none of the items has residual value to the seller. If this is not the case, then it is necessary for the seller to submit its own bids to represent its residual values.

The MAGNET Winner Determination process differs from the "classic" problem described here in 3 major ways:

1. Given the standard agent roles defined in Chapter 3, the MAGNET auction is a reverse auction, and we are interested in minimizing cost rather than maximizing revenue. There may of course also be other factors such as risk, but we ignore them for now.
2. Unless the customer agent is itself capable of performing all tasks in its plan, the free disposal assumption does not apply. All tasks must be covered by exactly one bid.
3. Unless we compose our RFQ so that conforming bids cannot violate precedence constraints, then our problem formulation must ensure than only feasible bid combinations are considered.

5.1.1 The naive approach

We first consider a straightforward extension of the basic formulation described above, as follows:

Minimize:

$$\sum_{i=1}^{n} c_i x_i$$

Subject to:

- Bid selection – each bid is either selected or not selected.

$$x_i \in \{0,1\}, i = \{1 \ldots n\}$$

- Coverage – each task s_j must be included exactly once.

$$\sum_{i:s_j \in \mathcal{S}_i} x_i = 1, \text{for each } s_j \in \mathcal{S}_r$$

- Local feasibility – each task s_j must be able to start after the earliest possible completion time of each of its predecessors $s_{j'}$. This constraint ignores global feasibility. In other words, here we are looking only at the start times of each individual task s_j and its immediate predecessors $\mathcal{V}_j$.

$$\begin{aligned} \text{for each} \quad & s_j \in \mathcal{S}_r, i : s_j \in \mathcal{S}_i, i' : s_{j'} \in (\mathcal{S}_{i'} \cap \mathcal{V}_j), \\ & x_i t_{ls}(s_j, b_i) \geq x_{i'}(t_{es}(s_{j'}, b_{i'}) + d(s_{j'}, b_{i'})) - M(1 - x_i) \end{aligned}$$

  where M is a "large" number (we typically use a value of 10^{12}), and the last term $M(1 - x_i)$ is used to make the constraint satisfied in the case where $x_i = 0$.

  Figure 5.1 shows a small example with 3 tasks, an RFQ with overlapping time windows, and 4 bids. The infeasibility between bids b_1 and b_2 in this example is captured by the constraint

$$x_1(7.0) \geq x_2(5.0 + 3.0) - M(1 - x_1).$$

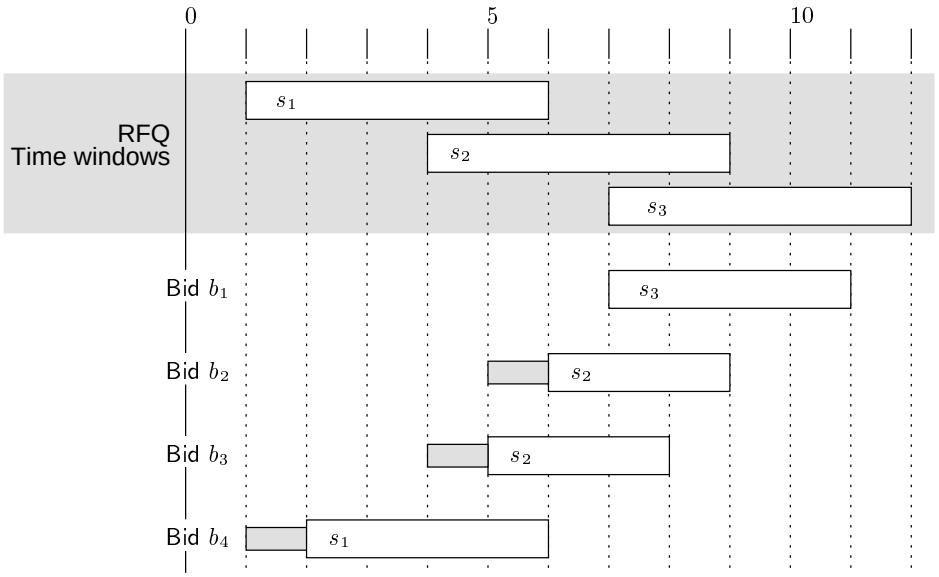

Figure 5.1: Sample bids showing 2-bid and 3-bid infeasibilities.

- Global feasibility – each task must be able to start after the earliest possible completion time of each of its predecessors, where the predecessors may in turn be constrained not by their

bids, but by their respective predecessors.

$$\begin{aligned}
&\text{for each} \quad s_j \in \mathcal{S}_r, i : s_j \in \mathcal{S}_i, i' : s_{j'} \in (\mathcal{S}_{i'} \cap \mathcal{V}_j), i'' : s_{j''} \in (\mathcal{S}_{i''} \cap \mathcal{V}_{j'}),\\
&\qquad x_i t_{ls}(s_j, b_i) \geq x_{i'} d(s_{j'}, b_{i'}) + x_{i''}(t_{es}(s_{j''}, b_{i''}) + d(s_{j''}, b_{i''})) - M(1 - x_i)\\
&\text{for each} \quad s_j \in \mathcal{S}_r, i : s_j \in \mathcal{S}_i, i' : s_{j'} \in (\mathcal{S}_{i'} \cap \mathcal{V}_j), i'' : s_{j''} \in (\mathcal{S}_{i''} \cap \mathcal{V}_{j'}), i''' : s_{j'''} \in (\mathcal{S}_{i'''} \cap \mathcal{V}_{j''}),\\
&\qquad x_i t_{ls}(s_j, b_i) \geq \; x_{i'} d(s_{j'}, b_{i'}) + x_{i''} d(s_{j''}, b_{i''})\\
&\qquad\qquad\qquad +x_{i'''}(t_{es}(s_{j'''}, b_{i'''}) + d(s_{j'''}, b_{i'''})) - M(1 - x_i)\\
&\quad \vdots
\end{aligned}$$

In the example of Figure 5.1, bids b_1 and b_3 can be combined by setting the start time of b_3 to 4.0, and bids b_3 and b_4 can be combined by setting the start time of b_4 to 1.0 and of b_3 to 5.0. However, the three bids $\{b_1, b_3, b_4\}$ clearly cannot be used together. This constraint is expressed as

$$x_1(7.0) \geq x_3(3.0) + x_4(1.0 + 4.0) - M(1 - x_1).$$

The number of constraints generated by these formulas is highly variable, depending strongly on the details of the submitted bids and how they interact with the precedence network in the plan. For example, in the 5 tasks – 20 bids case described in Section 6.2.2, the count varied from 10 to 24000.

5.1.2 Preprocessing to reduce problem size

The formulation given in Section 5.1.1 is correct, but it can be dramatically improved, based on several observations. The first is that if there are any uncovered tasks, then no solution is possible. An uncovered task is simply a task $s_j \in \mathcal{S}_r$ that is not included in the task set $\mathcal{S}_i$ of any bid $b_i \in \mathcal{B}$.

The second observation is that we can pre-process the coverage constraints to reduce the number of bids. If there is any task s_j for which only one bid b_i has been received (we'll call bid b_i a "singleton" bid for task s_j), b_i must be part of any complete solution. Bids b_k that conflict with b_i can then be discarded. In more formal terms,

$$\text{for each } s_j \in \mathcal{S}_r, \text{ if } |\{i : s_j \in \mathcal{S}_i\}| = 1, \text{ then } x_i = 1, \text{ and for each } k \text{ s.t. } \mathcal{S}_i \cap \mathcal{S}_k \neq \emptyset, x_k = 0$$

The bids b_k found by this process are then removed from $\mathcal{B}$, and the test repeated until no further singleton bids are detected.

Next, we make the following observations regarding the feasibility constraints:

1. We may have generated feasibility constraints between bids that cannot possibly be part of the same solution, because they contain overlapping task sets. The coverage constraints will ensure that only one bid will be chosen to cover each task. We can discard such constraints immediately. In our example, this means that we need not generate or evaluate any feasibility constraints that include both b_1 and b_3.

2. The feasibility constraints can be greatly simplified by doing the arithmetic during preprocessing, and including only those constraints that can have an impact on the outcome. This way, we eliminate all the feasibility constraints shown in Section 5.1, and replace them with a much smaller number of simple compatibility constraints. In other words, if we start with a constraint of the form

$$x_i t_{ls}(s_j, b_i) \geq x_{i'}(t_{es}(s_{j'}, b_{i'} + d(s_{j'}, b_{i'})) - M(1 - x_i),$$

we compute $t_{ls}(s_j, b_i) - (t_{es}(s_{j'}, b_{i'}) + d(s_{j'}, b_{i'}))$. Just in case the result is negative, we include a constraint of the form

$$x_i + x_{i'} \leq 1$$

which will prevent both b_i and $b_{i'}$ from being part of the same solution. This simplification can be similarly applied to the global feasibility constraints. In general, such constraints tell us that for some combination of n bids, at most $n - 1$ of them may be part of a solution.

If either b_i or $b_{i'}$ in the above formula is a singleton, then clearly the other cannot be part of a solution, so it can be eliminated. Also, if both b_i and $b_{i'}$ are singletons, then we know the problem cannot be solved.

In the example of Figure 5.1, we can observe the infeasibility among b_1, b_3, and b_4 during pre-processing, and rather than generate the formula given above in Section 5.1, we replace it with

$$x_1 + x_3 + x_4 \leq 2$$

3. If we have successive tasks in the same bid, we can filter the bids themselves for internal feasibility prior to evaluation. After the previous step, any constraints of the form $x_i + x_i \leq 1$ represent bids b_i that can be discarded. Of course, every time a bid is discarded, we have another opportunity to discover singletons and uncovered tasks.

With this background, we describe our preprocessing in more formal detail. The first step is the detection of singleton bids and the removal of bids that conflict with them, as described above. The next step is to enumerate conflict sets among the remaining bids. For each bid b_i, its conflict set $\mathcal{C}_i$ is defined as the set of bids $b_{i'}$ whose task sets overlap the task set of b_i, i.e.

$$\mathcal{C}_i = \{b_{i'} \in (\mathcal{B} - b_i) \text{ such that } \mathcal{S}_{i'} \cap \mathcal{S}_i \neq \emptyset\}$$

Clearly, only a single bid from any conflict set can be included in any solution.

The final preprocessing step converts precedence relations among tasks to compatibility constraints among bids. For each bid b_i we walk the task precedence network $\mathcal{V}_r$ to discover whether the time windows specified for other bids $b_{i'}$ are compatible with the time windows specified for b_i. Each conflicting time window results in an infeasible bid combination g. Infeasible combinations may be of any length $n = 2 \ldots k$ where k is the maximum depth of the precedence network. They are defined

as:

$$\begin{aligned}
\mathcal{G}_2 &= \{b_i \in \mathcal{B}, b_{i'} \in (\mathcal{B} \setminus \mathcal{C}_i) \text{ such that} \\
&\quad \exists s_j \in \mathcal{S}_i, s_{j'} \in (\mathcal{V}_j \cap \mathcal{S}_{i'}) \text{ for which } t_{ls}(s_j, b_i) < (t_{es}(s_{j'}, b_{i'}) + d(s_{j'}, b_{i'}))\} \\
\mathcal{G}_3 &= \{b_i \in \mathcal{B}, b_{i'} \in (\mathcal{B} \setminus \mathcal{C}_i), b_{i''} \in (\mathcal{B} \setminus (\mathcal{C}_i \cup \mathcal{C}_{i'})) \text{ such that} \\
&\quad \exists s_j \in \mathcal{S}_i, s_{j'} \in (\mathcal{V}_j \cap \mathcal{S}_{i'}), s_{j''} \in (\mathcal{V}_{j'} \cap \mathcal{S}_{i''}) \\
&\quad \text{for which } t_{ls}(s_j, b_i) < (t_{es}(s_{j''}, b_{i''}) + d(s_{j''}, b_{i''}) + d(s_{j'}, b_{i'}))\} \\
&\vdots
\end{aligned}$$

The final IP formulation is then

Minimize:

$$\sum_{i=1}^{n} c_i x_i$$

Subject to:

- Bid selection – each bid is either selected or not selected. These are the integer variables that make this an integer programming problem.

$$x_i \in \{0, 1\}$$

- Coverage – each task s_j must be included exactly once. This is an encoding of the coverage conflict sets discussed above.

$$\sum_{i \mid s_j \in \mathcal{S}_i} x_i = 1, \text{ for each } s_j \in \mathcal{S}_r$$

  Note that under the free disposal assumption, each task would be included at most once, rather than exactly once.

- Feasibility – The final preprocessing step described above produces sets $\mathcal{G}_n$ of bids that cannot be combined feasibly. We must specify that none of these sets can be entirely included in any solution.

$$\sum_{i \mid b_i \in g_k} x_i < n, \text{for each } n, \text{ for each } g_k \in \mathcal{G}_n$$

A major weakness of this formulation is that the preprocessing discards all the temporal data, reducing it to the compatibility constraints among bids we have just discussed. As a consequence, it is not possible to extend this formulation to optimize a time factor. One obvious application for this would be to minimize completion time by adding a weighted factor to the objective function that corresponds to the earliest possible completion time of the latest task. If there is not a single task at the tail of the task network, then a dummy completion task can be added that depends on

completion of all other tasks, and a dummy bid added that allows that task to be executed at any time with a duration of 0.0. This works with the original formulation in Section 5.1.1, but not with this formulation.

A second weakness of linear or integer programming for our winner-determination problem is that it is not easy to model preferences over how slack is distributed in the schedule that results from choosing a set of bids. Even in the "naive" formulation, it is not possible to express a preference such as "prefer bid combinations that allow a more equal distribution of slack over those with more skewed distribution." Such preferences typically are not linear; once sufficient slack exists between two tasks, more is not necessarily better. Worse, because schedule slack is typically a risk-management strategy, one would typically prefer to distribute more slack after tasks that are allocated to less reliable bidders. The usual method for dealing with such concerns in linear/integer programming models is to solve the problem without the extra constraints, and then re-run the model repeatedly, adding additional constraints to attempt to improve trouble spots. This might be helpful if the IP solver performed significantly better than the alternative methods we will explore, but that is not the case.

5.2 Simulated Annealing formulation

Simulated Annealing (or simply SA) is a stochastic optimization approach based on a mathematical analogy to the physical process of annealing [Kirkpatrick *et al.*, 1983]. The basic approach is to combine a hill-climbing search with a random walk, in such a way that the ratio of hill-climbing to random varies from a low value to a high value as the search progresses.

Both queue-based and depth-first formulations of Simulated Annealing are possible, and the search can be structured as a constructive search, in which a solution is built up step by step, or as a search among alternative complete solutions, in which all variables begin with "valid" values and the search proceeds by "switching" the values of variables and evaluating the results. Examples of depth-first search among complete solutions include Kirkpatrick's formulation of the Traveling Salesman problem [Kirkpatrick *et al.*, 1983], and Kautz and Selman's work on Propositional Satisfiability [Kautz and Selman, 1996]. The abstract core of these algorithms is shown in Figure 5.2.

In order to use this approach, the problem should ideally be such that a complete solution is easily constructed, and that applications of the move function F to states in the search space lead to other complete solutions. The *Casanova* system of Hoos and Boutilier [Hoos and Boutilier, 2000] uses this approach starting with an empty allocation, but it assumes free disposal (an empty allocation is a valid, but presumably suboptimal solution) and uses a ranking scheme that cannot easily be mapped to the MAGNET domain. If an empty allocation is not valid, as is the case for the MAGNET problem, then dummy bids may be used to construct the initial state. The dummy bids must be such that they are less attractive than any possible real bid, and if free disposal is not allowed, then a final result that includes dummy bids must be counted as failure.

A queue-based approach to Simulated Annealing can also be used. In this case, we use a sorted

```
 1  Inputs:
 2      s: a state within the search space
 3      f(s): the evaluation of state s
 4      F := [s ⇒ s']: a function that performs a "move" in the search space
 5      T: the current annealing temperature 0 ≤ T ≤ 1
 6      ε: the "annealing rate" used to lower T on each cycle
 7      k: the "randomization constant" used to control the ratio of random
           to hill-climbing behavior
 8  Output:
 9      s_best: the best state found
10  Process:
11      loop
12          s' ← F(s)
13          if (f(s') < f(s)) ∨ (e^{-(f(s')-f(s)/kT)} < random(0,1))
            then
14              s ← s'  "accept s'"
15              if f(s) < f(s_best) then
16                  s_best ← s
17          T ← T(1 − ε)  "lower the temperature"
18      end loop
19      return s_best
```

Figure 5.2: Abstract depth-first Simulated Annealing algorithm.

queue, usually of limited size, and during each cycle we make a random selection from the queue, rather than taking the first node. The selection is typically taken from an exponential distribution so that the most likely choice is the first node, and the "temperature" is used to control the mean of the distribution. This is the approach we have taken.

The performance of Simulated Annealing is strongly dependent on the cost of computing $F(s)$ and $f(s)$, and in the queue-based case, on the size of the queue and the efficiency of queue management.

5.2.1 Basic framework

Our simulated-annealing framework is a queue-based search, characterized by two elements, the annealing temperature T which is periodically reduced by a factor ε, and the stochastic node-selection procedure shown in Figure 5.4.

A high level algorithm for our Simulated Annealing winner determination search is depicted in Figure 5.3. We use a basic simulated-annealing framework [Reeves, 1993], with a finite queue (maximum length is *beam_width*) and a number of enhancements to minimize wasted effort. Many complications are omitted, such as the fact that `expand_node` (see Figure 5.5) may fail to produce a node. This can happen, for example, because the selected bid has already been tried against the selected node, or because of an optional uniqueness test.

Stopping conditions include an empty queue and a node-count limit, as shown in Figure 5.3, line 16. Search can also be stopped externally (from a separate thread) by imposing a time limit. The

```
1  Procedure simulated_annealing
2  Inputs:
3      {S, V}: the task network to be assigned
4      B: the set of bids
5      N_0: initial node, containing mapping of tasks to singleton bids
6  Output:
7      N_best: the node having a mapping M(N_best) = S → B of tasks
               to bids with the best known evaluation
8  Process:
9      Q ← priority queue of maximum length beam_width,
10         sorted by node evaluation f(N)
11     f(N_0) ← evaluate_node(N_0) "see narrative in this section"
12     N_best ← N_0 "initialize the result"
13     insert(Q, N_0) "insert the first node into the queue"
14     T ← T_0 "set the initial annealing temperature"
15     Z ← 0 "initialize the improvement counter"
16     while (¬empty(Q) ∧ (Z < patience) do
17         N ← select_node(Q, T) "see Figure 5.4"
18         B ← select_bid(N, B)
19         N' ← expand_node(N, B) "see Figure 5.5"
20         f(N') ← evaluate_node(N') "see narrative in this section"
21         insert(Q, N')
22         if f(N') < f(N_best) then
23             N_best ← N'
24             Z ← 0
25         else
26             Z ← Z + 1
27         T ← T(1 − ε) "the value ε is the annealing rate"
28     return N_best
```

Figure 5.3: Simulated Annealing algorithm for winner determination.

variable Z in lines 24 and 26 is simply the number of nodes that have been generated without finding an improved evaluation score. It is typically a function of the problem size, but may also be a function of the number of nodes generated prior to the latest improved score. The queue can become empty if we use an optional feature that keeps track of the bids that have been used to expand the node in the past; when all bids that do not conflict with bids already in the node (where "conflict" can include the conflict sets and feasible bid combinations found for the IP solver) have been tried, the node is removed from the queue. This greatly reduces redundant effort in the search at the cost of keeping track of the remaining compatible bid set per node. Although this is a somewhat unorthodox approach for a stochastic search, it has proved to be very valuable in our situation because of the high cost of node evaluation. On the other hand, this approach allows considerable freedom in defining the node-evaluation function, and so it is possible to specify evaluation functions that balance slack, prefer more slack after bids by unreliable suppliers, or prefer earlier completion times.

When a stopping condition other than time limit occurs, an optional restart protocol can be used. In most cases, search time is better spent running several shorter searches, rather than a single long search. We shall see examples of this in Section 6.3.

```
1  Procedure select_node
2  Inputs:
3      Q: the current node queue, sorted so that f(Q_n) ≤ f(Q_{n+1})
           "f(Q_n) is the evaluation of the n^th node in Q"
4      T: the current annealing temperature 0 ≤ T ≤ 1
5  Output:
6      N_next: the selected node
7  Process:
8      r ← random(0,1) "a random number uniformly distributed between 0 and 1"
9      R ← f(Q_0) − T ln(1 − r)(f(Q_last) − f(Q_0))
           "R is called the 'target evaluation' "
           "Q_0 and Q_last are the first and last nodes in the queue, respectively"
10     n ← 0
11     while f(Q_{n+1}) ≤ R do
           "find the last node whose evaluation is less than or equal to the target"
12         n ← n + 1
13     return Q_n
```

Figure 5.4: Simulated Annealing node-selection algorithm.

Bid selection can be configured in a number of ways. The simplest selector simply makes a random choice among bids that are not part of the current mapping, and that are compatible with the bids currently mapped in the node. Others may focus on improving coverage, feasibility, or cost. Extensive experimentation has failed to show that any more informed method performs consistently better than a simple random approach.

Node expansion (see Figure 5.5) is done by adding mappings of the selected bid to all the tasks covered by that bid, discarding mappings of any other bids that overlap the new bid. This will fail if the selected bid has already been used to expand the selected node, or because the bid was used to produce the current node. Not shown is the fact that expansion can also fail if an attempt is made to unmap a singleton bid (a bid that provides the only possible mapping for some task). Of course, no bids will ever be unmapped if we filter bid selections with the conflict sets $\mathcal{C}_i$, which is one of our tuning options.

Node evaluation produces a value $f(N)$ for node N which is a weighted sum of the following three factors:

- *coverage*: how many tasks are covered? a bid that covers tasks that are not yet covered should be preferred over a bid that covers tasks already covered. We apply a penalty for each uncovered task to drive the search toward a complete solution.
- *feasibility*: is the current partial solution feasible? One way to determine this is to apply the Critical Path algorithm. If it finds any negative slack, the current partial solution is not feasible, since it violates some time constraint. We can apply a penalty for infeasibility, or we can penalize each unit of negative slack. This penalty is not important if infeasible nodes are never added to the queue.

```
1  Procedure expand_node
2  Inputs:
3      N: the node to be expanded
4      B: the bid to be added to M(N), the task-bid mapping of N
5  Output:
6      N': a new node with a mapping that includes B, or null if the mapping fails
7  Process:
8      if (B ∈ B_{M(N)}) ∨ (B ∈ tried(N)) then return null
               "B is already in mapping of N, or as been tried previously."
9      N' ← copy(N)
10     insert(tried(N'), B)  "tried is a set"
11     S' ← S_B ∩ S_{M(N')}  "S' is the set of tasks in both B and N'.M */
12     B' ← b ∈ B_{M(N')} such that S_b ⊂ S'
               "B' is the set of bids in M(N') whose tasks overlap the tasks in B"
14     for each b ∈ B', s ∈ S_b, M(N') ← M(N') − m(s, b)  "remove mappings for B' */
15     for each s ∈ S_B, M(N') ← M(N') + m(s, B)  "add the mappings for B"
16     return N'
```

Figure 5.5: Simulated Annealing node-expansion algorithm.

- *cost*: what is the total cost of all the bids in the node?

Other factors could also be included, such as a "schedule-risk" computation that evaluates the distribution of schedule slack. The cost of node evaluation strongly limits the performance of the algorithm. Cost and coverage can be computed in $O(m)$ time, where m is the number of tasks in the RFQ. The feasibility test requires $O(md)$ time, where d is the maximum length of the transitive precedence relation in the task network.

5.2.2 Tuning

Stochastic search algorithms in general and simulated annealing in particular can be tuned in a wide variety of ways. There is limited guidance in the literature on how to do this tuning, although there is good guidance in evaluating the results of tuning efforts (see for example [Hoos, 1998]). In this section we explore the more significant tuning parameters that can be manipulated in our implementation. All of these parameters are specified in the configuration file, making experiments with them straightforward. The results of a series of these experiments are presented in Section 6.3.4.2.

Temperature profile – At "high" temperatures, SA exhibits highly random behavior, with little focused hill-climbing, while at "low" temperatures, the focus is on hill-climbing. The parameters of interest here are the initial temperature and the rate of decay. In our implementation, the maximum temperature is 1.0, and the minimum is 0.0. Empirical testing has shown that a "good" schedule starts with a temperature in the range of 0.3-0.5 and decays to a value within an order of magnitude of 0.001. Decay is controlled by a "multiplier" ε as shown in Figure 5.3, line 27. In the actual code, we also use a "rate" parameter r and only apply the multiplier every r iterations. Therefore the effective multiplier is $\varepsilon = \varepsilon'/r$ where ε' and r are

the actual setup parameters used. This was done to make scaling easier, as we shall see below under "adaptive tuning".

Stopping Condition – Since SA is an incomplete search method and is not able to prove optimality of a solution, either in the local sense or in the global sense, it will not stop unless it exhausts its queue. Therefore, there must be some way to stop the search. We can either try to determine that further search effort is unlikely to lead to improvement, or we can place a hard time limit on the search. Our formulation allows us to do both. To determine whether further search effort is likely to lead to further improvement, the usual approach, and the one we take, is to keep track of the number of nodes explored since the last improvement was seen (the last time N_{best} was updated – line 23 in Figure 5.3). The limit can either be fixed, or it can be readjusted by a *stop-ratio* every time a new best solution is found. So, for a fixed patience factor of 300, the search will stop after 300 nodes have been generated without seeing a new best solution. With a stop-ratio of 1.5, on the other hand, if a new best solution was found in node 800, then the limit would be reset to 1200.

Queue length – As discussed earlier, we use a queue-based implementation. This can provide for parallel exploration of many alternatives, since each iteration begins by selecting a random entry from the queue. The queue is sorted by $f(N)$, and the focus shifts toward the front of the queue as the temperature is reduced. To aid performance and limit memory usage, we place an upper limit on the length of the queue. If the queue limit is too short, then it is easy for the queue to become dominated by entries that are all in the neighborhood of a single local minimum. If it is too long, then the $n \log n$ complexity of adding and finding nodes becomes expensive. We generally use values within an order of magnitude of 1000.

Adaptive tuning – Stochastic search is not a magic bullet that will turn our $\mathcal{NP}$-complete problem into a $\mathcal{P}$ problem. As we shall see in Section 6.3, exponential complexity is observed with respect to both task count and bid count. If we ignore this and leave the patience factor, the temperature profile, and the queue size constant as we vary the problem size, what we observe is that we simply don't find solutions to the larger problems.

To accommodate problem-size variation, we compute a *scale-factor* at the beginning of the search, and use it to adjust the temperature profile, patience factor, and queue length. We call these three parameters the *scalable parameters*. The formula for computing the scale-factor is itself a parameter, in the form of a Scheme script that has access to problem-size data and to the values of the various parameters. Its output is a number α. If the parameter values given in the configuration file for annealing multiplier, patience factor, and queue length are z_0, ρ_0, and ϕ_0 respectively, then the scale factor is applied to produce a temperature multiplier $\varepsilon = \varepsilon_0/\alpha$, a patience factor $\rho = \alpha\rho_0$, and a queue length $\phi = \alpha\phi_0$.

Restarts – Most stochastic search methods start their searches in one (usually randomly chosen) location in the search space and then do some sort of random walk in the space. We cannot start in a random location, since we always start with the node that has no bids. This is true of any search that must construct a solution from an empty mapping. The consensus (see, for

example, [Hoos, 1998, Kautz and Selman, 1996, Reeves, 1993]) is that the best use of time is to run several shorter searches starting from multiple locations in the space, rather than spending all the available time on one search. The MAGNET system takes a parameter specifying a maximum number of restarts, and can be configured to run all of them, to stop as soon as a solution is found, to stop at the end of the cycle during which a solution is found, or to run n additional cycles after a solution is first found. In addition, we can increase the scale factor by a ratio on each restart.

Experiments show that the required search effort is strongly correlated with the number of bids in the solution (see Section 6.3.4). The result is that the strategy of gradually increasing the scale factor on each restart will tend to accentuate the difficulty of finding solutions with large numbers of bids. For this reason, a more effective use of restarts is to adjust the scale-factor by an exponentially-distributed random number on each restart.

External stopping condition To support search performance experiments, the SA solver can also be configured to continue running until an external condition is signaled. This can be used to place a time limit on the search, or to keep the search running until a solution is found that meets some criterion. This capability is necessary to support analysis of the time required to find optimal solutions, and requires that one first use a provably optimal search method to find the value of an optimal solution.

Bid selection – A major design choice in building a stochastic search for a discrete domain is to determine the definition of a "move" in the space, and to have some way to choose the next move. Moves may be completely random, or they may be more or less informed to encourage hill-climbing. In the MAGNET SA search engine, we define a move as the addition of a bid to a partial solution (line 18 in Figure 5.3). Because the task sets of bids can overlap, adding a bid may displace a bid that is already mapped in the original solution. The choice of bid to add may be arbitrary, or it may be entirely random, or it may be a random choice among some chosen subset.

The SA search engine uses "bid selectors" to choose bids for expansion. A number of bid selectors have been written, with different types and different levels of informed behavior. Some of them use the current annealing temperature to control their degree of randomness. We have studied their performance in [Collins *et al.*, 2000b], and the conclusion is that no method yields better performance than one called "random-usable-bid" which makes a random choice among all the bids that do not conflict with the bids currently mapped in the node.

Whether to allow infeasible nodes – Adding a bid to a partial solution always runs the risk of producing infeasibility, unless the choice of bids is limited to those that cannot produce infeasibility. As we shall see in Section 6.2, the cost of preprocessing to find infeasible bid combinations can be quite significant. The alternative is to try an expansion and then determine whether the resulting node is feasible. Then we have to determine whether to keep the node for further expansion, or to discard it immediately. Clearly, discarding infeasible nodes will close some paths in the search space. However, for an infeasible node to lead to a feasible

solution, one of the bids that is responsible for the infeasibility would have to be displaced by a further expansion. This may or may not be a high-probability event. If we are using a bid selector that avoids bid-displacement, then the probability drops to zero. Experimentation has shown no advantage to allowing infeasible nodes to be candidates for further expansion.

Uniqueness testing – A problem with stochastic search in general is that there is often no easy way to prevent the search from visiting the same location in the search space multiple times. There are two general ways to prevent this. The first is the use of a "tabu list" which is typically the set of expansions that have been tried in the "recent" history of the node [Reeves, 1993]. In other words, if bid b_5 has been chosen within the last n rounds (n is usually a relatively small number, between 5 and 15), then its use is blocked. Since our formulation uses a queue, a given node may be chosen for expansion multiple times, particular if it hangs out near the front of the queue, and so in our case there is a second way that duplication can result: the same bid may be chosen twice to expand a given node. We prevent this by keeping track of the bids that have been tried against a given node, and we disallow attempts to try the same expansion twice.

The second, more robust way to direct the search into new territory, is to prevent a given location in the space from being used twice at all. This "uniqueness test" can be somewhat expensive, but in our case it is less expensive than the feasibility test that must be performed on each node, and it has proved to be of considerable value. It is accomplished by creating a "name" for each node by concatenating the ID strings of all the bids (the bid ID strings are guaranteed to be unique among bids), and maintaining a hash set of the names that have been encountered so far.

Penalties – Another design choice we must make is how to evaluate partial solutions (and infeasible solutions if we allow them to be entered into the queue). For the A* formulation in Section 5.3 below, we use an "admissible heuristic" to ensure that when a complete solution shows up on the front of the queue, it is guaranteed to be optimal. For this to be the case, the estimate to complete an incomplete solution $h(N)$ must be "optimistic". If we were to use an admissible heuristic in the SA search, we would end up doing something very similar to a breadth-first search, because large numbers of incomplete solutions would inhabit the front of the queue. If instead we use a pessimistic evaluation for incomplete nodes, our search will have a more depth-first flavor, and the hill-climbing behavior will drive preferentially toward complete solutions. This is important for larger problems and smaller bids, because the path to a solution is as long as the number of bids in the solution.

To be a pessimistic evaluation, the value of the heuristic completion function $h(N)$ must be an upper bound on the cost of mapping all the unmapped tasks. We can use either a large fixed constant that is guaranteed to be larger than any bid price, or we can use for each unmapped task s_u a value that is at least as large as the highest per-task cost in any of the bids that specify that task. If we disallow infeasible nodes, this approach forces the queue to strictly order nodes by the number of mapped tasks. Experience shows that a larger penalty value

drives the search more strongly toward a solution, with the caveat that such solutions are on average farther from optimal.

5.3 Optimal tree search formulation

Sandholm recently published an approach to solving the standard combinatorial auction winner-determination problem [Sandholm, 2002] using an iterative-deepening A* formulation. Although many of his optimizations, such as the elimination of dominated bids and partitioning of the problem, cannot be easily applied to the MAGNET problem, we have adapted the basic structure of Sandholm's formulation, and we have improved upon it by specifying a means to minimize the mean branching factor in the generated search tree.

In general, tree search methods are useful when the problem at hand can be characterized by a solution path in a tree that starts at an initial node (root) and progresses through a series of expansions to a final node that meets the solution criteria. Each expansion generates successors (children) of some existing node, expansions continuing until a final node is found. The questions of which node is chosen for expansion, and how the search tree is represented, lead to many different search methods. In the A* method, the node chosen for expansion is the one with the "best" evaluation[1], and the search tree is typically kept in memory in the form of a sorted queue. A* uses an evaluation function

$$f(N) = g(N) + h(N)$$

for a node N, where $g(N)$ is the cost of the path from initial node N_0 to node N, and $h(N)$ is an estimate of the remaining cost to a solution node. If $h(N)$ is a strict lower bound on the remaining cost (upper bound for a maximization problem), we call it an *admissible heuristic* and A* is complete and optimal; that is, it is guaranteed to find a solution with the lowest evaluation, if any solutions exist, and it is guaranteed to terminate eventually if no solutions exist.

We describe here both the basic A* formulation of the MAGNET winner-determination problem, and then we show how this formulation can be adapted to a depth-first iterative-deepening model [Korf, 1985] to reduce or eliminate memory limitations.

5.3.1 Bidtree framework

For a basic introduction to the A* algorithm, see [Russell and Norvig, 1995], or another textbook on Artificial Intelligence. Our formulation depends on two structures which must be prepared before the search can run. The first is the *bidtree* introduced by Sandholm, and the second is the *bid-bucket*, a container for the set of bids that cover the same task set.

A bidtree is a binary tree that allows lookup of bids based on item content. The bidtree is used to determine the order in which bids are considered during the search, and to ensure that each bid combination is tested at most once. In Sandholm's formulation, the collection of bids into groups

[1] lowest for a minimization problem, highest for a maximization problem.

that cover the same item sets supports the discard of dominated bids, with the result that each leaf in the bidtree contains one bid. However, because our precedence constraints create dependencies among bids in different buckets, bid domination is a much more complex issue in the MAGNET problem domain. Therefore, we use bid-buckets at the leaves and defer the bid-domination issue for now.

The principal purpose of the bidtree is to support content-based lookup of bids. Suppose we have a plan $\mathcal{S}$ with tasks $s_m, m = 1..4$. Further suppose that we have received a set of bids $b_n, n = 1..10$, with the following contents: $b_1 : \{s_1, s_2\}$, $b_2 : \{s_2, s_3\}$, $b_3 : \{s_1, s_4\}$, $b_4 : \{s_3, s_4\}$, $b_5 : \{s_2\}$, $b_6 : \{s_1, s_2, s_4\}$, $b_7 : \{s_4\}$, $b_8 : \{s_2, s_4\}$, $b_9 : \{s_1, s_2\}$, $b_{10} : \{s_2, s_4\}$. Figure 5.6 shows a bidtree we might construct for this problem. Each node corresponds to a task. One branch, labeled *in*, leads to bids that include the task, and the other branch, labeled *out*, leads to bids that do not.

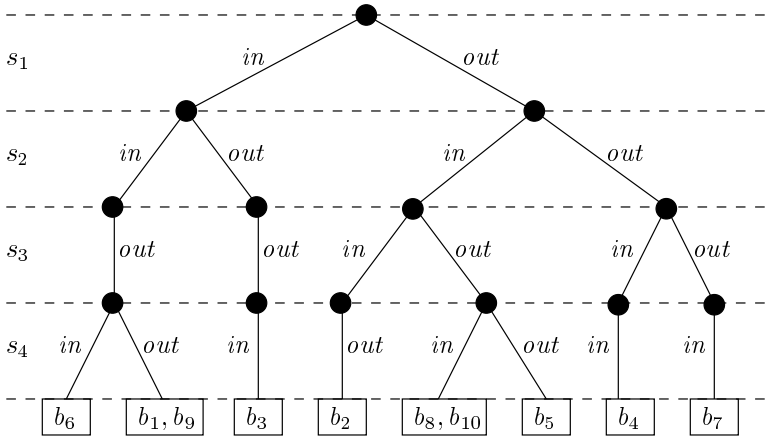

Figure 5.6: Example bidtree, lexical task order

We use the bidtree by querying it for bid-buckets. A query consists of a *mask*, a vector of values whose successive entries correspond to the "levels" in the bidtree. Each entry in the vector may take on one of three values, {*in*, *out*, *any*}. A query is processed by walking the bidtree from its root as we traverse the vector. If an entry in the mask vector is *in*, then the *in* branch is taken at the corresponding level of the tree, similarly with *out*. If an entry is *any*, then both branches are taken at the corresponding level of the bidtree. So, for example, if we used a mask of [*in*, *any*, *any*, *in*], the bidtree in Figure 5.6 would return the bid-buckets containing $\{b_6\}$ and $\{b_3\}$.

A bid-bucket is a container for a set of bids that cover the same task set. In addition to the bid set, the bid-bucket structure stores the list of other bid-buckets whose bids conflict with its own (where we use "conflicts" to mean that they cover overlapping task sets). These lists correspond to the $\mathcal{C}_i$ in Section 5.1.2. Here, we recognize the fact that all bids with the same task set will have the same conflict set.

In order to support computation of the heuristic function, we use a somewhat different problem formulation for A* and IDA* than the IP formulation shown above in Section 5.1. In that formulation, we were minimizing the sum of the costs of the selected bids. In this formulation, we minimize the

cost of each of the tasks, given a set of bid assignments. This allows for straightforward computation of the A* heuristic function $f(N)$ for a given node N in the search tree. We first define

$$f(N) = g(\mathcal{S}_m(N)) + h(\mathcal{S}_u(N))$$

where $\mathcal{S}_m(N)$ is the set of tasks that are mapped to bids in node N, while $\mathcal{S}_u(N) = \mathcal{S}_r \setminus \mathcal{S}_m(N)$ is the set of tasks that are not mapped to any bids in the same node. We then define

$$g(\mathcal{S}_m(N)) = \sum_{j|\, s_j \in \mathcal{S}_m} \frac{c(b_j)}{n(b_j)}$$

where b_j is the bid mapped to task s_j, $c(b_j)$ is the total cost of b_j, $n(b_j)$ is the number of tasks in b_j, and

$$h(\mathcal{S}_u(N)) = \sum_{j|\, s_j \in \mathcal{S}_u} \frac{c(b_j^\star)}{n(b_j^\star)}$$

where $b_j^\star$ is the "usable" bid for task s_j that has the lowest cost/task. By "usable," we mean that the bid $b_j^\star$ includes s_j, and does not conflict (in the sense of having overlapping task sets) with any of the bids b_j already mapped in node N.

Note that, unlike the case with the IP solver, the definition of $g(\mathcal{S}_m(N))$ can be expanded to include other factors, such as risk estimates or penalties for inadequate slack in the schedule, and these factors can be non-linear. The only requirement is that any such additional factor must *increase* the value of $g(\mathcal{S}_m(N))$, and not decrease it, because otherwise the admissibility of the heuristic will be compromised, and we no longer would have a complete search method.

5.3.2 A* formulation

Now that we have described the bidtree and bid-bucket, we can explain our optimal tree search formulation. The algorithm is given in Figure 5.7.

The principal difference between this formulation and the "standard" A* search formulation (see, for example, [Russell and Norvig, 1995]), is that nodes are left on the queue (line 15) until they cannot be expanded further, and only a single expansion is tried (line 17) at each iteration. This is to avoid expending unnecessary effort evaluating nodes.

The expansion of a parent node N to produce a child node N' (line 17 in Figure 5.7) using the bidtree is shown in Figure 5.8. Here we see the reason to keep track of the buckets for the candidate-bid set of a node. In line 16, we use the mask for a new node to retrieve a set of bid-buckets. In line 18, we see that if the result is empty, or if there is some unallocated task for which no usable bid remains, we can go back to the parent node and just dump the whole bucket that contains the candidate we are testing.

In line 17 of Figure 5.8, we must find the minimum-cost "usable" bids for all unallocated tasks $\mathcal{S}_u$ (tasks not in the union of the task sets of $\mathcal{B}_{N'}$), as discussed earlier. One way (not necessarily the

```
1  Procedure A*_search
2  Inputs:
3      {S, V}: the task network to be assigned
4      B: the set of bids, represented as a bidtree
5  Output:
6      N_opt: the node having a mapping M(N_opt) = S → B
               of tasks to bids with an optimal evaluation, if one exists
7  Process:
8      Q ← priority_queue, sorted by node evaluation f(N)
9      N_0 ← empty node
10     mask ← {in, any, any, cdots}
11     B^c_{N_0} ← bidtree_query(B, mask)
           "B^c_{N_0} is a set of bids (in the form of a set of bid buckets), containing the
           bids that can be used to expand N_0"
12     insert(Q, N_0)
13     loop
14        if empty(Q) then return failure
15        N ← first(Q)
16        if solution(N) then return N
17        N' ← astar_expand(N) "see Figure 5.8"
18        if N' = null then remove_front(Q) "Remove nodes when they fail to expand"
19        else if feasible(N') then insert(Q, N')
```

Figure 5.7: Bidtree-based A* search algorithm.

most efficient way) to find the set of usable bids is to query the bidtree using the mask that was generated in line 14, changing the single *in* entry to *any*. If there is any unallocated task that is not covered by some bid in the resulting set, then we can discard node N' because it cannot lead to a solution (line 22). Because all other bids in the same bidtree leaf node with the candidate bid b_x will produce the same bidtree mask and the same usable-bid set, we can also discard all other bids in that leaf node from the candidate set of the parent node N.

This implementation is very time-efficient, as we will see in Chapter 6, but A* fails to scale to large problems because of the need to keep in the queue all nodes that have not been fully expanded. Limiting the queue length destroys the optimality and completeness guarantees. Some improvement in memory usage can be achieved by setting an upper bound once the first solution is found in line 18 of Figure 5.8. Once an upper bound *flimit* exists, then any node N for which $f(N) > flimit$ can be safely discarded, including nodes already on the queue. Unfortunately, this helps only on the margin; there will be a very small number of problems for which the resulting reduction in maximum queue size will be sufficient to convert a failed or incomplete search into a complete one.

5.3.3 Bidtree ordering

One of the design decisions that must be made when implementing a bidtree-based search is how to order the tasks (or items, in the case of a standard combinatorial auction) when building the bidtree. It turns out that this decision can have a major impact on the size of the tree that must be searched, and therefore on performance and predictability.

```
1  Procedure astar_expand
2  Inputs:
3      N: the node to be expanded
4  Output:
5      N′: a new node with exactly one additional bid, or null
6  Process:
7      buckets ← ∅
8      while buckets = ∅ do
9          if B^c_N = ∅ then return null "B^c_N is the set of candidate bids for node N"
10         b_x ← choose(B^c_N) "pick a bid from the set of candidates"
11         B^c_N ← B^c_N − b_x "remove the chosen bid from the set"
12         N′ ← new node
13         B_N′ ← B_N + b_x "B_N′ is the set of bids in node N′"
14         S_u ← unallocated_tasks(N′) "tasks not covered by any bid b ∈ B_N′"
15         mask ← create_mask(B′_N)
               "for each task that is covered by a bid in B_N′, set the corresponding
               entry to out. Then find the first task in s ∈ S_u (the task in S_u with the
               minimum index in the bidtree) and set its entry to in. Set the remaining
               entries to any"
16         buckets ← bidtree_query(B, mask)
17         B_u ← ∀s ∈ S_u, minimum_usable_bid(s) "see the narrative"
18         if (solution(N′))
               ∨((buckets ≠ ∅) ∧ (¬∃s ∈ S_u|minimum_usable_bid(s) = null))
19         then
20             B^c_N′ ← buckets "candidates for N′"
21         else
22             remove(B^c_N, bucket(b_x))
                   "all bids in the bucket containing b_x in node N will produce the same
                   mask and therefore an empty candidate set or a task that cannot be
                   covered by any usable bid"
23     end while
24     g(N′) ← Σ_{b∈B_N′} c_b
25     h(N′) ← Σ_{b∈B_u} avg_cost(b)
26     return N′
```

Figure 5.8: Bidtree-based node-expansion algorithm.

We define a *complete partition* of the bids $\mathcal{P} \subseteq \mathcal{B}$ to be a set of bids such that each task $s \in \mathcal{S}$ is covered by exactly one bid $b \in \mathcal{P}$. The cost of a complete partition $\mathcal{P}$ is simply the sum of the costs of the bids in $\mathcal{P}$. In these terms, a solution to the MAGNET winner-determination problem for m tasks $\mathcal{S} = \{s_1, \cdots, s_m\}$ and n bids $\mathcal{B} = \{b_1, \cdots, b_n\}$ would be defined as the minimum-cost complete partition $\mathcal{P}_{\min}$ that is consistent with the temporal and precedence constraints on the tasks as expressed in the RFQ and the mapped bids.

Note that if a node N is a solution, then $\mathcal{B}_N$ is a complete partition; otherwise, it is an incomplete partition $\mathcal{P}_s(N)$. For every incomplete partition, there is a set of "complementary partitions" $\mathcal{P}_{\overline{s}}$ that can be combined with $\mathcal{P}_s(N)$ to form complete partitions. A subset of the complementary partitions is associated with each of the candidate bids for node N. Clearly, the *branching factor* β for a node N (the number of child nodes N' in the search tree) is exactly equal to the number of candidate bids that do not lead to node abandonment in line 22 of Figure 5.8. Also, the mean branching factor in

the subtree below N is a function of the size of the set of complementary partitions.

The order in which the tasks are represented in the bidtree can have a major influence on the sizes of the complementary partitions in the search tree. In Figure 5.9, we see bidtrees for the example problem of Figure 5.6, with the tasks sorted by increasing bid count (top), and by decreasing bid count (bottom). At first glance, they seem to be almost mirror images of each other.

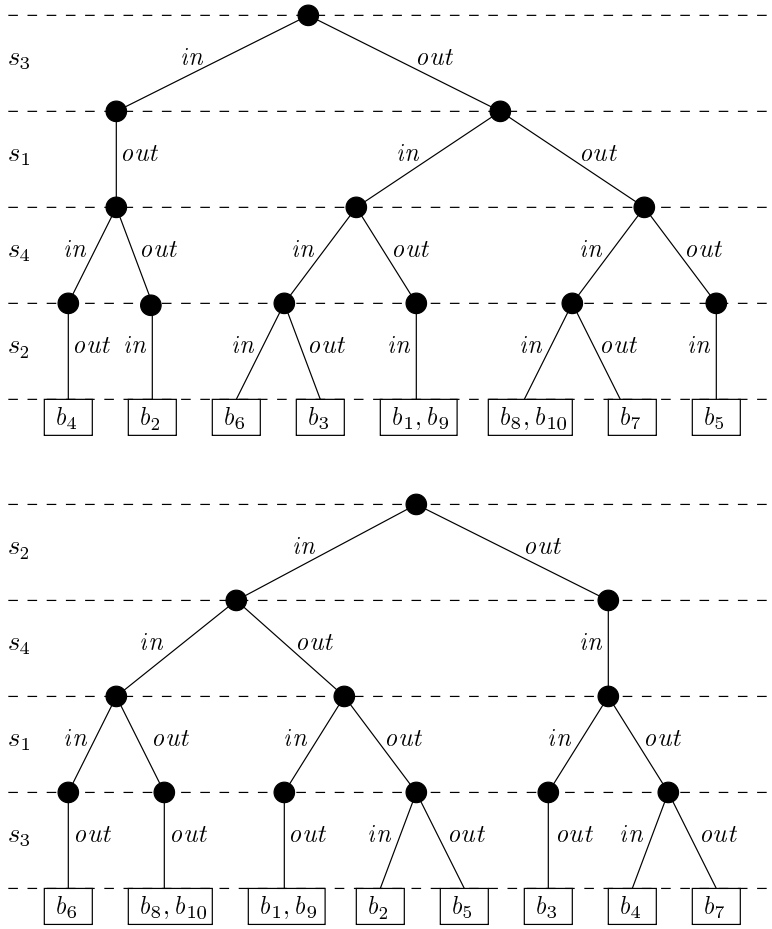

Figure 5.9: Example bidtree sorted by increasing bid count (top) and by decreasing bid count (bottom).

The real difference between the two bidtrees in Figure 5.9 is the size of the complementary partitions in the nodes that are generated using them. This is easy to see with respect to the root node, where the set of candidate bids is the entire left subtree, and the sizes of the complementary partitions for the first tier of children is at most the size of the right subtree. This indicates that the ideal task ordering is one in which the left subtree (the *in* branch) is always maximally larger than the right subtree (the *out* branch). This happens when the task with the largest *in* branch is at the top of the tree, followed by the task with the next largest *in* branch, etc., which corresponds to the ordering in Figure 5.9 (bottom). In Figure 5.10, we see the search trees that are generated by following our process using the two bidtrees shown in Figure 5.9.

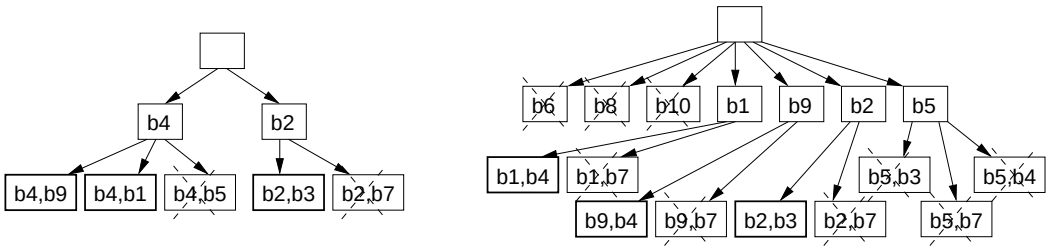

Figure 5.10: Example search trees that result from using bidtrees with increasing bid count (left) and decreasing bid count(right). Incomplete nodes that can have no children are crossed out, and complete nodes are outlined in bold boxes.

The number of nodes N in a tree of uniform depth d with uniform branching factor β is

$$N = \beta^{d-1},$$

counting the root node as depth 1. In our winner-determination search tree, the maximum depth is equal to the number of bids in a solution, which is bounded by m, the number of tasks in the task network. So, the number of nodes in our search tree is bounded by

$$N \leq \beta^{m},$$

since the root node has no bids. We can achieve a tighter estimate if we make two simplifying assumptions: first, that all bids contain the same number of tasks

$$bs^{\star} = \overline{|\mathcal{S}_i|}, i \in n,$$

where bs is the number of tasks/bid, the *bidsize*, and second that all tasks have attracted the same number of bids

$$bt^{\star} = n \cdot bs^{\star}/m,$$

where bt is the number of bids/task. Then the depth d of the search tree, after the root node, will be the number of bids required to cover all the tasks,

$$d = m/bs^{\star},$$

the branching factor (ignoring task coverage conflicts among the bids) will be

$$\beta^{\star} = bt^{\star},$$

and the node count will be

$$N = (n \cdot bs^{\star}/m)^{m/bs^{\star}}.$$

This is an estimate, not a bound, since in a real problem the assumptions will seldom hold, and a pathological case could easily be constructed in which the solution would be composed entirely of bids that were much smaller than average, thus increasing the depth of the tree.

Another way to place an upper bound on the branching factor β at each level in the tree is as follows: Suppose we have m tasks $\mathcal{S} = \{s_1, s_2, \cdots, s_m\}$ and n bids $\mathcal{B} = \{b_1, b_2, \cdots, b_n\}$, where the m tasks are included $\alpha_1, \alpha_2, \cdots, \alpha_m$ times in the bids. Suppose $\alpha_1 > \alpha_2 > \cdots > \alpha_m$. We term task s_1 the *most preferred* task since it has attracted the largest number of bids.

1. Suppose that the search tree is expanded starting with the most preferred task s_1. After expanding the root node using the bids containing s_1, we have α_1 nodes in the second tier of the tree. At this level, then, each node contains a bid for s_1 and can have at most $n - \alpha_1$ children. Obviously, this is a quite rough upper bound since each bid has certain subset of tasks, and for a particular node there might be no complementary partition at all. With similar reasoning, one expects at most $n - \alpha_1 - \alpha_2$, $n - \alpha_1 - \alpha_2 - \alpha_3$, so on children for each node in the third, fourth etc. tiers.

2. Alternatively, let us start by expanding the root node with the least preferred task s_m. Then the subsequent third, fourth etc. depths of the search tree will have at most $n - \alpha_m$, $n - \alpha_m - \alpha_{m-1}$, $n - \alpha_m - \alpha_{m-2}$ etc. children for each node.

A comparison of 1 and 2 reveals that, if there were no conflicts among the bids (which can happen if bids contain single tasks or if they cover disjoint sets of tasks), the number of nodes generated would be the same for both orderings. However, in general we would expect at least some bids to conflict with each other. A bid for $\{s_1, s_2\}$ conflicts with another bid for $\{s_2, s_3\}$ because they both specify task s_2.

The advantage of the decreasing sort is exploited in line 18 of Figure 5.8, where bid conflicts are detected. With the decreasing sort, conflicts are detected higher in the tree, because of the smaller complementary partitions that are available to the bidtree query (line 16 of Figure 5.8). If we assume random distribution of tasks among bids (not necessarily a supportable assumption), then consider the case when expanding the second tier of the tree: With the decreasing sort, we have $n - \alpha_1$ bids to choose from to cover $m - 1$ tasks (ignoring the fact that some bids in the second tier will cover multiple tasks), and those are the tasks that are least likely to be covered by bids, since there are fewer bids that cover them. On the other hand, with the increasing sort we have $n - \alpha_m$ bids available to cover the $m - 1$ tasks. Clearly the probability of achieving coverage is higher with the increasing sort, and so the detection of conflicts is deferred to lower in the tree. It is possible that a more sophisticated method for computing the heuristic $h(N')$ in line 25 of Figure 5.8 would reduce this disparity, but that would very likely amount to solving the problem in order to solve the problem.

The arguments above give us upper bounds, and they can obviously overestimate the children at a node. For a more realistic description of the relative branching factors, one has to deal also with the lower bounds on the children of a node. However, distributions of bids over the tasks, as well as their interdependencies, strongly affect the lower bound. In fact, as shall see later in Section 6.4, the increasing sort does indeed occasionally win.

5.3.4 Iterative Deepening A*

Iterative Deepening A* (IDA*) is a variant of A* that uses the same two functions g and h in a depth-first search, which keeps in memory only the current path from the root to a particular node. In each iteration of IDA*, search depth is limited by a threshold value *flimit* on the evaluation function $f(N)$. We show in Figure 5.11 a version of IDA* that uses the same bidtree and node structure as the A* algorithm. The recursive core of the algorithm is shown in Figure 5.12. This search algorithm uses the same node expansion algorithm as we used for the A* search, shown in Figure 5.8 above.

```
 1  Procedure IDA*_search
 2  Inputs:
 3      {S, V}: the task network to be assigned
 4      B: the set of bids, represented as a bidtree
 5  Output:
 6      N_opt: the node having a mapping M(N_opt) = S → B of tasks to bids
              with an optimal evaluation, if one exists
 7  Process:
 9      N_0 ← empty node
10      g(N_0) ← 0
11      h(N_0) ← Σ_{s∈S} avg_cost(minimum_bid(s))
12      flimit ← f(N_0)
13      mask ← {in, any, any, ···}
14      best_node ← null
15      while (best_node = null) ∧ (flimit ≠ ∞) do
16          B^c_{N_0} ← bidtree_query(B, mask)
                "B^c_{N_0} is a set of bids (in the form of a set of bid buckets),
                containing the bids that can be used to expand N_0. We have to
                repeat this for every iteration."
17          new_limit ← dfs_contour(N_0)
18          if best_node = null then
19              flimit ← max(new_limit, z · flimit)  "see narrative in this section"
20      end while
21      return best_node
```

Figure 5.11: Bidtree-based Iterative Deepening A* search algorithm: top level.

There are three issues to note in this algorithm:

- The tuning parameter z shown in line 19 of Figure 5.11 is a positive number > 1. This controls the amount of additional depth explored in each iteration of the main loop that starts on line 15. Experimentation shows that a good value is 1.15, and that it is not very sensitive (performance falls off noticeably with $z < 1.1$ or $z > 1.2$).
- Whenever a solution is found, the value of *flimit* is updated in line 11 of Figure 5.12. This follows the usage in Sandholm [Sandholm, 2002], and limits exploration to nodes (and solutions) that are better than the best found so far.
- We test nodes for feasibility in line 17 of Figure 5.12 to prevent consideration and further expansion of nodes that cannot possibly lead to a solution.

```
1  Procedure dfs_contour
2  Inputs:
3      N: a node
4  Output:
5      new_limit: a candidate for the flimit value for the next contour. This is
                  either the first f(N) value seen that is larger than flimit, or
                  the value of the best solution node found. If it is determined
                  that no solution is possible, then we return ∞.
6  Process:
7      new_limit ← f(N)
8      if new_limit > flimit then "enforce contour limit"
9          return new_limit
10     if solution(N) then "switch to branch-and-bound"
11         flimit ← new_limit "set new upper bound"
12         best_node ← N
13         return new_limit
14     nextL ← ∞
15     while B^c_N ≠ ∅ do
16         N' ← astar_expand(N)
17         if (N' ≠ null) ∧ (feasible(N')) then
18             new_limit ← dfs_contour(N')
19             nextL ← min(nextL, new_limit)
20     end while
21     return nextL
```

Figure 5.12: Bidtree-based Iterative Deepening A* search algorithm: depth-first contour.

Chapter 6

Search performance

A critical element of a Customer agent's behavior is the determination of winning bids. This is clearly a costly combinatorial problem, and it must be done within the confines of a time-limited negotiation scenario. More significantly, the agent must allocate time to the winner-determination process when it sets the negotiation timeline prior to issuing an RFQ. Failure to solve the winner-determination problem within the allocated time will result in failure of the negotiation process.

For each of the methods outlined in Chapter 5, our goal will be to measure the probability distributions of search times across a range of easily-measured (or easily-estimated) problem metrics. We prefer metrics that can be estimated prior to issuing an RFQ, since that is when deliberation scheduling must be done. This will allow us to schedule the winner-determination deliberation with a known level of confidence in finding a solution.

Two secondary goals are (a) to find combinations of search methods that can be used together (in parallel, for example) to reduce search-time variability, and (b) to evaluate the anytime characteristics of our search methods.

We have chosen four problem size and complexity metrics for evaluation.

Task count – This is simply the number of tasks m in a task network, and can be directly measured.

Bid count – The number of bids submitted n. Alternatively, the number of bids/task. We will assume that this value can be estimated from market statistics, given the task network composition and possibly other data such as lead time or allowable schedule slack.

Bid size – The mean number of tasks specified in each bid $\overline{|\mathcal{S}_i|}, i = 1 \ldots n$. We assume this can also be estimated from market statistics. We can think of a "specialist market" as being one in which most bidders bid on only one or a few task types, while a "generalist market" is one in which many bidders will bid on large chunks of a plan.

Plan complexity – The mean size of the precedence set of a task in the task network. This can be directly measured in the task network.

Some of these problem-size parameters can be controlled independently, and some are not as independent as we might like. For example, when we increase the number of tasks in the task network, we also need to increase the number of bids and/or the number of tasks/bid if we wish to retain the same number of bids for each task. For that reason, we will use bids/task and bid size/task network size when it is important to consider independent variables.

In the remainder of this chapter, we report on a set of experiments that give us the necessary probability distribution data for Integer Programming, Simulated Annealing, A*, and IDA* winner-determination methods. There are clearly limits on scalability of all these methods, because the problem has unavoidable exponential complexity. As we shall see, all of the methods exhibit runtime characteristics with exponential tails, so there will be unpredictable situations where good solutions will not be found within any fixed time limit. This is just another way of saying that 100% probability of finding a solution within a predetermined time limit is not achievable. Instead, our goal is to develop the data that will allow us to convert a desired probability of success to a time allocation, given the necessary problem-size metrics.

6.1 Experimental setup

The experimental setup consists of a plan generator to produce randomly-generated task networks, a bid generator to produce randomly-generated bids for each task network, and a bid evaluator to perform the winner-determination search. The winner-determination process is instrumented to measure both elapsed time and the number of steps performed (equation count for the IP solver, node count for the SA, A*, and IDA* solvers). For the search-based solvers, the time and value of the first solution found is reported. In addition, for the SA solver, some experiments were run in which the times required to find solutions within 5% and 1% of optimal are reported.

Much of the problem-generation process, as well as the Simulated Annealing solver, require a stream of random numbers. In order to be able to repeat test conditions, we maintain 3 separate streams, one for generating plans and RFQs, one for generating bids, and one for the solver. Each can be initialized with a seed, so we have the ability to repeat the same plan sets with different bid sets, run the SA solver repeatedly without disturbing the plan sequence, etc.

With one exception (Section 6.2.1), all experiments were run using a dedicated 1800 MHz Intel/Linux machine. Timings are given in wall-clock time. The MAGNET system (including the IP preprocessor and the Simulated Annealing, A*, and IDA* solvers) is written in Java, and the IP solver is `lp_solve`, written in C and available from `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`.

6.1.1 Customer: Generate plan

For these experiments, plans are randomly-generated task networks, with a number of controllable parameters. Task Network variables include:

Number of tasks

Mix of task types – Task types are characterized by expected duration d_e, duration variability $\sigma(d)$, average price c_e, and price variability $\sigma(c)$. Both duration and price are normally distributed, positive values (the distributions are truncated at 0). For all experiments, the task types and their properties are as given in Table 6.1.

Branch factor – This controls the average number of precedence relationships generated per task, which we call "fan-in." For example, if a particular task has two predecessors, then the fan-in value for that task is 2. As the plan is built, each new task s_j is linked to each of the previous tasks $s_{j'}, j' = 1 \ldots (j-1)$ with a probability p_l of $(branchFactor/(taskCount-1))$. The completed network is filtered to remove most redundant precedence relations, so the final number is lower than the initial number generated.

Table 6.1: Task types and relative proportions used in performance experiments

Name	Proportion	Duration		Price		Resource
		d_e	$\sigma(d)$	c_e	$\sigma(c)$	Availability
short	40%	2.0	0.4	500	0.2	0.8
medium	40%	3.0	0.2	1000	0.3	0.7
long	20%	8.0	0.3	1500	0.3	0.4

As an example, Figure 6.1 shows the task network for the first of the problems used in the bid-count test series described in Section 6.2.2 below. Keep in mind that duration values at this point in the process are expected values. Actual durations cannot be known until bids are awarded.

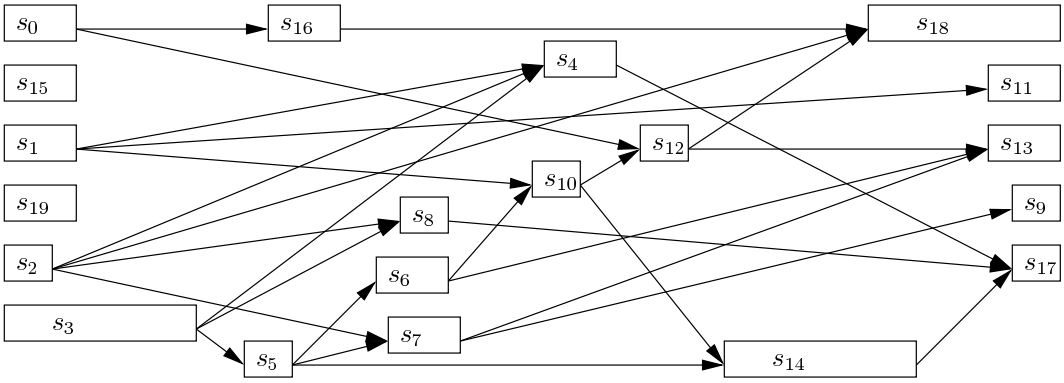

Figure 6.1: Task network for Problem #1. Box widths indicate relative task durations.

6.1.2 Customer: Construct and issue a Request for Quotes

The output of the planning step is simply a task network, made up of tasks for which we have some statistical data. Before we can ask for bids, we must add some constraints to communicate our desires more clearly to potential bidders, and we must consider the availability of bidders and their resources. The goal is to maximize the "usefulness" and minimize the cost of the bids we receive. Ultimately, we would expect to use Babanov's method based on Expected Utility to do this [Babanov *et al.*, 2002], but for these experiments we use a simpler approach that generates bids with well-defined statistics that are adequate to exercise our winner-determination solvers.

As described in Section 4.4, an RFQ is a tuple $\{\mathcal{S}, \mathcal{V}, \mathcal{W}_r \tau\}$. $\mathcal{S}$ and $\mathcal{V}$ are the tasks and precedence relations of the task network, and $\mathcal{W}_r$ is a set if time windows $w_j, j = 1 \ldots m$, one for each task in $\mathcal{S}$. Each time window w in the RFQ specifies the earliest start time $t_{es}(s)$ and the latest finish time $t_{lf}(s)$ for a particular task s. Durations are not specified. We will illustrate the RFQ generation process with the example task network shown in Figure 6.2.

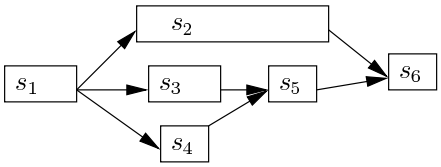

Figure 6.2: Example task network for RFQ illustration.

For these experiments, the RFQ time windows $\mathcal{W}_r$ are computed as follows:

1. Compute the "expected makespan" d_m for the entire task network. The makespan is simply the sum of the expected durations of the "critical" tasks, or the tasks that are on the critical path assuming all tasks are assigned their expected durations.

2. Add some amount of "plan slack" ζ to the task network. This simply relaxes the plan deadline by some fraction. If the start time of the plan is t_0, then the earliest possible completion time, assuming all tasks are completed within their expected durations, is $t_m = t_0 + d_m$. After applying plan slack, we have $t_{goal} = t_0 + \zeta d_m$.

3. Determine final time windows for individual tasks. This is done in two steps. The first step is to set the minimum time allocation $d(s_j)$ for each task s_j to some value $d_{\min}(s_j) = \eta d_e(s_j)$ where $\eta <= \zeta$. Then we run the CPM algorithm with a start time of t_0, and a deadline of t_{goal}. Figure 6.3 shows the resulting start and finish times for the task network of Figure 6.2, using $\zeta = 1.2$ and $\eta = 1.15$.

There is no doubt that this procedure generates time windows that are suboptimal, because of the large overlaps produced for noncritical tasks. However, it is very adequate for our experimental purposes, because it exercises the feasibility testing by ensuring that virtually all problems will have some infeasibilities among bids.

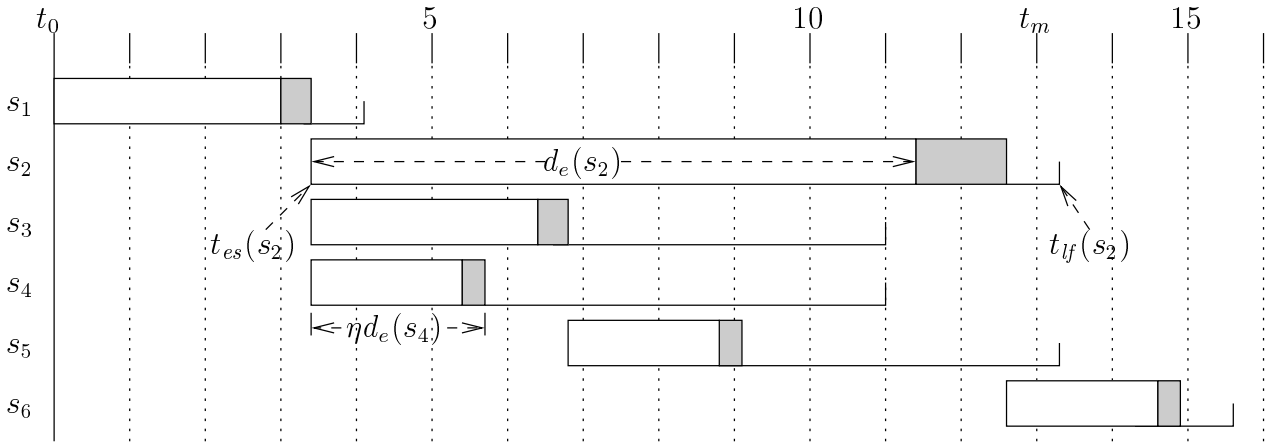

Figure 6.3: Gantt chart for task network of Figure 6.2 with $\zeta = 1.2$, $\eta = 1.15$.

6.1.3 Supplier: Generate Bids

For these experiments, we used a test component called a "supply monster" that masquerades as an entire community of supplier agents. A schematic view of its structure and operation is presented in Figure 6.4. Each time a new RFQ is passed to it, the Bid-Set Generator attempts to generate a specified number of bids.

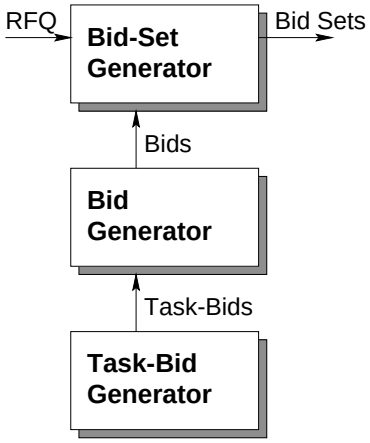

Figure 6.4: The Supply Monster component

For a given RFQ r, the Bid-Set Generator uses the Bid Generator to generate individual bids b_i as follows:

1. The Bid Generator selects a task s_j at random from $\mathcal{S}_r$, and passes it to the Task-Bid Generator to attempt to generate bid parameters for s_j.

2. The Task-Bid Generator uses the task-type parameters to generate a supplier time window $w_j(b_i) = \{t_{es}(s_j, b_i), t_{ls}(s_j, b_i), d(s_j, b_i)\}$ for the task. It first generates the duration $d(s_j, b_i)$

using a normal distribution with mean and standard deviation equal to the expected duration and variability from the task type. If the resulting duration $d(s_j, b_i) > (t_{lf}(s_j, r) - t_{es}(s_j, r))$, then it cannot be used and the attempt is abandoned. Otherwise, an early start $t_{es}(s_j, b_i)$ is generated from a uniform distribution over the interval $[t_{es}(s_j, r), (t_{lf}(s_j, r) - d(s_j, b_i))]$, and a late start over the remaining slack in the RFQ time window $[t_{es}(s_j, b_i), (t_{lf}(s_j, r) - d(s_j, b_i))]$.

Actually, this step is slightly more complex after the first task-bid is generated, because of the need to generate bids with internal feasibility. The Task-Bid Generator simply tightens up the RFQ time window with the maximum early finish of preceding tasks and the minimum late start of succeeding tasks.

3. If the Task-Bid Generator returns a time window, the Bid Generator calls it a "valid task specification" and adds the task s_j and its time window $w_j(b_i)$ to the bid. If a valid task specification was generated, then with probability p_{link}, each predecessor and successor link from task s_j is followed to choose additional tasks $s_{j'}$ to add to the bid, and so on recursively.

4. To complete the process for a single bid, the Bid Generator determines a cost for the overall bid and returns it to the Bid-Set Generator.

5. Because valid task specifications are not always achieved, some attempts to generate bids will fail, and so the number of bids actually generated is nearly always somewhat smaller than the target number. The Bid-Set Generator can optionally check the resulting bid-set before it is returned, and check for coverage. If coverage is not achieved, then for each missing task, we make an additional attempt to generate a bid, with the missing task selected as a starting point in Step 1 rather than a random task. This is useful because returned bid-sets that do not cover all tasks will not exercise our winner-determination solvers. All experiments reported in this chapter used this feature.

The resulting bids specify "contiguous" sets of tasks, and each bid is guaranteed to be internally feasible.

6.1.4 Customer: Evaluate bids

At this point we have an RFQ and a set of bids, and we can run our winner-determination solvers. Our test setup allows us to use any or all of the solvers on each problem, and to do multiple runs of the SA solver on each problem. In addition, we can run the IP or IDA* solver to find an optimal solution, then run the SA solver and capture the first solution found that is at or below any of several multiples of the optimal cost. For example, if we find an optimal solution with IDA* of 11000, then we could capture the time or node-count required to find solutions with evaluation scores of 11550 and 11110 (1.05 and 1.01 times the optimal score) as well as the time/node count to find an optimal solution.

6.2 Characterizing the Integer Programming solver

We start our evaluation exercise with the IP solver, and we consider it to be the baseline case against which the other solvers can be compared. This is because IP is a "generic" solver, not written specifically to solve the winner-determination problem, and because various IP encodings are the most popular winner-determination methods in the literature [Andersson *et al.*, 2000, Nisan, 2000a, Parkes and Ungar, 2001, Rothkopf *et al.*, 1998, Walsh *et al.*, 2000].

Each problem set consists of 100 problems, with randomly-generated task networks and randomly-generated bids as described above. Our problem generator has a large number of parameters; we kept all of them constant except for Task Count, Bid Count, and Bid Size.

6.2.1 Performance with and without preprocessing

The first experiment compares the performance of the original "naive" IP formulation described in Section 5.1.1 to the revised formulation described in Section 5.1.2. We were only able to run the original formulation on the smallest problems (8 tasks, up to 32 bids) because some problems were generating more than 10^5 rows and taking inordinate amounts of time to solve (we stopped one after 13 hours). Table 6.2 shows the results of this experiment. Note that this experiment was run on a different machine from the others (an 833 MHz Linux box) and so the timing numbers are not comparable with the other experiments. All entries are averaged across 200 problems. In the table, the "Rows" columns give the number of constraints in the IP formulation, and for the Revised IP data, we break out the preprocessing time as "PP Time" and the IP solver time as "IP Time." The "σ(time)" column gives the standard deviation of the total solution time for both solvers; for the Original IP sets it is the standard deviation of solver time, while for the Revised IP sets it shows the standard deviation of the sum of PP Time and IP Time values.

Table 6.2: Comparing IP formulations

		Original IP			Revised IP			
Task Count	Bid Count	Rows	Time (msec)	σ(time)	Rows	PP Time (msec)	IP Time (msec)	σ(time)
5	11.4	490	195	1000	16.3	6.57	64.7	144
5	13.6	876	233	937	18.6	8.41	62.5	150
5	15.0	1150	753	5130	20.2	10.9	57.9	125
5	17.0	2280	3150	16300	22.7	15.5	80.4	279
5	18.7	2547	5260	50100	24.1	16.9	48.2	103
6	16.1	1904	2343	12500	22.8	15.0	27.8	26.5
6	18.3	4580	39163	296000	24.7	27.4	307	519
8	25.8	29900	992000	6810000	39.4	58.2	239	690

Key features to observe are the relative problem sizes (number of rows) and the extreme variability (given as σ(time)) of the original formulation. The anomolously low value for IP Time in the Revised IP, 5 task, 18.7 bid row, seems to be due to the fact that there were no large outliers in that particular

set, also indicated by the lower standard deviation. The time required for preprocessing appears to be time well spent. Our experience attempting to solve larger problems with the original formulation shows that the advantage of preprocessing grows dramatically as problem size increases.

6.2.2 Bid count experiment

The next series examines the scalability of the (revised) IP method as the number of bids is varied over a 4:1 range, with task count and bid size held constant. Each row in Table 6.3 represents 100 problems, each with 20 tasks and varying numbers of bids. The same set of 100 task networks is used in each row; only the bid sets are varied. In the table, the "Bid Size" column gives the average size of bids (number of tasks per bid). The "# Solved" column gives the number of problems solved out of 100 (not all 100 problems were necessarily solvable), "Rows" gives the number of constraints in the generated IP problem, "Mean Time" gives the mean search time in milliseconds, including both preprocessor time and time spent in the IP solver itself, and σ gives the standard deviation of the "Mean Time" column. The meaning of the remaining 3 columns will become clear shortly.

It is evident from the table that the mean time scales exponentially with bid count; deviations in mean time and the lower variability at the low end of the range are likely the result of fixed overhead[1]. This will show up much more dramatically in Section 6.2.4 below.

Table 6.3: Bid Count experiment for the IP solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ	m	v	χ^2 (9 dof)
20	51.7	7.36	75	92.2	109	61	95.1	0.255	0.599
20	69.6	7.32	95	125	185	95	163.4	0.260	0.480
20	86.5	7.30	94	193	335	228	278	0.364	1.01
20	104.4	7.35	96	284	501	452	398	0.438	2.77
20	121.0	7.29	98	583	1160	1820	730	0.761	1.59
20	139.2	7.26	100	782	1750	1970	1198	0.714	0.996
20	156.4	7.29	99	1078	3060	3810	1936	0.908	2.74
20	173.2	7.30	100	1691	4500	4220	2960	0.960	0.692
20	190.9	7.37	100	2342	8300	15,000	4410	1.104	3.55
20	207.4	7.32	100	3284	11,800	15,500	6290	1.351	0.208

While the data in Table 6.3 show average performance and give some indication of variability, we are really more interested in knowing the *probability* that a solution can be determined in a given amount of time. For that purpose, we show in Figure 6.5 the complete runtime distributions for five of the problem sets. For each curve, we show actual observations along with a lognormal distribution that minimizes the χ^2 metric[2]. In Table 6.3, the last 3 columns give the parameters of the inferred lognormal density function. The m column is the median value (the value of $\log m$ is the mean of

[1] The IP solver runs as an external Unix process, and must be restarted for each problem

[2] The χ^2 metric is determined by dividing the inferred density function into a number of equal areas, and counting the number of observations that fall into each of those zones. The χ^2 value is a measure of the deviation from a "perfect fit".

the log of the distribution), and the v column is the standard deviation of the log of the distribution. We also give χ^2 values, computed by the equiprobable method [Law and Kelton, 1991]. For a detailed explanation of the procedure, see Section 6.3.4.1 below. For 9 degrees of freedom, $\chi^2 = 3.5$ corresponds to roughly a 95% confidence that the data fit the hypothesized distribution. In general, a large value of χ^2 shows up in a plot such as Figure 6.5 as a very obvious failure of the observations to lie atop the inferred distribution curve.

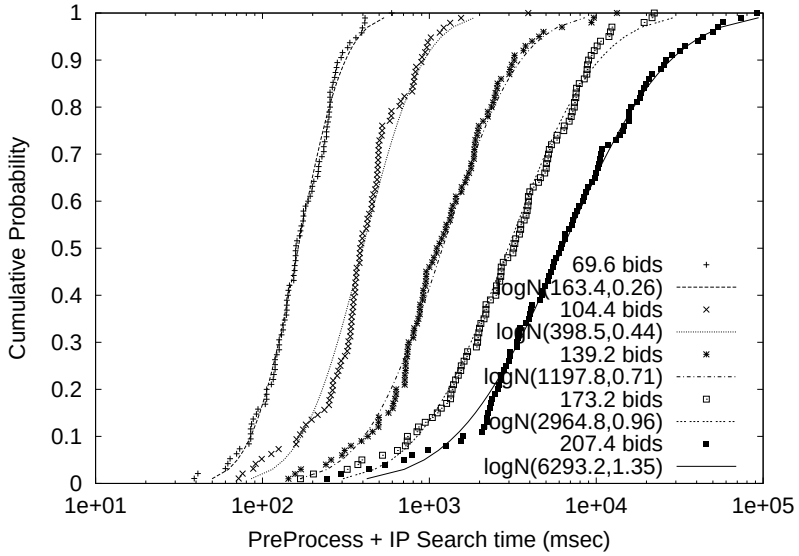

Figure 6.5: Observed and inferred runtime distributions for IP search for a range of bid count values, with task count = 20, and bid size $\approx$ 7.3 tasks/bid.

Note that the first parameter of the lognormal distribution is equal to the median value, which is significantly smaller than the mean for all sets. We choose the lognormal distribution as an initial hypothesis based on the intuition that the log of the runtime distribution should fit a normal distribution – if the random variations among the individual problems were distributed normally, and if the search difficulty scales exponentially with problem characteristics, then the resulting runtime distribution should be lognormal. As we shall see, it fits the data remarkably well.

We now have the the beginnings of a data set that will allow our agent to make decisions about time allocation with a specific degree of confidence. For example, we can say that for a problem the size of our 20-task, 173-bid sample, we have a 95% confidence in finding a solution using the IP solver in less than 10 seconds. In order to make use of this sort of data in an agent environment, we need to know how runtime scales along as many dimensions of problem size as we can reasonably estimate.

The next step is to use the probability data from Figure 6.5 to generate runtime predictions. The process is to choose a desired "probability of success", and then to estimate parameters for a function that reasonably matches the inferred distributions at that level of probability. For example, in Figure 6.6, we show the 95, 50, and 5 percentile points from the inferred distributions from the bid-count experiment. The overlaid curve is $2^{(6.04+0.045x)}$, which yields a root mean square error

$\sqrt{\epsilon^2}$ of 376 msec.

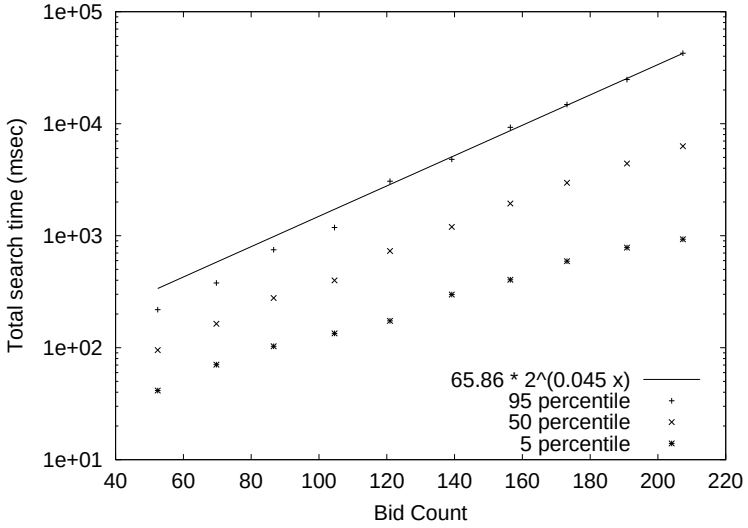

Figure 6.6: Estimating the time required to achieve 95% success rate using the IP solver for 20-task problems across a range of bid counts.

Our goal will be to develop an aggregate empirical formula that can be calibrated on whatever machine an agent finds itself running on, and then to use that formula as input in scheduling the negotiation process.

6.2.3 Bid size experiment

We now turn to the bid-size dimension of problem variability. We expect that larger average bid sizes will result in shorter search times, because the number of bids in a solution will be smaller. This leads to fewer combinations in the search space, and fewer precedence constraints that must be generated and solved (since bids are required to be internally feasible). One way to see this is to look at an upper bound on the number of possible solutions as bid size varies. A simple upper bound is the number of combinations of bids $C = n!/(n - (m/b))!$ where n is the number of bids, m is the number of tasks, and b is the mean number of tasks/bid. Sandholm [Sandholm, 2002, Sandholm *et al.*, 2001] avoids this with small bid sizes by partitioning the bids into non-overlapping subsets. This works because small bids have fewer overlaps, and because the items in a simple combinatorial auction are arbitrarily separable, there being no constraints among them. We can not use this approach because of the presence of precedence constraints that connect the items at auction (the tasks).

Our problem generator uses a somewhat inexact method to "influence" the sizes of bids. As described earlier, the bid-generation process generates "contiguous" bids by choosing a starting point in the task network, and then recursively following predecessor and successor links with some probability. In this series, we have varied that probability from a low of 0.2 to a high of 0.9. Bid size is also influenced by the sizes of time windows, since the "success" in generating a bid for a particular task

is a function of both the size of the time window specified in the RFQ, and the simulated "resource availability" recorded for the type of each task. The task networks in these problem sets are the same as the ones used in the bid-count experiment in the previous section.

As shown in Table 6.4, the resulting bid size varies from a low of 1.6 tasks/bid to 7.9 tasks/bid. Note the extreme difficulty and high variability of the 1.6 tasks/bid set. This happens because the conflict sets for individual bids are smaller, and so we are able to exclude fewer bid combinations from the precedence constraints.

Table 6.4: Bid Size experiment for the IP solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ	m	v	χ^2 (9 dof)
20	87.3	1.59	85	8430	83,200	326,000	3220	5.54	1.23
20	86.8	2.21	98	2440	7280	12,200	2250	2.69	0.291
20	86.8	3.06	84	1760	5460	13,600	1490	2.48	0.492
20	86.5	4.21	100	781	1940	2700	1020	1.31	1.54
20	86.7	5.40	99	374	756	646	564	0.595	0.627
20	87.1	6.33	100	259	459	359	365	0.451	0.606
20	86.6	7.24	99	193	339	245	278	0.377	0.497
20	86.5	7.85	99	164	283	195	240	0.308	0.973

Figure 6.7 shows the runtime observations and inferred distributions for four of the problems from Table 6.4. It appears that bid size variation affects the variability of solution time as much as it affects the mean time. The χ^2 values for this set are all very low, giving 99%+ confidence in the fit of the inferred lognormal distributions.

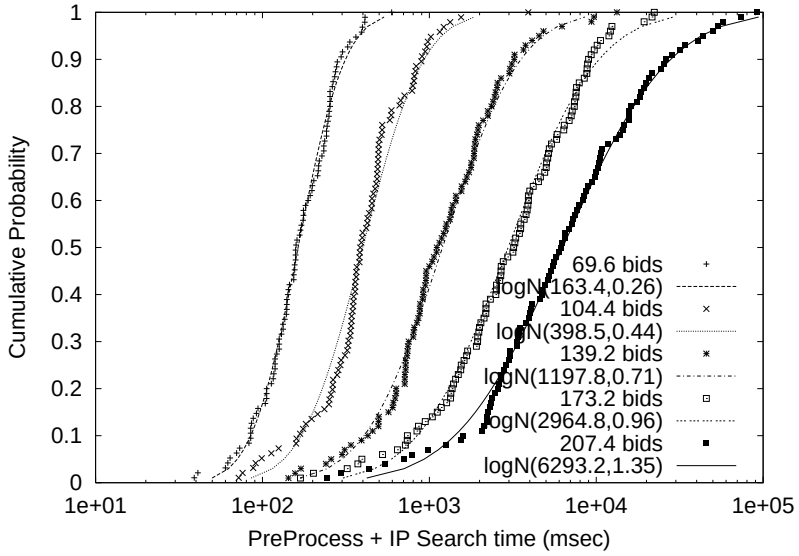

Figure 6.7: Observed and inferred runtime distributions for IP search for a range of bid size values, with task count = 20, and bid count $\approx$ 87.

Now we extend our empirical runtime estimator. In Figure 6.8, we show the 95, 50, and 5 percentile points from the bid-size experiment. The overlaid curve in this plot is $2^{(18.52-1.29x)}$, which yields a $\sqrt{\epsilon^2}$ of 8.2 sec. It seems likely that actual relationship between bid size and run time is slightly more complex than the simple exponential relationship shown here.

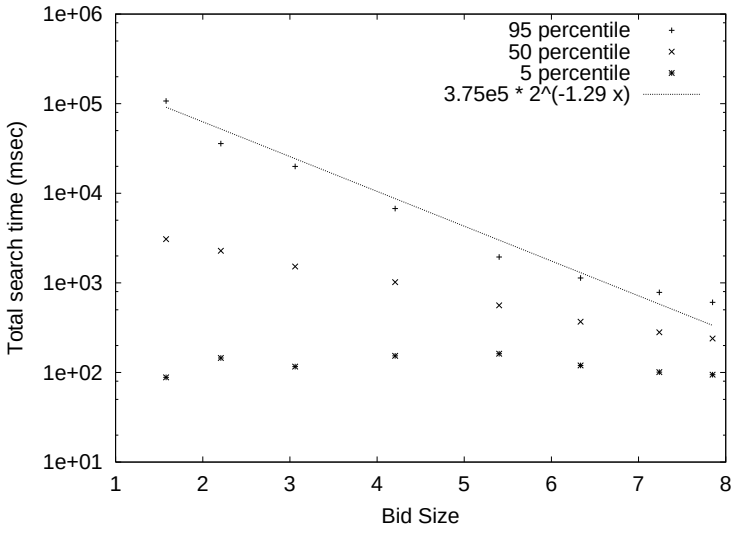

Figure 6.8: Estimating the time required to achieve 95% success rate using the IP solver for 20-task problems across a range of bid sizes.

6.2.4 Task count experiment

The next set of experiments examines scalability of the search process as the number of tasks varies, with a (nearly) constant ratio of bids to tasks (the ratio varies somewhat due to the random nature of the bid-generation process). Table 6.5 shows problem characteristics for this set. Each row in the table shows results for 100 problems. In the table, the "Bid Size" column gives the average size of bids (number of tasks per bid). The ratio of Task Count to Bid Size varies a bit because of the way bids are generated: Bid Size is more closely related to the depth of the precedence network than to Task Count.

Table 6.5: Task count experiment for the IP solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ	m	v	χ^2 (9 dof)
5	13.6	2.1	95	19.9	16	7	14.8	0.106	3.67
10	29.2	3.2	100	46.3	41	23	36.0	0.219	0.910
15	45.4	4.5	100	84.0	110	86	91.7	0.309	1.20
20	61.4	5.4	98	153	233	165	191	0.395	0.573
25	77.9	6.2	98	287	595	716	416	0.655	0.948
30	93.8	7.2	100	496	1026	898	748	0.660	0.580
35	111.1	8.1	100	774	2450	3470	1520	0.809	2.97

In Figure 6.9, we show observed and inferred runtime distributions for four of the problem sets from Table 6.5. These curves are a bit more difficult to interpret than the previous sets because we are seeing a conflation of three sources of complexity: bid count, task count, and bid size. Next, will try to pull them apart.

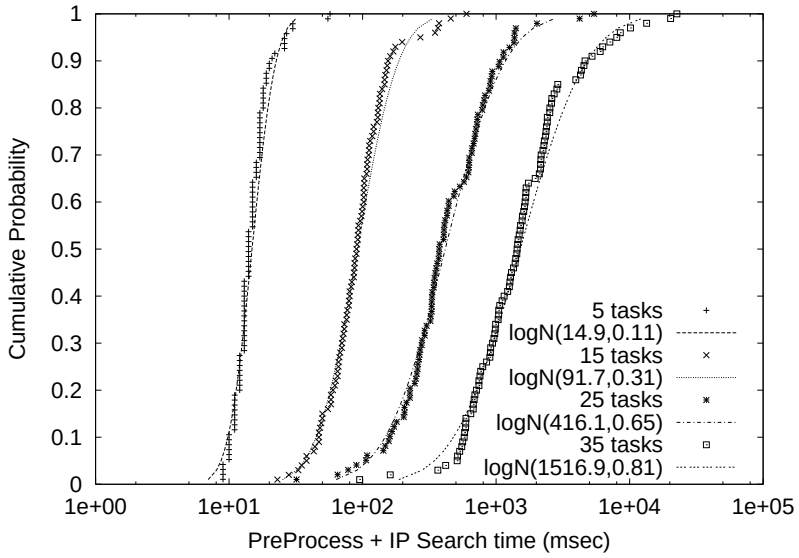

Figure 6.9: Observed and inferred runtime distributions for the IP solver across a range of task count values, with a nearly constant ratio of bids to tasks.

We now have the data to add a third dimension to our runtime estimator. This experiment, though, is a bit more complicated to analyze. In addition to varying the task count, this experiment varies the bid count and the ratio of bid size to task count. Assuming that these three parameters are enough to make predictions with the data we have so far, we ran a linear regression using the ordinary least squares method, using the 95% runtime data from all 3 experiments (this section and the previous two sections) with three different formulas:

$$\begin{aligned}
\log_2 rt &= x_1\, tc + x_2\, bc + x_3\, bs + x_4 \\
\log_2 rt &= x'_1\, tc + x'_2\, bc + x'_3 \frac{bs}{tc} + x'_4 \\
\log_2 rt &= x''_1\, tc + x''_2\, bc + x''_3 \frac{tc}{bs} + x''_4
\end{aligned}$$

The resulting coefficients are:

$$\begin{array}{lllll}
x_1 = 0.303 & x_2 = 0.0598 & x_3 = -1.168 & x_4 = 6.18 & \sigma_x = 0.612 \\
x'_1 = -0.0518 & x'_2 = 0.0565 & x'_3 = -23.7 & x'_4 = 13.9 & \sigma_{x'} = 0.337 \\
x''_1 = 0.0535 & x''_2 = 0.0493 & x''_3 = 0.846 & x''_4 = 1.98 & \sigma_{x''} = 0.264
\end{array}$$

where the σ_x, $\sigma_{x'}$, and $\sigma_{x''}$ values are the standard error values (the standard deviation of the difference between the predicted values produced by these formulas and the logs of the 95% runtime values on the individual experiment runs). Clearly the use of the ratio tc/bs produces the best result.

Transformed back into the exponential space, this translates to a 20% error for the x'' coefficients, a 26% error for the x' coefficients, and a 52% error for the x coefficients.

In Figure 6.10, we show the resulting plots for all three experiments using the x parameters derived above, along with the 95%, 50%, and 5% points for each set.

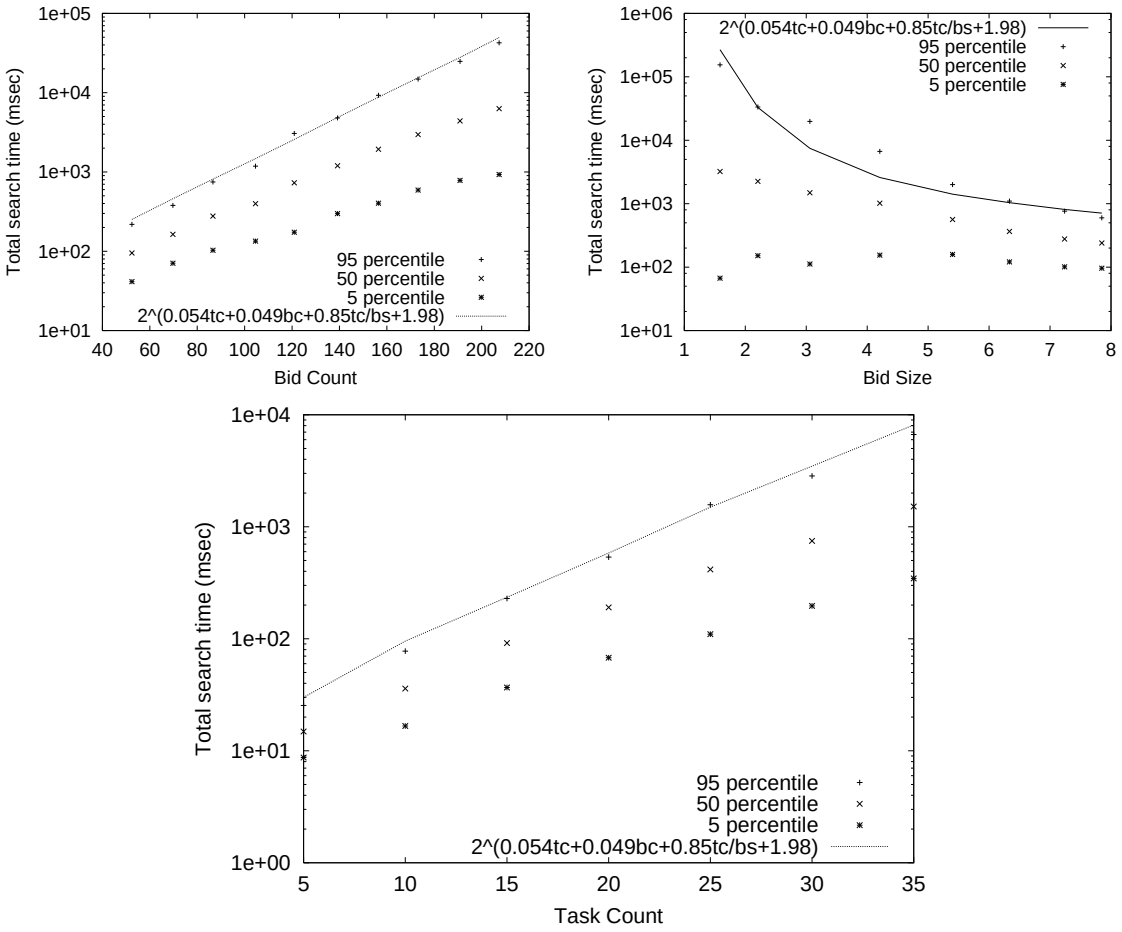

Figure 6.10: (Re-)estimating IP search time for the bid-count and bid-size experiments (top), and the task-count experiment (bottom).

The bid-count and task-count experiment data fits remarkably well, while the bid-size data seems overcorrected. That may be because the tc/bs ratio was within a relatively small range for the bid-count and task-count experiments, giving more weight to observations in that range.

One further question that can be answered with linear regression is the relative contributions of the various factors to the computed search time. This can be done by normalizing the data [Cohen, 1995]. Each variable v is transformed into $V = (v - \overline{v})/\sigma_v$, where $\overline{v}$ is the mean, and σ_v is the standard deviation of the observations of v. We compute the regression coefficients for the transformed

function

$$\frac{y-\overline{y}}{\sigma_y} = n_1\frac{tc-\overline{tc}}{\sigma_{tc}} + n_2\frac{bc-\overline{bc}}{\sigma_{bc}} + n_1\frac{w-\overline{w}}{\sigma_w}$$

where $w = tc/bs$, the ratio of task count to bid size. The resulting regression coefficients for all three experiments are $n_1 = 0.096$, $n_2 = 0.76$, and $n_3 = 0.65$. This tells us that 50% of the variability is due to bid count, 43% to the ratio of task-count to bid-count (which is roughly the mean number of bids in a solution), and the remaining 7% to the number of tasks. The resulting plots are indistinguishable from the plots in Figure 6.10.

6.2.5 Task network complexity experiment

We have examined variability due to the number of tasks, the number of bids, and the number of tasks in an auction. There are many other potential sources of variability, some of which can be measured or estimated prior to submitting an RFQ, and some of which clearly cannot (for example, the number of bids in an optimal solution). One parameter that is easy to measure is the density of precedence constraints in the task network. However, given the way we generate plans and bids, it is not easy to build experiments that control this parameter independently.

In all experiments reported so far, the "Branch Factor" (see the definition in Section 6.1.1) has been set to a value of 2.4. The example task network shown in Figure 6.1 was built with this value. In this experiment, the branch factor is varied from 1.0 to 4.0 in increments of 1.0. Because root tasks have no predecessors, and because of the elimination of redundant precedence links, the actual number of precedence relationships per task is generally much lower than the branch factor. The results are shown in Table 6.6, where the first column, labeled "Fan In," gives the actual mean number of precedence links per task. There were 35 tasks in each set of 100 problems.

Table 6.6: Task network complexity experiment for the IP solver.

Fan In	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ	m	v	χ^2 (9 dof)
0.489	112	2.01	86	176	124	96.9	103	0.318	0.774
0.876	111	5.06	100	405	1242	1624	750	0.964	0.995
1.371	109	11.9	100	1579	8450	18300	4180	0.970	0.955
1.735	109	16.0	100	1113	12800	21900	7060	1.003	0.885

As is evident from the table, the way bids are generated in our experimental setup causes bid count to go down as the number of precedence links is reduced. This is likely to cause a conflation of the network complexity factor with bid count effects, which as we have seen is a dominant factor. Figure 6.11 shows the distributions for these problem sets. It seems that bid count is not the dominant factor in these experiments after all. In this case, the low number of precedence relations has significantly reduced the number of rows in the IP problem.

The next step is to find a linear regression formula that gives a reasonable standard error value. The

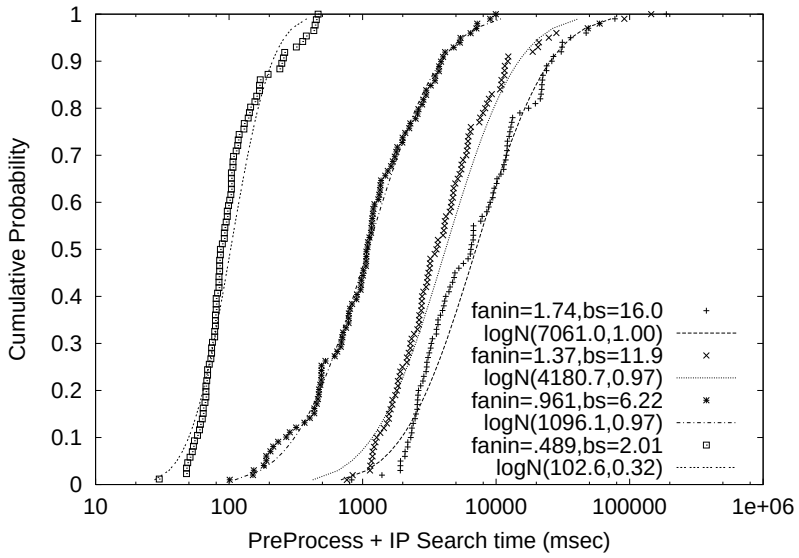

Figure 6.11: Observed and inferred runtime distributions for the IP solver across a range of task network complexity values.

first attempt was to simply add the fan-in value from the table, as

$$\log_2 rt = x_1 tc + x_2 bc + x_3 \frac{tc}{bs} + x_4 fi + x_5$$

where *fi* is the fan-in number. The resulting values are

$$x_1 = -0.0710 \quad x_2 = 0.0467 \quad x_3 = 0.606 \quad x_4 = 13.04 \quad x_5 = -8.751 \quad \sigma_x = 1.562$$

This is a very poor fit; the standard error of 1.562 translates to being off by a factor of about 3. Rather than work through a long list of alternative formulas, the alternative is to create a formula with a wide variety of factors and let the ols procedure tell us which ones are useful. Table 6.7 gives the factors in the left column, and results for the "raw" numbers, for a normalized version titled "Norm 1" (which of course omits the "1" factor) and for a normalized version titled "Norm 2" that omits the last 3 factors.

This gives a good fit, at the expense of some extra complexity. In the normalized versions, the fan-in factors dominate the formula. They are negative, as we would expect, given that precedence relationships translate directly to constraints in the IP formula.

A weakness of this data is that very few of the experiment sets show significant variation in the fan-in factor. The bid-count and bid-size experiments both used the same 20-task problem sets, and so there was no variation in the fan-in number for more than half of the experiments. The task-count experiment showed some variation, and the task-complexity experiment showed much variation over very few problems. To improve on this empirical formula, a much larger set of experiments needs to be designed that will spread the variations across all the factors.

Table 6.7: IP regression factors and results for raw and normalized data.

Factor	Raw	Norm 1	Norm 2
task count tc	-2.83	-6.799	-7.031
bid count bc	0.0509	0.734	0.777
bid size bs	-0.613	-0.599	-0.985
tc/bs	0.433	0.489	0.157
fan-in fi	-483	-29.5	-4.02
fi/tc	-3050	-27.9	-1.43
$fi \cdot tc$	3.75	13.0	8.83
$fi/\log_2 tc$	2260	35.1	
$\log_2 fi/tc$	-82.3	-0.206	
$\log_2 fi \cdot tc$	0.531	1.136	
1	105.7		
σ	0.114	0.0118	0.0291

6.3 Characterizing the Simulated Annealing solver

Since the Simulated Annealing solver is a stochastic algorithm, we are interested in two types of performance data. The first is the same type of data that we saw for the IP solver in the last section: performance across a range of task counts, bid counts, and bid sizes. The second is a more detailed look at the variability observed on individual problems.

For these experiments, we use the same sets of 100 problems that were used for the IP experiments in Sections 6.2.2 through 6.2.4. We did not run the task network complexity experiment with the SA solver. For each problem, we ran the IP solver to find an optimal solution, or to determine that the problem could not be solved. Then for each "solvable" problem, we ran the SA solver 20 times with different random number sequences, and captured time and node count data when the solver first found solutions with evaluations within 5% of the optimal value, within 1% of optimal, and equal to the optimal value found by the IP solver. In the remainder of this section, we'll call these "5% solutions," "1% solutions," and "optimal solutions," respectively. We then averaged the values for successful attempts on each problem, and we show here the statistics for those average values across the various problem sets.

For the experiments in Sections 6.3.1 through 6.3.3, the SA solver was tuned with the following parameters, although a few exceptions are noted in the text:

Patience factor: $pf = 80$.

Scale factor: $sf = 0.84 \cdot pf \cdot (m + n)$ where m is the number of tasks and n is the number of bids. For 20 tasks, 100 bids, this gives $sf = 806$.

Temperature profile: Initial Temperature $T_0 = 0.35$, annealing rate $z = 0.997/(0.15 \cdot sf)$.

Queue length: $200 \cdot sf$.

Stopping condition: If the last improvement was observed in node n, then search stops at node $n_s = max(n + pf \cdot sf, 1.5n)$.

Restarts: 100 maximum.

Random restarts: Scale factor was multiplied by an exponentially-distributed random value with a mean of 5.0 on each restart.

Bid selector: Random selection among the "usable" bids - those that are not a member of the conflict sets of any bids in the current partial solution.

Infeasible nodes: Dropped.

Uniqueness testing: Non-unique nodes are dropped.

Heuristic function: Cost of non-covered tasks was twice the maximum cost per task among all bids.

6.3.1 Bid count experiment

Table 6.8 shows SA performance results on a subset of the problems for which IP results are reported in Table 6.3. The long runtimes precluded us from running the full 20-search/problem SA experiment on all 10 of the problem sets in the bid count experiment. In addition, the 156-bid set is significantly truncated, and we assume that the level of truncation in the larger sets would have been even greater.

Table 6.8: Bid count experiment for the SA solver

Bid Count	Failure Rate (%)	Node Time (ms)	5% Solution Mean (ms)	5% Solution Median (ms)	5% Solution σ	Optimal Solution Mean (ms)	Optimal Solution Median (ms)	Optimal Solution σ
51.7	0	0.41	508	352	572	3660	787	11,600
86.5	7	0.43	4490	1130	11,200	24,500	3840	48,200
121.0	16	0.46	13,000	2910	39,000	52,600	12,000	78,600
156.4	31	0.46	37,400	7850	72,900	111,000	36,300	117,000

In Table 6.8, the "Failure Rate" column gives the proportion of attempts in which the SA solver failed to find optimal solutions to solvable problems (problems solved by the IP solver). The "Node Time" column reports the mean time per search node generated. These times grow as problem size increases because of the cost of maintaining the sorted search queue. Then we give Mean, Median, and Standard Deviation data for the time in milliseconds required to find a 5% solution and an optimal solution. We give both Mean and Median here because they are substantially different, due to the skewed distribution.

In Figure 6.12 we see the observed runtime distributions for these four sets. We also show the "closest" lognormal curves, as determined by computing maximum likelihood estimators, for comparison purposes. Whereas the lognormal curves were a good match for the characteristics of the

IP runtime distributions, they are clearly not a good match here. These data are not well-behaved, and for the larger problems the high end is truncated. This happens because we have placed limits on the ability of SA to search, including limiting the queue size and the number of restarts, and by running the annealing temperature down to a level where there is little randomness (typically the end temperature is between 10^{-2} and 10^{-3}) at the end of each restart attempt.

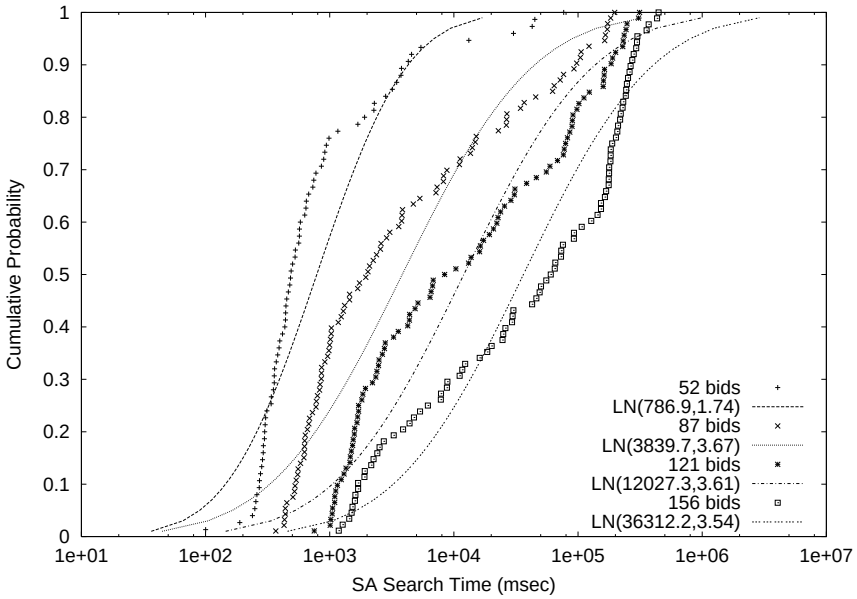

Figure 6.12: Observed and lognormal runtime distributions for SA search to find optimal solutions across a range of bid count values.

6.3.2 Bid size experiment

In Table 6.9 we see a summary of results from running the SA solver on 8 sets of 100 problems in which the bid size is varied between sets. These are the same problem sets used for the IP bid-size experiments as summarized in Table 6.4. For the 1.61 tasks/bid set, the mean length of the optimal solutions (number of bids in the solution) was 14.2. For solutions in that set of length 10 or less, the SA solver found optimal solutions 95% of the time. For solutions longer than 14 bids, SA found no optimal solutions, although there were no problems for which at least a 5% solution was not found on at least 3 of the 20 attempts.

Figure 6.13 shows observed runtime distributions of four of the problem sets from Table 6.9. We also show "inferred" lognormal distribution curves, although they are clearly not a good match. In particular, the failure rate for the 3.04 tasks/bid case was over 50%. Since we arbitrarily stopped the SA solver after 100 retries, it is reasonable to expect that the solution times for the unsolved problems would all have been larger than the observed times for solved problems. Therefore the 3.04 tasks/bid curve should probably be plotted only up to the 43% probability coordinate. This would put the median time value much higher. Other than the poorly-defined shapes of these distributions, the striking factor is that they have a very different character from the distributions for the IP solver

Table 6.9: Bid size experiment for the SA solver

Bid Size	Failure Rate (%)	Node Time (ms)	5% Solution Mean (ms)	5% Solution Median (ms)	5% Solution σ	Optimal Solution Mean (ms)	Optimal Solution Median (ms)	Optimal Solution σ
1.61	93	0.43	113,000	75,200	68,900	126,000	79,600	103,000
2.20	74	0.42	61,800	29,500	56,400	127,000	90,600	69,600
3.04	57	0.43	34,200	12,600	46,100	116,000	73,300	71,900
4.22	28	0.45	18,000	5740	28,100	70,900	22,800	77,500
5.42	12	0.44	9710	2320	24,200	42,900	10,600	64,700
6.40	9	0.43	4270	1430	10,100	32,300	6620	54,500
7.30	7	0.43	4490	1130	11,200	24,500	3840	48,200
7.90	6	0.43	3070	1080	8490	19,400	3250	40,800

on the same problems (see Figure 6.7).

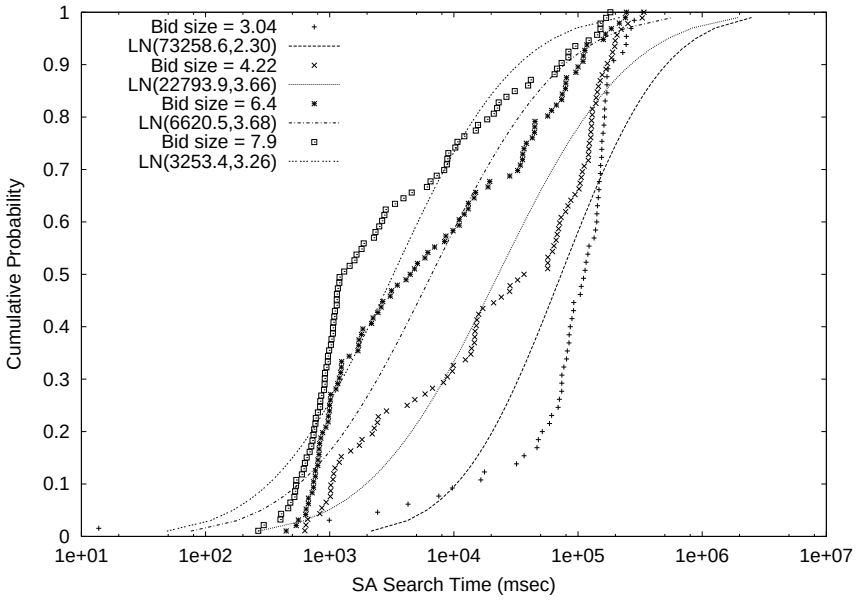

Figure 6.13: Observed and lognormal runtime distributions for SA search to find optimal solutions across a range of bid sizes. All problems have 20 tasks, 100 bids.

6.3.3 Task count experiment

In Table 6.10 we report data on SA performance as task count varies. This is the same set of problems that was reported in Table 6.5 on IP performance, so bid count and bid size data can be read from that table.

Figure 6.14 shows graphically the observed distributions for 4 of the problem sets shown in Table 6.10. Each individual data point represents the average time required to find an optimal solution using the SA solver. Only the successful trials were included, so there is likely some truncation at the upper

Table 6.10: Task count experiment for the SA solver

Task Count	Failure Rate (%)	Node Time (ms)	5% Solution Mean (ms)	5% Solution Median (ms)	5% Solution σ	Optimal Solution Mean (ms)	Optimal Solution Median (ms)	Optimal Solution σ
5	0	0.23	5	4	4	7	5	5
10	0	0.31	92	75	82	250	116	915
15	2	0.36	392	282	442	2270	652	7610
20	4	0.42	2000	857	4400	12,600	2550	29,500
25	11	0.48	9420	2420	20,300	26,500	6610	41,300
30	19	0.54	19,600	4270	45,600	50,800	11,300	93,800
35	23	0.60	35,700	8220	72,500	97,900	27,900	126,000

end of the range on the larger problems. We also show the closest lognormal curves, but clearly the data is not as well-behaved as the IP data, and doesn't fit a lognormal distribution for the larger problems.

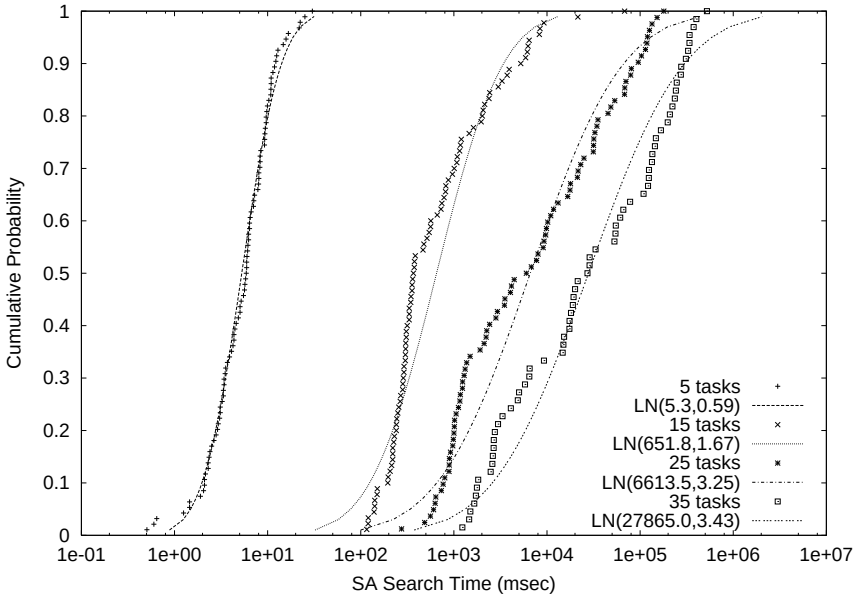

Figure 6.14: Observed and lognormal runtime distributions for SA search to find optimal solutions across a range of task count values.

In a time-limited negotiation situation, we might be willing to settle for a solution that is less than optimal. An obvious question is how much time we gain by doing so. Figure 6.15 shows the detailed aggregate behavior for two of the problem sets in Table 6.10. Again, each point represents the mean time required to find solutions to the indicated level of accuracy, for trials in which solutions were found. There is some degree of truncation in the data for finding optimal solutions in the 35-task case: the overall failure rate was 22.5%, and there were 7 solvable problems for which the SA solver never found optimal solutions. The failure rates for the 5% and 1% solutions were 2.6%

and 13.3% respectively, and there was one problem for which no 1% solution was found. For the 20-task problems, there were no failures to find the 5% solution, a 2.1% failure rate for finding 1% solutions, and a 3.8% failure rate for finding optimal solutions.

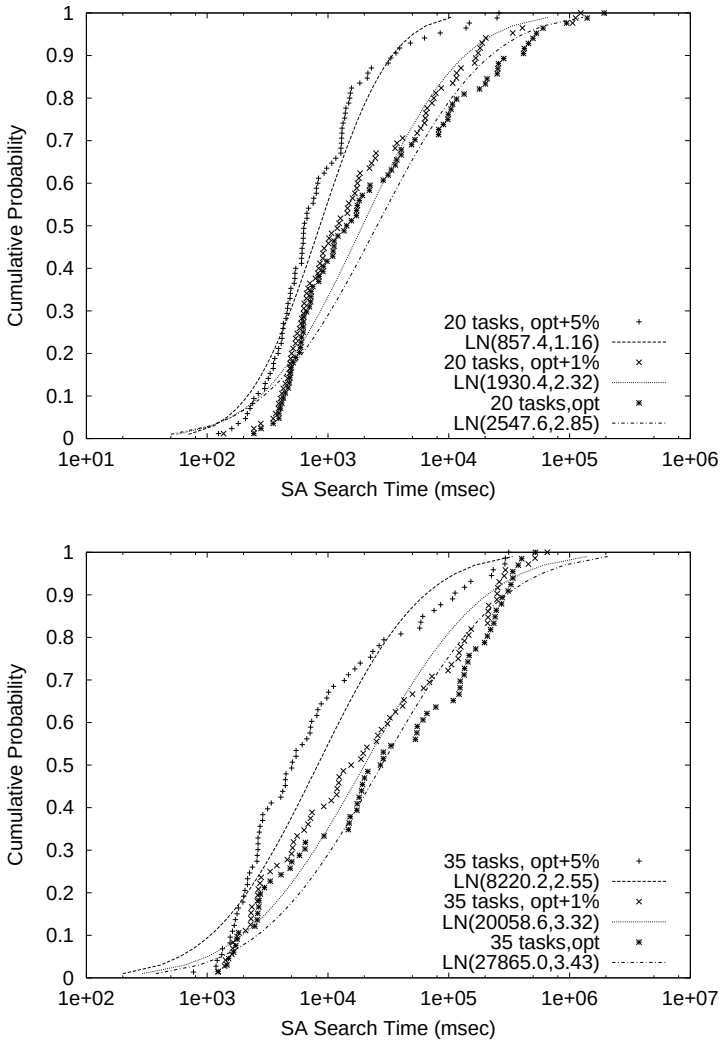

Figure 6.15: Observed distributions for finding 5%, 1%, and optimal solutions using Simulated Annealing, on problems having 20 tasks (top, row 4 of Table 6.10) 35 tasks (bottom, row 7 of Table 6.10).

6.3.4 Detailed analysis of the SA solver

As noted earlier, the results shown for the SA search in the three experimental series described so far are mean values from relatively small samples (20 observations or fewer), with unknown random distributions. They may or may not be representative, depending on the shape and variability of the underlying distributions. Since the computing resources needed to find the underlying distributions on each of the problems would be prohibitive, we have undertaken a smaller study to attempt to

characterize the SA search on a representative set of problems. For this set, we have used the same problem generator to produce 5 problems, each having 20 tasks, up to 90 bids[3], and an average bid size of 5.7. For each problem, we used the IP solver to find an optimal solution, and then ran the SA solver 200 times, measuring the time and node count when the optimal solution was found. We also measured the time and node count at the point where the first solution within 5% of optimal and within 1% of optimal was found. The search was terminated and restarted after a fixed number of iterations without improvement, and up to 100 restarts were permitted, unless an optimal solution was found earlier.

Although we collected both time and node-count data, and in the earlier series we reported time for comparison with the IP solver, for this series we will report node-count data in order to factor out the small variability due to machine loading, garbage collection, etc. For comparison purposes, the average per-node performance for 20-task problems on the machine we used for all experiments is 0.418 ms/node.

6.3.4.1 Performance distribution on individual problems

Table 6.11 summarizes the data for this problem set. The "IP Time" column gives the sum of preprocessing and IP runtime, in seconds. "# of bids in optimal solution" is the number of bids in the optimal solution found (it is possible, but unlikely, that other optimal solutions exist with fewer bids). "# of opt. soln's found by SA" gives the number of trials out of 200 in which the SA solver actually found an optimal solution. "Mean node count" and "σ (node count)" give the sample mean and standard deviation for the number of nodes search by SA for runs in which an optimal solution was found.

Table 6.11: SA Performance Experiment: 200 trials/problem, each problem had 20 tasks and up to 90 bids

Problem	Bid count	Bid size	IP time (sec)	# of bids in optimal solution	# of opt. soln's found by SA	Mean node count	σ node count
1	80	5.8	1.07	7	200	79100	71800
2	83	7.63	2.30	9	200	58800	59200
3	71	6.63	0.42	5	200	6240	5900
4	75	4.33	0.36	11	69	297000	187000
5	79	4.20	0.30	10	139	277000	187000

Several observations are evident from this data:

- For only three of the five problems was the SA search able to find an optimal solution every time. Since all of these problems are known to be solvable (the IP solver found optimal solutions for all of them), the actual mean node counts for the other three problems are larger,

[3]The bid generator made 90 attempts to generate bids, some of which were unsuccessful

potentially much larger, than the reported sample mean node count values. Once we have an approximation of the underlying distribution, we can get a better estimate of the true means.

- For one of the problems, #3, the search effort for the SA solver compared favorably with the IP search effort. For the others it was much worse.
- The correlation between average bid size and search effort is weak.
- There is a strong correlation between the number of bids in the optimal solution found by IP and the difficulty our SA search engine has in finding an optimum. This makes some sense, since the SA search is a local search, and it must traverse a longer path to find a solution with a greater number of bids. Unfortunately this problem metric is not known until after an optimal solution is known.

Following the method outlined by Hoos and Stützle [Hoos and Stützle, 1998], we have attempted to further characterize the variability of our SA algorithm. We start by hypothesizing a distribution function. It must have an exponential "tail", but the exponential distribution $F(x) = 1 - e^{x/\beta}$ by itself does not have the desired properties. This is because its *hazard function* $h(x) = \frac{1}{\beta}$ is constant. In our application, the hazard function is the probability that node x will produce a solution given that nodes $1 \ldots x-1$ did not. We expect in general that the probability of finding a solution will be an increasing function of x, since the search expands nodes by adding single bids, and nodes with more bids are more likely to contain solutions than nodes with fewer bids. The Weibull distribution $F(x) = 1 - e^{-(x/\beta)^\alpha}$ is a good candidate, because its hazard function, $h(x) = \alpha\beta^{-\alpha}x^{\alpha-1}$, has the desired property that with $\alpha > 0$ it is an increasing function of x. Also, the Weibull distribution is equivalent to the exponential distribution if $\alpha = 1$.

To confirm the hypothesis and to find the associated parameters, the analysis process is as follows:

1. Use the raw node-count data to generate estimates for the parameters of a Weibull(α, β) distribution $F(x) = 1 - e^{-(x/\beta)^\alpha}$. We start with α (shape) and β (scale) values computed as *maximum likelihood estimators* (MLE) for the distribution as described in [Law and Kelton, 1991].
2. The MLE computation depends on having samples from the full distribution, while for some problems we have a biased sample; we know the problem can be solved (the tuning parameters cut off search at about 10^6 nodes), so failure to solve it means that the effort would have been larger than the maximum number of nodes generated. Because of this, the sample for the optimal value of problem #4 as shown in Table 6.11 would contain data from the first $\frac{78}{200}$ of the underlying distribution. The missing 122 samples would presumably all have values larger than any of the values seen.
3. Generate a histogram from the sample data. Pick a bin size (we used a bin size of approximately 25 for this analysis), compute the number of bins as $binCount = SAsuccess/binSize$ and divide the Weibull(α, β) density function $f(x) = \alpha\beta^{-\alpha}x^{\alpha-1}e^{-(x/\beta)^\alpha}$ into $binCount$ equal areas (this is

an *equiprobable* approach). This gives the boundaries of the bins. For the incomplete samples, use the left-hand *observations/trials* fraction of the density function. Allocate the run-length observations to the resulting bins.

4. Compare the resulting histogram with the "ideal" histogram we computed from the Weibull(α, β) density function. Compute the χ^2 metric to determine degree-of-fit.

5. For the truncated samples (problems 4 and 5), adjust the α and β values manually to minimize the χ^2 value.

Figure 6.16 gives two examples of the density functions and the accompanying histograms for the node count required to find an optimal solution and a solution within 1% of optimal in problem #1. The χ^2 values for these two cases give $\chi^2 = 10.1$ and $\chi^2 = 18.0$ with 13 degrees of freedom. This yields less than 25% probability in the (optimal + 1%) case, and less than 75% probability in the optimal case, that the data does not fit the hypothesized distribution. This is a reasonably strong confirmation that the Weibull distribution is a good fit.

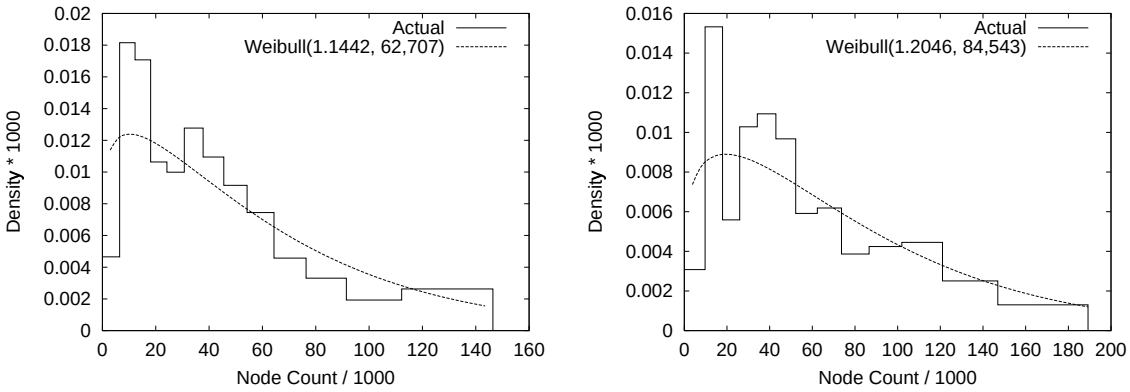

Figure 6.16: Weibull density function and SA node-count histograms for problem #1, for solutions within 1% of optimal (left) and optimal (right)

Table 6.12 summarizes the distribution data for the five problems. For each problem, we show the mean and standard deviation for the inferred Weibull distributions for the search effort required to find an optimal solution, and to find solutions within 1% and within 5% of optimum. Note that the inferred means for problems #4 and #5 are much larger than the sample means given in Table 6.11.

Comparing problem #2 with problem #3 in Table 6.12, we can see that finding an optimal solution is an order of magnitude easier for problem #3, but finding a solution within 5% of optimal is about the same for both problems. This could be because there are several solutions to both problems, some of which have few bids, that are within 5% of optimal, while problem #2 has fewer and/or longer solutions that are optimal (although in no case was more than one optimal solution ever found).

Another, possibly more intuitive way to look at this data, is to look at the probability distributions rather than density. Figure 6.17 shows the Weibull probability distributions along with the actual

Table 6.12: Inferred node count statistics for problems 1, 2, 3, and 5

Problem	Optimum		Within 1%		Within 5%	
	Mean	σ	Mean	σ	Mean	σ
1	79400	66300	59800	57700	18300	14800
2	58800	60900	11900	14200	1700	1750
3	6240	5900	2070	1730	1650	1570
4	1170000	1060000	420000	400000	64600	61200
5	499000	454000	114000	114800	23900	25300

node-count observations for problem #1 (top), and for problem #2 (bottom).

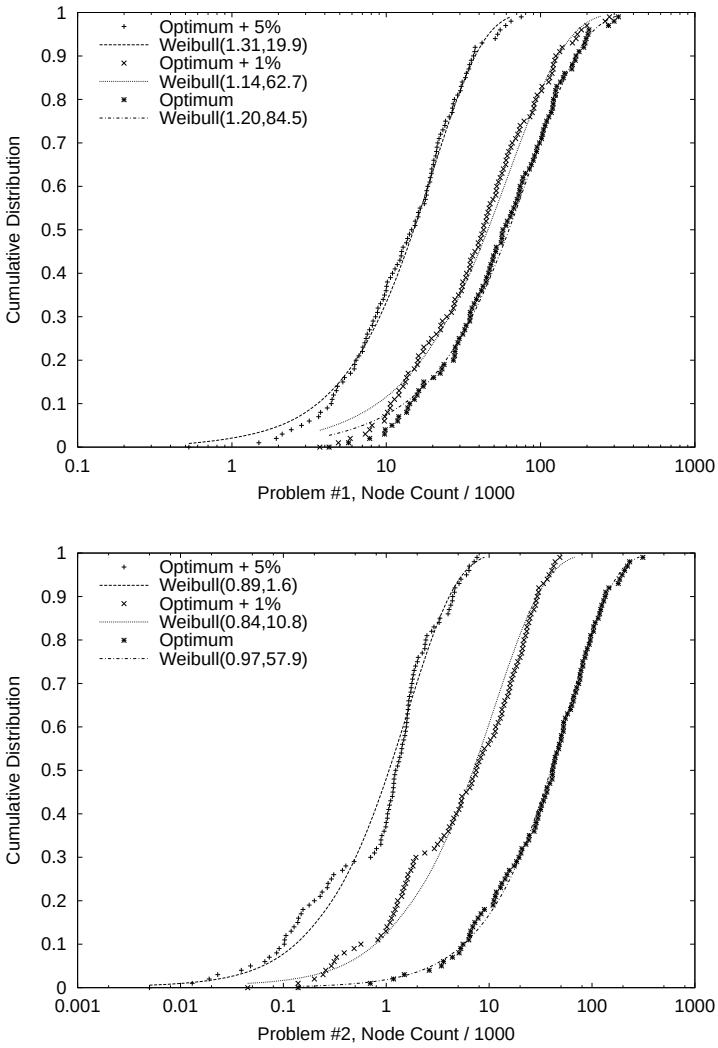

Figure 6.17: SA node-count observations and inferred Weibull distributions, problem #1 (top), problem #2 (bottom).

One interesting feature of these two plots is that the optimal+5% solution is significantly easier to

find for problem #2 than it is for problem #1, while the difficulty of finding an optimal solution for for problem #2 is very similar to the difficulty for problem #1. Also, in the plot for problem #2, we can see that the density function for the optimal+5% solution exhibits multiple peaks toward the low end. This appears when most observations involve very few restarts of the SA algorithm. There is a peak for observations without restart, another peak for observations that require a single restart, etc.

A second interesting feature of the bottom plot in Figure 6.17 is that the Weibull α parameter is less than one. This means that the hazard function $h(x) = \alpha\beta^{-\alpha}x^{\alpha-1}$ is a decreasing function of the node count, especially on the 1% and 5% curves, and not an increasing function as it is in the top plot. The implication is that finding a solution actually gets harder as the search progresses. This could be an artifact resulting from the search "getting lucky" on the first restart in a significant fraction of the trials.

It is also interesting to see the truncated distributions for problems #4 and #5 in the same plot format, as shown in Figure 6.18. Because these represent more difficult problems with typically many more restarts, we see very good correspondence between the sample data and inferred distributions. The slight tail-off for the last few observations of problem #5 is likely because our assumption of truncation is an approximation.

Another question we might ask is whether the high variability of search difficulty among these 5 problems arises from the task networks, or from the particular sets of bids that are submitted. Because of the nature of the bidding process, it is easy to generate different random bid sets for each task network, but it is not possible to use the same bid set for a different task network. In Figure 6.19, we see the data for finding the optimal solutions with SA for the task networks from problems #2 and #5, with 5 different bid sets each. One interesting observation here is that there is a single outlier in both cases. In the first case it happened to be in the set we used for the original study, and in the second case it was in bid set #5. Another observation is that, at least within this small sample, both the task network and the specific bid set seem to contribute to the variability among problems of the same "size".

6.3.4.2 Tuning the SA solver

The Simulated Annealing search engine has many tuning parameters, many of which are automatically adjusted according to problem size. It would be tempting to attempt to optimize each of them, although the time required would be prohibitive. It is also very likely that the optimal values for one set of problems would not be optimal for any other set.

As an example, we show in Figure 6.20 the results of varying the "patience factor", or the number of iterations without improvement that will trigger a restart. In this series, we show how performance varies with three different combinations of patience factor and maximum allowed restart-count. For the "base" case, we set the patience factor to allow 670 nodes without improvement before a restart, and up to 60 restarts. We've then added two cases in which we doubled and halved the

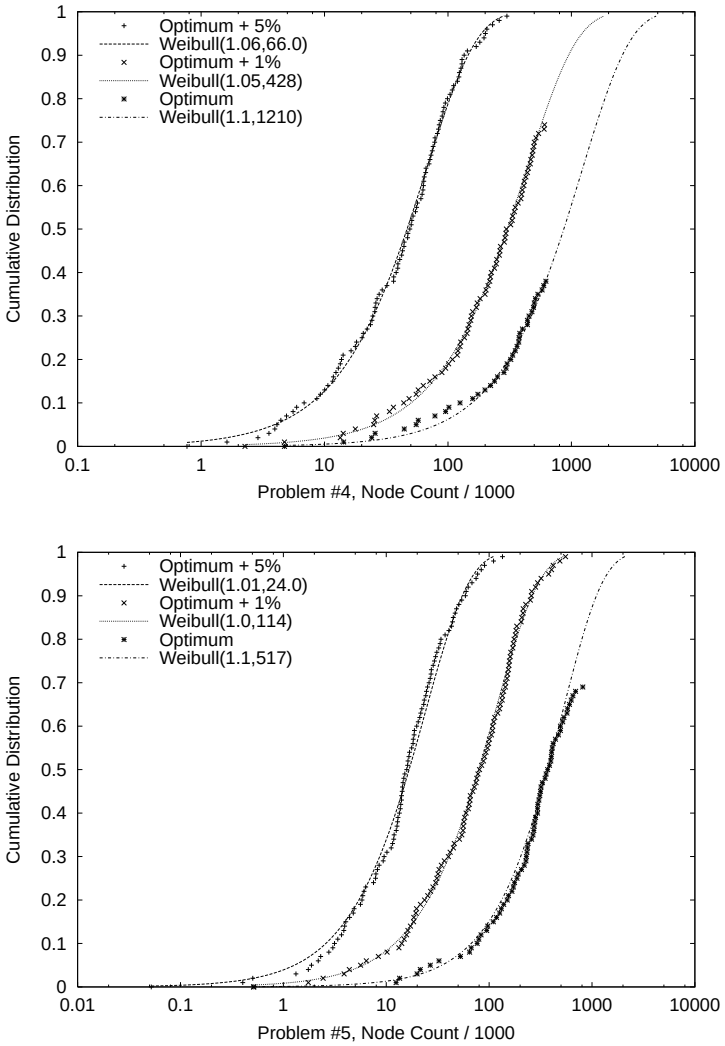

Figure 6.18: SA node-count observations and inferred Weibull distributions, problems #4 (top) and #5 (bottom).

patience factor, with the allowable restart count also being halved and doubled to allow about the same amount of search effort. From the top plot in Figure 6.20, we might conclude that the higher patience factor gives better results, at least on problem #1. However, in the bottom plot, we see that the smaller patience factor (for which twice as many restarts were allowed) produced about the same average performance, but the optimal value was found in all 200 trials.

Because of results like this, all the earlier experimental results were set up to vary the patience factor at each restart. This is done by selecting an exponentially-distributed random number to re-scale the patience factor, the maximum queue length, and the annealing schedule on each restart. We can see by comparing Figure 6.20 with Table 6.12 that this produces significantly worse results for problem #1, but much better results on problem #2. It is also the only setting we have found that finds solutions more than 10% of the time for problem #4.

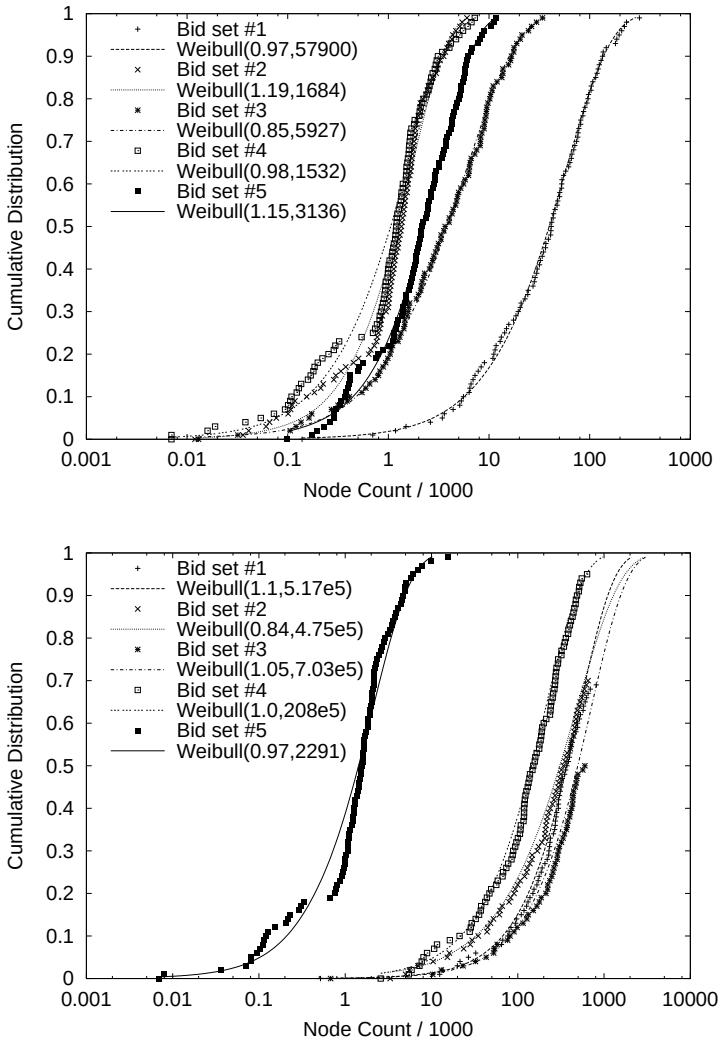

Figure 6.19: SA node-count observations and inferred distributions for finding optimal solutions, task network #2 (top) and task network #5 (bottom), with 5 different bid sets.

6.3.5 Problem-by-problem comparisons of IP and SA solvers

The previous tables and figures describe aggregate performance of the two solvers, and they allow us to confirm that, for an arbitrary problem in a time-limited situation, we have a higher probability of finding an optimal solution using the IP solver than using the SA solver. This is not a surprise. However, this does not tell us that the IP solver is *always* faster.

In Figure 6.21 we can see the relative performance of the IP and SA solvers on each of the (solvable) individual problems for the 20-task set from Section 6.3.3, and in Figure 6.22 for the 35-task set, from the task-count experiment series. In each figure, the top plot compares the IP time with the time required for SA to find a solution within 5% of optimal, and the bottom plot compares IP time with time for SA to find optimal solutions. For problems below the $x = y$ lines, SA was faster; above

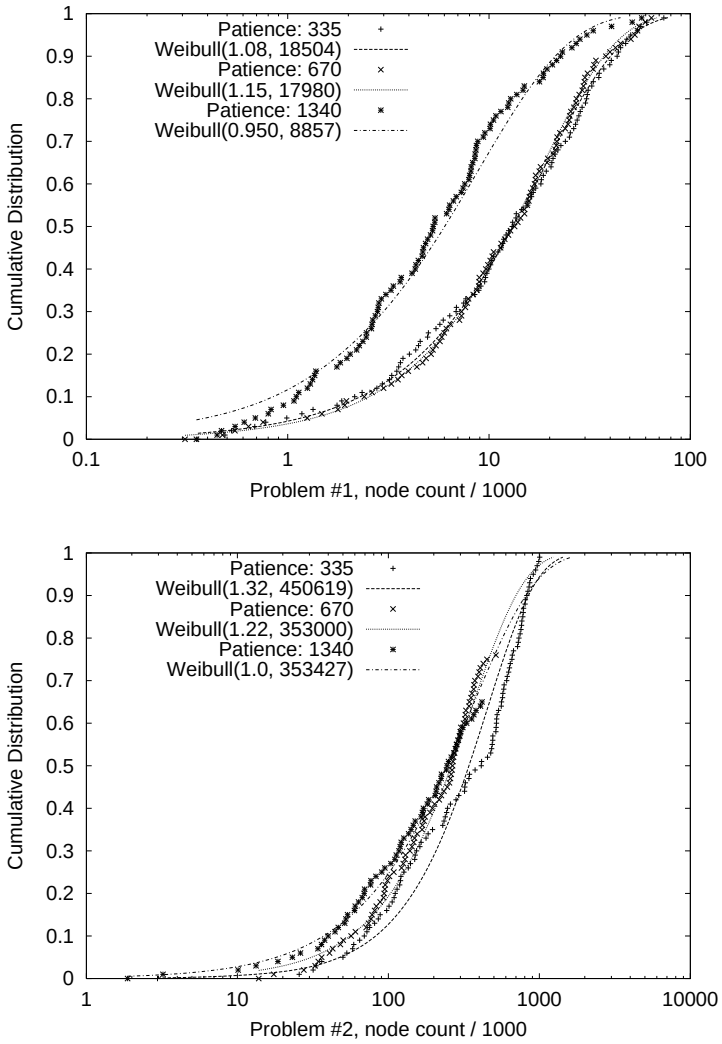

Figure 6.20: Varying the patience factor, problem #1 (top), and problem #2 (bottom)

the line, IP was faster.

There are two interesting features in these plots:

1. The IP search time is not correlated with the SA search time.
2. For larger problems and for lower demands on optimality, SA outperforms IP in many cases, but seldom (in this data set) by more than a factor of 2.

Figure 6.23 shows similar data for the 2.2 bids/task problem set from the bid-size experiment. In this case the difference between the two is even more striking. We can easily see the large variability of the IP solution time, and some of the SA solutions were more than an order of magnitude faster than the IP solution for the same problems.

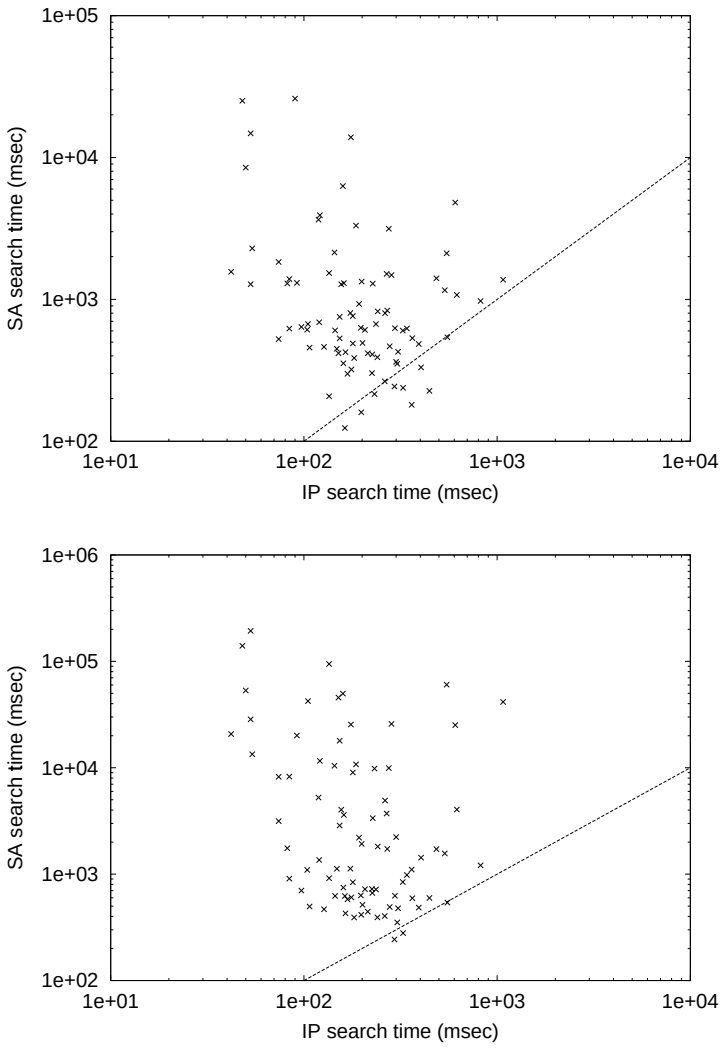

Figure 6.21: Problem-by-problem comparison between IP solution time and time required by SA to find optimal+5% solutions (top), and optimal solutions (bottom), 20-task problem set.

We have not attempted to develop runtime predictors for the SA solver as we did in Section 6.2 for the IP solver. This is partly because we have not found probability distributions that are a good match for the runtime characteristics of the SA solver, but more importantly it is because there are many tuning parameters for the SA solver, and these data represent just one possible combination of settings of those parameters. If the goal is to use the SA solver to "hedge" the runtime of the IP solver, one would likely use very different parameter settings, and accept a higher failure rate. For a final comparison, we show in Figure 6.24 the result of changing two of the basic SA parameter settings. This plot is a variation of Figure 6.21 (top), in which the "patience factor" is set to 80, and the "beam-width" is set to 200. As discussed earlier, the patience factor controls the number of iterations without finding an improved node that will trigger termination of a restart cycle, and the beam width is a limit on the size of the search queue. Both factors are scaled by problem size;

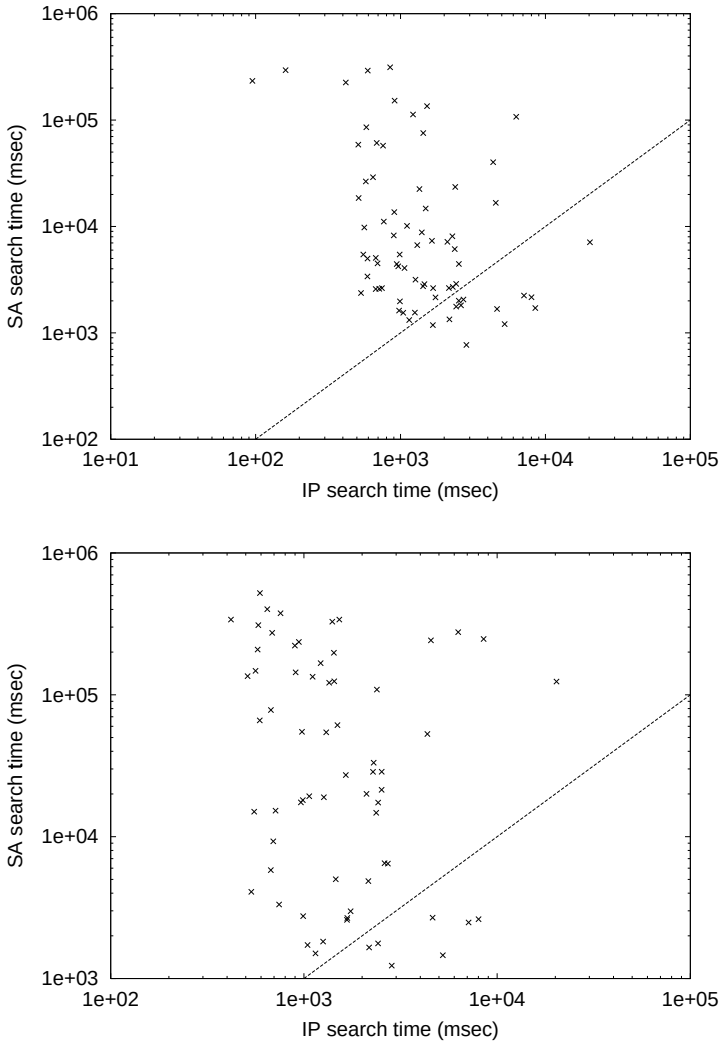

Figure 6.22: Problem-by-problem comparison between IP solution time and time required by SA to find optimal+5% (top), and optimal solutions (bottom), 35-task problem set.

these are the settings used in all the SA runs described so far. The plot in Figure 6.24 shows the same problem set, the same IP data, but we have set the patience factor to 40 and the beam width to 50. The number of problems for which SA finds its "optimal+5%" solution faster than IP finds its solution is nearly doubled. However, the failure rate for the SA solver under these conditions jumps from 3.8% in the original set to nearly 15% in the "faster" set.

6.4 Characterizing the A* and Iterative Deepening A* solvers

In this section we examine the performance of the bidtree-based A* and IDA* solvers that were described in Section 5.3. We are interested in understanding in detail the impact of the bidtree-ordering issue raised in Section 5.3.3. We also want to examine the scalability and predictability of

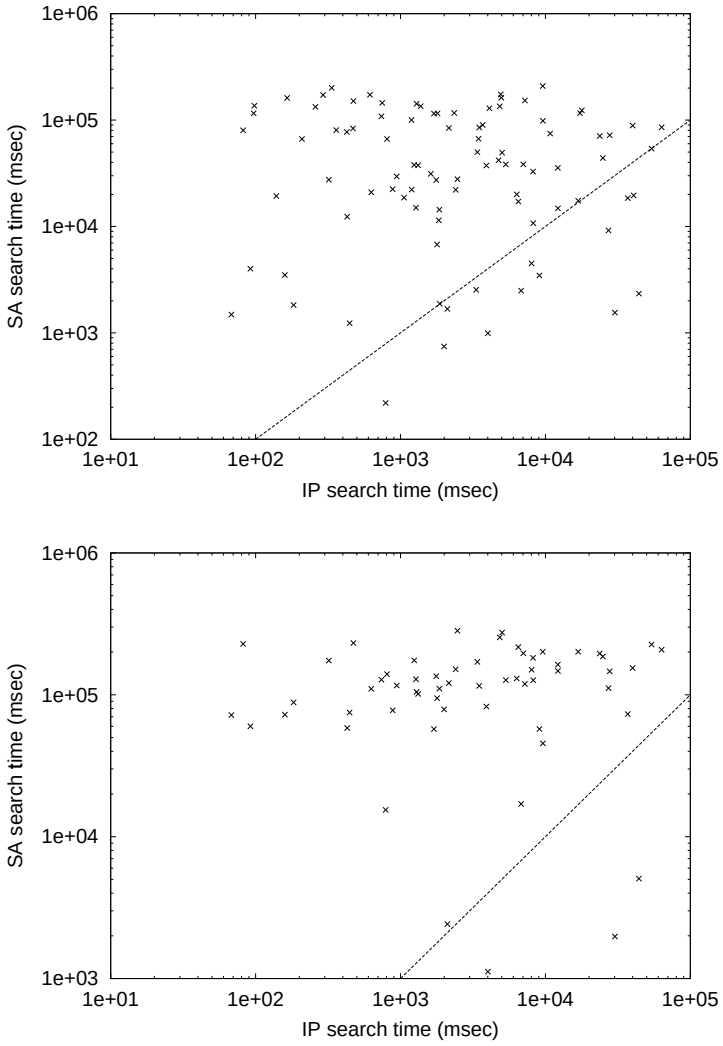

Figure 6.23: Problem-by-problem comparison between IP solution time and time for SA to find optimal+5% solutions (top), and optimal solutions (bottom), 20 tasks, 2.2 tasks/bid.

these algorithms, and to compare them with the IP solver.

We have not attempted to reproduce Sandholm's original experiments [Sandholm, 2002], although that would certainly be desirable. That would require access to his tools, code, and problem instances[4].

6.4.1 Measuring the impact of bidtree order

In Figure 6.25 we see search-performance curves for the IP solver, and for A* and IDA* using a bidtree sorted by increasing bid count (labeled A*(inc) and IDA*(inc)), and for A* and IDA* using

[4]In a personal communication, he indicated that these materials are not currently in good condition to share, and so we must defer that exercise to a future date.

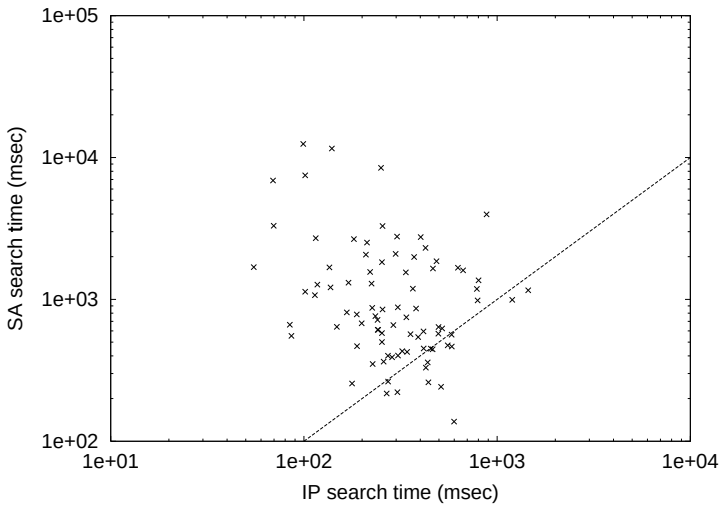

Figure 6.24: Problem-by-problem comparison between IP time and time for SA to find optimal+5% solutions, 20-tasks, SA parameters: pf=40, bw=50.

a bidtree sorted by decreasing bid count (labeled A*(dec) and IDA*(dec)). All five experiments used the same series of 100 randomly-generated problems. Clearly, the A* and IDA* perform much better, with lower variability, if the bid tree is constructed starting with the task having the largest number of bids[5].

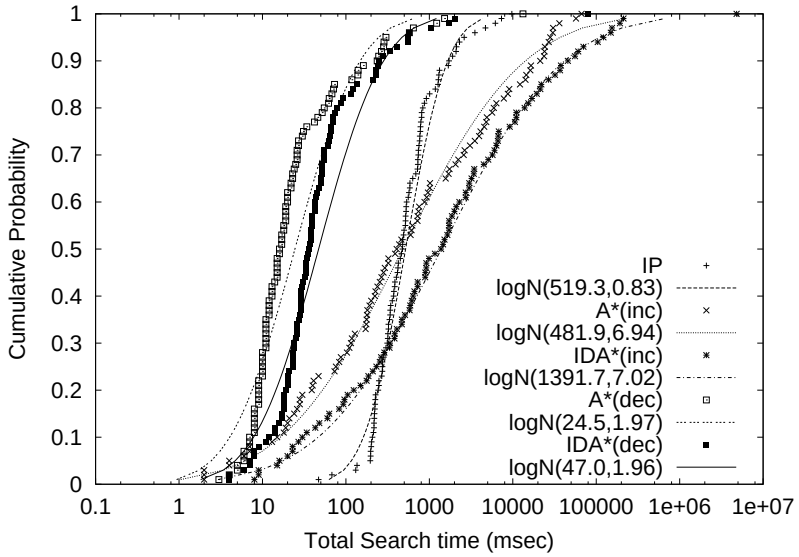

Figure 6.25: Performance of A*, IDA*, and IP search (35 tasks, 111 bids) with bid tree sorted by increasing bid count (A*(inc) and IDA*(inc)), and by decreasing bid count(A*(dec) and IDA*(dec)).

Now we see graphically the impact of the lower branching factor induced by the decreasing sort of

[5]In the interest of full disclosure, it should be noted that the A* search failed because of memory overflow (more than 30 000 nodes in the queue) for 6 of the 100 problems with the increasing sort, and for 1 of the 100 problems with the decreasing sort.

the bidtree. We can use an approximation known as the *effective branching factor* $\beta^\star$ to quantify the difference. If N is the total number of nodes generated and d is the depth of the solution found, then the effective branching factor $\beta^\star$ is the branching factor that a uniform tree of the depth d would have in order to contain N nodes. We can approximate this given that we know the depth of the branch of the tree that includes the solution. This is simply one more than the number of bids in the solution, since the root node contains no bids, and each expansion adds a bid to a partial solution. The value of $\beta^\star$ can be approximated as

$$\widetilde{\beta^\star} = N^{1/(d-1)}$$

For the two problem sets illustrated in Figure 6.25, the mean effective branch factors across the full set of 100 problems are

$$\begin{aligned} \widetilde{\beta^\star}_{\text{inc}} &= 2.8 \\ \widetilde{\beta^\star}_{\text{dec}} &= 1.7 \end{aligned}$$

where "inc" refers to the increasing bid-count sort, and "dec" refers to the decreasing bid-count sort.

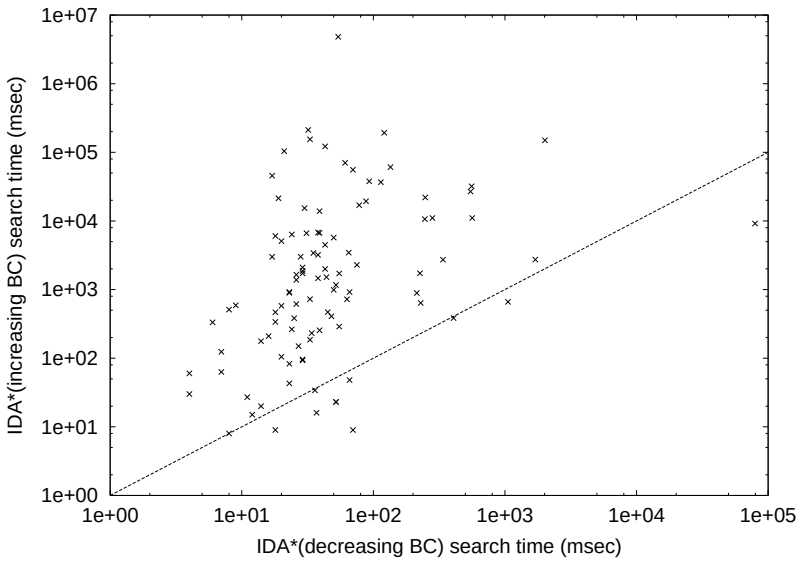

Figure 6.26: Scatter plot showing relative performance of IDA* using increasing and decreasing sort on a set of 100 randomly-generated problems.

Figure 6.26 is a scatter plot showing the relative performance of the IDA* search when using the two alternative bidtree orderings. For observations above the $x = y$ line, the decreasing sort performed better than the increasing sort. It clearly is not the case that the decreasing sort is *always* better. More interesting, perhaps, is the fact that there is very little correlation between the two orderings in terms of the difficulty of individual problems.

We also experimented with sorting the bid sets returned by a bidtree query. The idea is that if a large set is returned, then expanding with the lowest-cost bid first would generate the lowest-cost nodes earlier, thus reducing search effort. We could discern no impact on performance. Clearly, at least with the problems we are using, the performance improvement is no larger than the cost of sorting. Also, sorting the bids makes it more difficult to discard whole buckets when a single bid from a bucket turns out to be unusable (see line 22 of Figure 5.8).

6.4.2 Bid count experiment

Now that we know that the decreasing bidtree sort gives better performance than alternative arrangements, we can proceed to develop the data needed to predict runtime for the IDA* algorithm based on problem size metrics. We will not work with the A* version of the algorithm for this process, for two reasons: First, we know that when A* does not fail due to memory starvation, it nearly always beats IDA* by a small ratio (see Figures 6.25 and 6.32 for comparison). Therefore, knowing the performance and predictability data for IDA* gives us an upper bound on the A* performance. Second, we wish to run these experiments to complexity levels at which the A* version will fail due to memory exhaustion. The agent can easily choose A* when it knows the problem is small enough to solve within its memory limitations.

Table 6.13 shows sample data for the IDA* bid-count experiment. Compare this with the IP solver bid-count experiment in Section 6.2.2. We used larger problems for this set than we did for the IP experiment, simply to extend the range of our data. As in the IP experiment, each row represents 100 problems.

Table 6.13: Bid Count experiment for the IDA* solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	m	v	χ^2 (9 dof)
30	80	7.45	64	45.4	15.9	1.35	2.36
30	106	7.45	82	106	48.5	1.33	1.48
30	133	7.53	99	894	163	1.96	7.33
30	159	7.55	100	2180	500	2.36	3.50
30	186	7.48	100	34,700	1410	3.03	1.18
30	213	7.38	100	22,000	3070	3.33	2.92
30	239	7.46	99	137,000	9230	4.80	0.83
30	265	7.35	100	152,000	16,000	4.55	1.60

It seems clear that the 186-bid row in Table 6.13 contains a large outlier. Figure 6.27 shows the observed and inferred distributions for all of the problem sets in this experiment. Again, the lognormal distributions provide a good approximation for this data. For 9 dof, the 95% confidence point is approximately 3.5, so all but one of these sets is at or above a 95% match to the given lognormal distribution. This gives us confidence that we can base predictions on these inferred distributions. The single outlier in the 186-bid set can be seen clearly in the plot at about $3 \cdot 10^6$ msec.

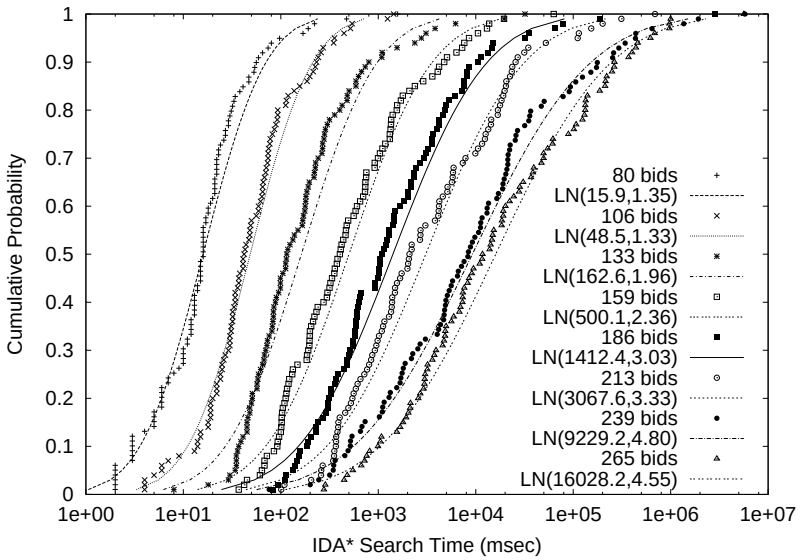

Figure 6.27: IDA* run-time distributions for problems with 30 tasks, bid count ranging from 80 bids to 265 bids, bid size approx. 7.5 tasks/bid.

6.4.3 Bid size experiment

For this experiment, we generated 100 problems with 30 tasks and 150 "bid attempts". We then varied the probability that the bidder will follow precedence links in generating multiple-task bids from 0.3 to 0.9. Table 6.14 gives the raw data and the parameters for the inferred lognormal distributions for this experiment. All but the 7.53 tasks/bid and the 11.5 tasks/bid sets fit the lognormal distribution quite well, as evidenced by the χ^2 values. Compare with the IP bid-size experiment in Section 6.2.3.

Table 6.14: Bid Size experiment for the IDA* solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	m	v	χ^2 (9 dof)
30	134	2.43	96	3,120,000	21,900	7.76	1.13
30	133	3.81	92	800,000	1300	5.38	1.66
30	133	5.58	97	2030	314	2.61	1.13
30	133	7.53	99	936	163	1.98	7.10
30	133	9.00	90	249	110	1.34	3.82
30	133	10.30	91	121	60.2	0.874	3.17
30	133	11.50	92	69.0	48.4	0.483	7.63

Figure 6.28 shows the observed and inferred runtime distributions for selected sets from this experiment. Compare with Figure 6.7. In both cases, both difficulty and variability rise significantly as bid size is reduced.

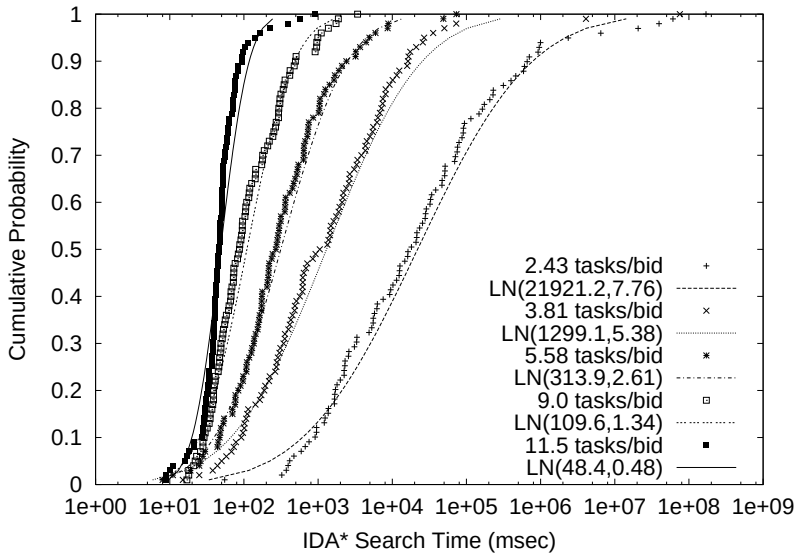

Figure 6.28: IDA* run-time distributions for problems with 30 tasks, 133 bids, bid size ranging from 2.4 tasks to 11.5 tasks/bid.

6.4.4 Task count experiment

For this experiment, we again generated 100 problems for each set, varying the task count over a 10:1 range from 5 tasks to 50 tasks, with the number of "bid attempts" at about 3.5 bids/task. Table 6.15 gives the raw data and the parameters for the inferred lognormal distributions for this experiment. These data do not fit the "inferred" lognormal distributions as well as the data in the previous section, for reasons that are unclear. In the first two sets that appears to be because of quantizing error (the clock resolution is only 1 ms, and the mean search time for the 5-task set was just 2.12 ms). For the other poor fits in the 35-task and 50-task sets, the errors seem unlikely to strongly impact our ability to use the 95% point as an estimator, since there is a crossover between the data and the lognormal curve in that region. Compare with the IP task-count experiment in Section 6.2.4.

Figure 6.29 shows the observed and inferred runtime distributions for selected sets from this experiment. Compare with Figure 6.9. The performance advantage of IDA* over IP is dramatically evident in this comparison. Note, however, that this experiment has no unix process-setup overhead, since the IDA* solver is integrated into the Java agent framework.

We now turn to the problem of finding the regression formula and coefficients for this data. Using the same 3 regression formulas as we used in Section 6.2.4, the resulting coefficients are:

$$\begin{array}{lllll} x_1 = 0.237 & x_2 = 0.0729 & x_3 = -1.401 & x_4 = 4.27 & \sigma_x = 1.364 \\ x'_1 = -0.180 & x'_2 = 0.0687 & x'_3 = -38.7 & x'_4 = 16.8 & \sigma'_x = 1.457 \\ x''_1 = -0.0120 & x''_2 = 0.0718 & x''_3 = 1.351 & x''_4 = -1.14 & \sigma''_x = 0.247 \end{array}$$

Table 6.15: Task Count experiment for the IDA* solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	m	v	χ^2 (9 dof)
5	13.5	2.10	94	2.12	1.81	0.499	31.7
10	29.0	3.27	94	4.78	3.01	0.716	9.27
15	45.4	4.46	90	9.22	5.93	0.664	4.02
20	61.0	5.45	85	22.4	12.2	1.18	3.97
25	77.7	6.30	85	31.1	19.0	0.874	0.76
30	93.4	7.40	80	81.4	27.3	0.955	4.03
35	111	8.42	74	914	47.0	1.96	8.55
40	127	9.65	68	149	61.1	1.35	2.99
45	142	11.2	66	146	74.7	1.19	1.44
50	160	12.2	61	597	98.9	1.61	6.38

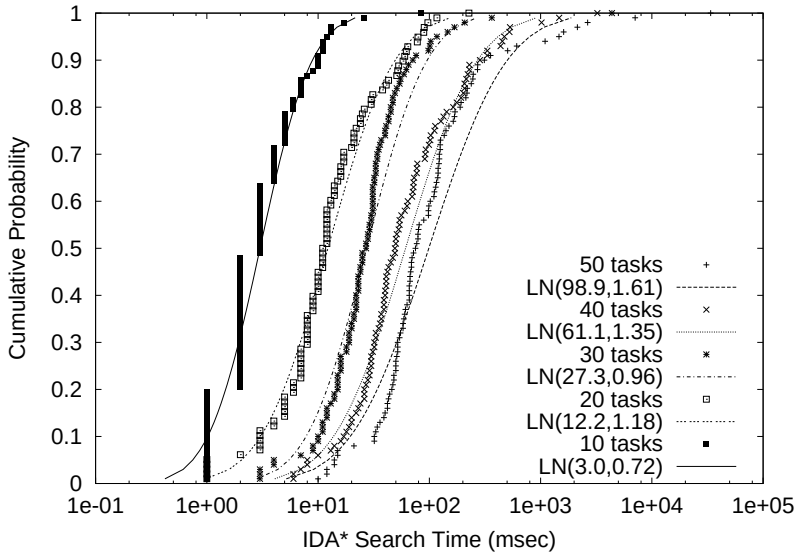

Figure 6.29: IDA* run-time distributions for problems with 10 to 50 tasks, about 3.1 bids/task.

6.4.5 Task network complexity experiment

In this experiment, we used the same sets of 100 problems as we used in Section 6.2.5, and we added one additional problem with a high fan-in and low bid-size. This last problem turned out to be beyond the abilities of our IP solver, and so we have only IDA* data for it. As in the previous network complexity experiment, the branch factor is varied from 1.0 to 4.0 in increments of 1.0. The results are shown in Table 6.16. There were again 35 tasks in each set of 100 problems.

The probability distributions for this set are shown in Figure 6.31. Here we see a major departure in the relative performance of the IP and IDA* solvers. In Section 6.2.5, we see the performance dominated by the number of precedence relations, likely because of the number of IP rows generated by the bid precedence relationships. Here, we see the problem difficulty dominated by the bid size;

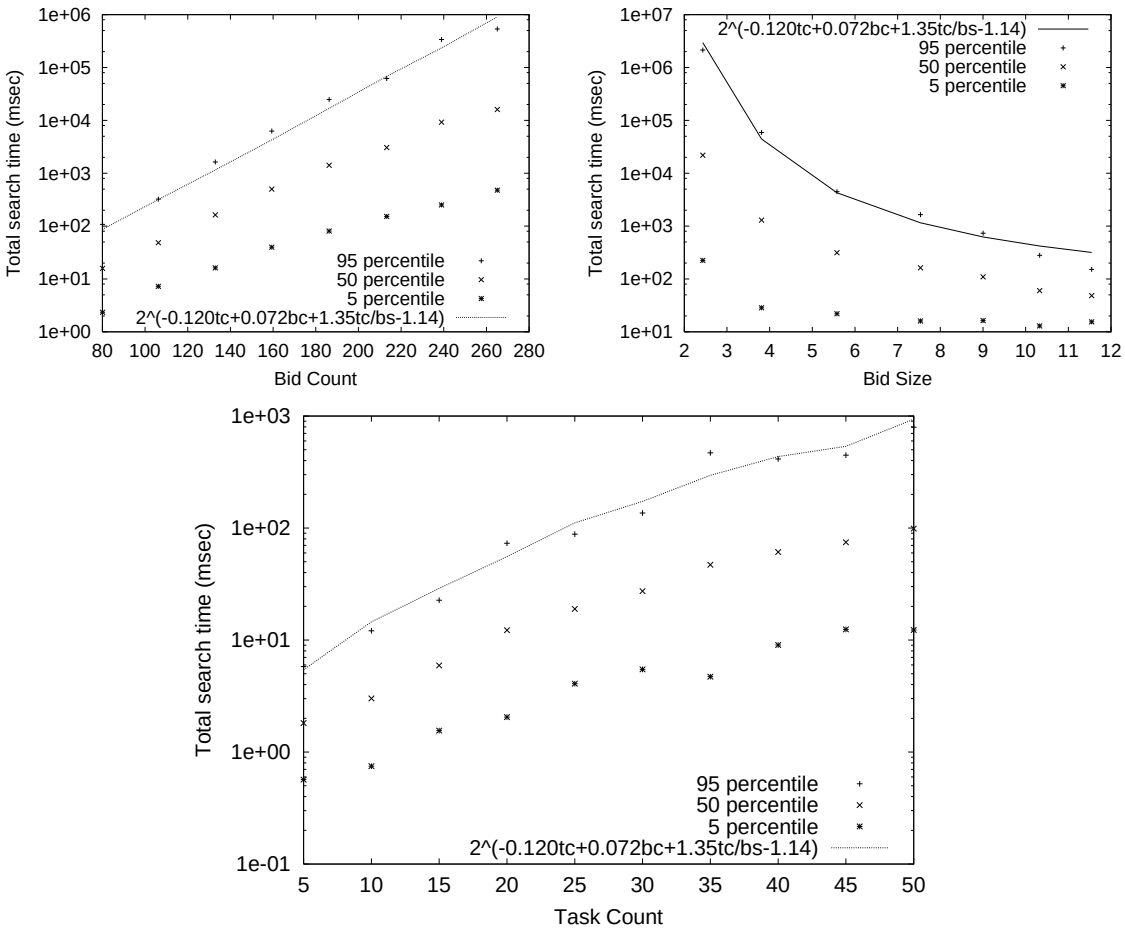

Figure 6.30: Estimating IDA* search time for the bid-count and bid-size experiments (top), and the task-count experiment (bottom).

Table 6.16: Task network complexity experiment for the IDA* solver.

Fan In	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	σ	m	v	χ^2 (9 dof)
0.489	112	2.01	86	76100	516,000	505	6.74	0.966
0.876	111	5.06	100	427	2000	75.8	2.15	0.974
1.371	109	11.9	100	30.5	22.8	22.9	0.632	0.972
1.735	109	16.0	100	20.7	14.8	15.9	0.552	0.930
1.735	109	2.65	56	365,000	2,044,000	1975	9.85	1.000

Figure 6.31 looks much more like Figure 6.28 than it does like Figure 6.11.

To derive a regression formula that includes the network complexity results, we have used the same factors we used in the IP regression formula in Section 6.2.5. Table 6.17 shows the factors in the left column, and results for the "raw" numbers, for a normalized version titled "Norm 1" (which of

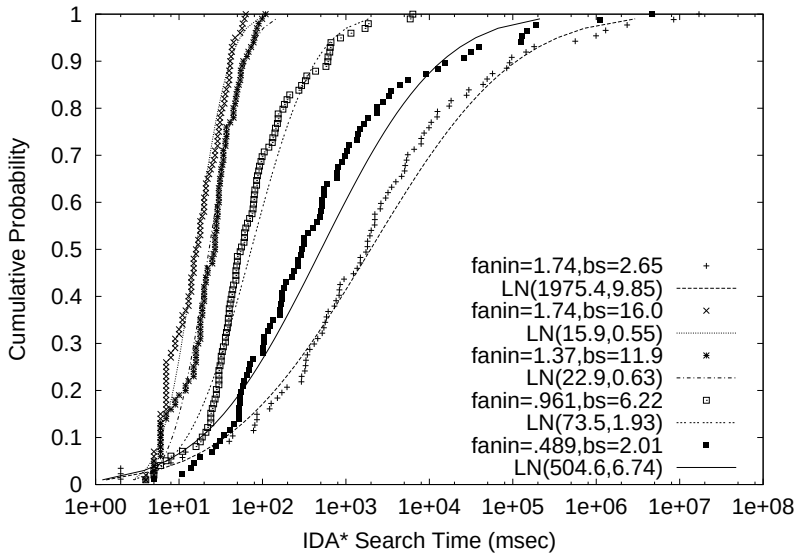

Figure 6.31: Observed and inferred runtime distributions for the IDA* solver across a range of task network complexity values.

course omits the "1" factor) and for a normalized version titled "Norm 2" that omits the two factors fi/tc and $\log_2 fi/tc$.

Table 6.17: IDA* regression factors and results for raw and normalized data.

Factor	Raw	Norm 1	Norm 2
task count tc	-0.645	-1.225	-2.06
bid count bc	0.0708	0.832	0.833
bid size bs	-0.358	-0.249	-0.250
tc/bs	1.05	0.695	0.695
fan-in fi	-15.4	-0.610	-0.986
fi/tc	-11.4	-0.0735	
$fi \cdot tc$	0.378	0.978	1.94
$fi/\log_2 tc$	-35.6	-0.378	-0.585
$\log_2 fi/tc$	-27.4	-0.0621	
$\log_2 fi \cdot tc$	0.386	0.796	0.747
1	30.1		
σ	0.189	0.00835	0.00768

Compare these results with Table 6.7. Fan-in remains a negative factor, but doesn't dominate the formula as it did with the IP data. and the error is remarkably low. As was the case with the data in Section 6.2.5, the task network complexity was relatively constant for the task-count, bid-count, and bid-size experiments, and so these numbers could presumably be improved with additional experiment sets.

6.4.6 Problem-by-problem comparisons of IP, A*, and IDA* solvers

Figure 6.32 shows the relative performance of the A* and IDA* solvers, using a bidtree with decreasing sort, on the 35-task, 111-bid problem set from Section 6.4.4. This is the largest problem size for which the A* memory limitation does not cause serious problems. We can see that there is a very strong correlation between the runtimes of the two formulations. The one outlier was an outlier for both methods, and A* generally beats IDA* by about a factor of 2.

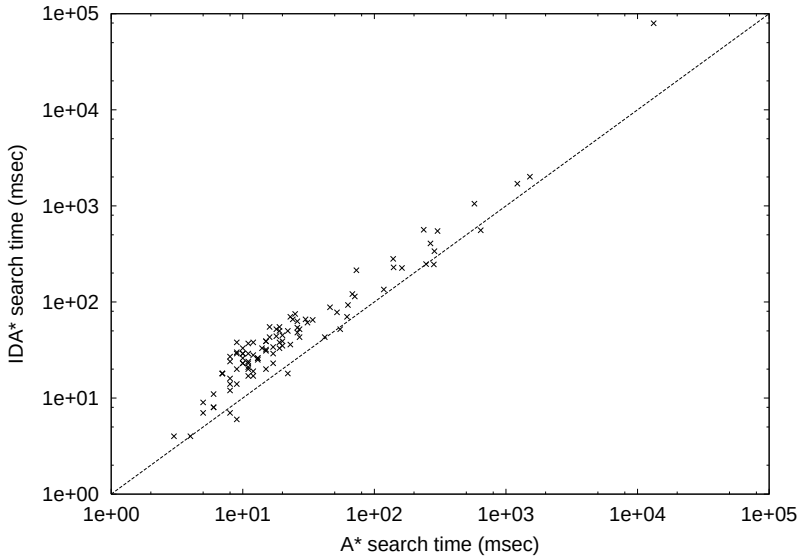

Figure 6.32: Problem-by-problem comparison of A* and IDA* run-times from 35-task, 111-bid set of Section 6.4.4.

In Figure 6.33, we see the relative performance of the IP solver and the IDA* solver on the 50-task, 160-bid set from Section 6.4.4. As was the case for the IP solver and the SA solver in Section 6.3.5, there is virtually no correlation between the runtimes of the two solvers. The IDA* solver is the clear winner in almost all cases, with one dramatic exception.

Before we leave with the impression that IDA* is a faster search method than IP for the MAGNET winner-determination problem, we need to recall the results on the task network complexity experiments. In Figure 6.34, we see the extreme cases in this experiment. In the left plot, we see the generally faster performance and much lower variability of the IP solver for the low-fan-in problem set. For points plotted above the $x = y$ line, IP was faster. The IDA* solver would clearly not be the best choice for these problems, even though it does win, sometimes by more than an order of magnitude, in about a third of the cases. On the other hand, the right plot of Figure 6.34 shows the other extreme. Here, there are no cases where IP wins, and in fact the $x = y$ line is completely above the area of the plot.

As a final sanity check, we run our predictions on two problems that were aborted because it was clear they would not finish in a reasonable time. The first is the time for the IP solver on the fifth problem in Table 6.16. Using the "raw" coefficients from Table 6.7, we get a 95% confidence

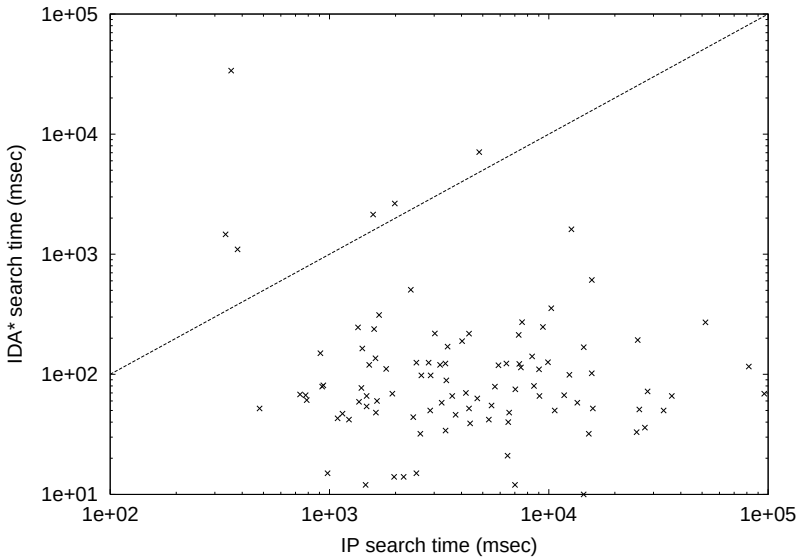

Figure 6.33: Problem-by-problem comparison of IP and IDA* run-times from 50-task, 160-bid set of Section 6.4.4. Points above the $x = y$ line are problems for which the IP solver was faster than the IDA* solver.

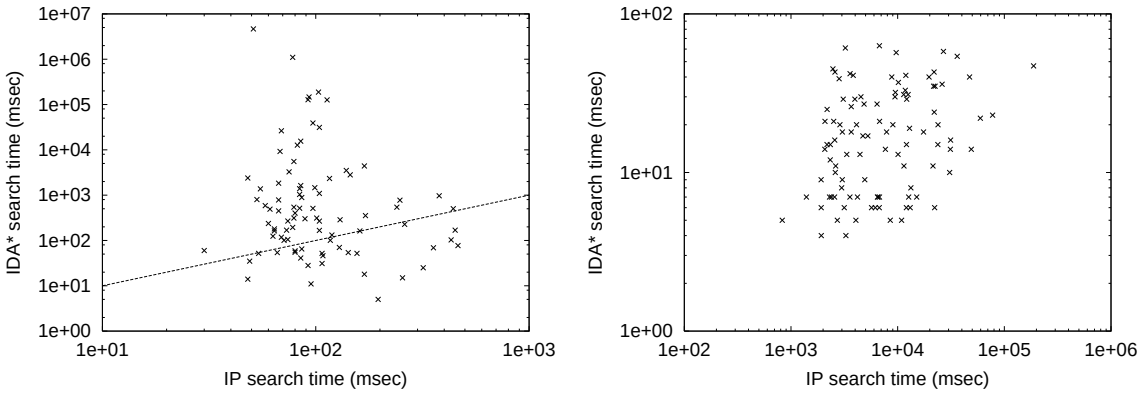

Figure 6.34: Extreme examples of IP vs. IDA* performance comparisons, 35 tasks, 111 bids: fan-in=0.49, bid-size=2.0 (left); fan-in=1.7, bid-size=16 (right).

of finishing a single problem instance in $4.11 \cdot 10^9$ msec, or about 1140 hours. The second was an attempt to run the IDA* solver on a problem with $tc = 100$ tasks, $bc = 900$ bids, bid size $bs = 26$, fan in $fi = 1.29$. The predicted 95% confidence runtime is $1.25 \cdot 10^{18}$ msec, more than 10^7 years.

6.5 Using performance data for deliberation scheduling

Now that we have regression formulas that describe the performance of the IP and IDA* solvers, the next question is how the agent should best use them. From the data we have developed here, it seems that there are at least two decisions the agent must make. The first is deciding which search method to use, and the second is calibrating its platform to convert the search complexity

predictions to actual runtimes.

Our data shows that for most of the experimental conditions we used, the IDA* search was the best performer (A* for smaller problems if sufficient memory is available). However, the results from the network complexity experiment demonstrate that there are some problem profiles for which the IP solver is the better performer. If the agent is able to accomodate the limitations of the IP solver (for example, not being able to optimize the distribution of slack in the schedule), then for those problems the IP solver is clearly the best choice. The decision would be driven by measuring and estimating the problem-size parameters, and then solving both of the empirical runtime formulas. If one is predicted to be substantially faster (more than a factor of 4, for example) than the other, then it should be used. If they are nearly identical, then doubling the time and running both of them in parallel can reduce the uncertainty, due to the lack of correlation in their performance results on individual problems.

The obvious way to address the calibration problem is for the agent to test its computing environment when it starts up, measuring its performance against a benchmark. The benchmark should be a run of the winner-determination solver on a sample problem, rather than some simple loop test, because computing platforms vary in performance along many dimensions. We have not implemented this behavior yet, and there remain many unanswered questions related to this exercise:

- On machines with limited memory or contention from other processes, the timing of a single trial benchmark can be almost meaningless, depending on what else was happening at the time, or what else might be happening when the need arises to evaluate a real set of bids.
- Because of virtual memory effects, and because of Java class loading and just-in-time compiling behavior, the first trial of anything after starting up a system can be significantly slower than subsequent runs. In other words, the benchmark needs to be run at least twice, with the first result discarded. This is mitigated in the experiments we reported here because we use hundreds of runs, because we used a dedicated machine with ample memory, and because operating system optimizations virtually eliminated the need to read code from disk in most cases. For example, although there was process set-up overhead in our trials using the external `lp_solve` process, the disk cache behavior was able to eliminate the need to read the code from disk after the first trial.
- The agent is a multi-threaded application, and can manage many negotiation sessions simultaneously. There is currently nothing to prevent it from committing itself to evaluation of multiple auctions in the same time period. This could be managed by maintaining a schedule of such commitments in the time map. When planning an RFQ, a request for evaluation time would be made, and space allocated in the time map. To avoid conflict with prior evaluation commitments, bids would not be evaluated until their scheduled evaluation time arrived, or until the computing resource became free earlier than expected.
- The agent may be running on a multi-processor platform, and may be able to make use of multiple processors to handle multiple threads of evaluation. The current design runs

the evaluation in a separate thread from the agent itself, but the evaluators themselves are not multi-threaded. Both the SA and IDA* solvers could potentially benefit from multi-thread implementations on a multi-processor platform, but we currently lack easy access to a computing environment to test this idea. The question of how the agent could detect that it could benefit from a multi-processor platform is open, but it might make sense to make this a configuration option. Just tell the agent how many processors it is allowed to use, and it can build multiple schedules in its time map.

- The agent is designed to be a persistent entity, with persistent state and identity. This means that it can be shut down, moved to a different platform, and restarted. What happens to outstanding evaluation commitments when this happens, and how does the agent detect and adapt to a new platform when it starts up? These are open design issues.

As a result of examining these issues, it appears that the agent needs some control of its computing environment if it is to operate in a mode where it needs to produce winner-determination results on a tight timeline. This is probably not a major issue in domains where the bid-evaluation time is dominated by user think time, but even then, it may be important to help the user understand how performance is deviating from predicted values so that the impact of concurrent processor demands, memory limits, etc. are clear to the user. In domains where the agent is making decision autonomously, it may need dedicated hardware in order to meet its commitments reliably.

Chapter 7

Conclusions

We have examined the problem of rational economic agents, who must negotiate among themselves in a market environment in order to acquire the resources needed to accomplish their goals. We are interested in agents that are self-interested and heterogeneous, and we assume that a plan to achieve an agent's goal may be described in the form of a *task network*, containing task descriptions, precedence relationships among tasks, and time limits for individual tasks. Negotiation among agents is carried out by holding *auctions* in a marketplace, in which a *customer* agent offers a task network in the form of a *request for quotations* (RFQ). *Supplier* agents may then place bids on portions of the task network, each bid specifying the tasks they are interested in undertaking, durations and time limits for those tasks, and a price for the bid as a whole. Because the bid-taker is the buyer, this is a *reverse auction*, and because bidders can bid on bundles, it is also a *combinatorial auction*. The presence of temporal and precedence constraints among the items at auction requires extensions to the standard *winner-determination* procedures for combinatorial auctions, and the use of the enhanced winner-determination procedure within the context of a real-time negotiation requires us to be able to predict its runtime when planning the negotiation process.

There are a number of real-world business scenarios where such a capability would be of value. These include flexible manufacturing, mass customization, travel arrangement, logistics and international shipping, health care resource management, and large-scale systems management. Each of these areas is characterized by limited capabilities and suboptimal performance, due at least in part to the limits imposed by human problem-solving capabilities. In each of these areas, a general ability to coordinate plans among multiple independent suppliers would be of benefit, but does not exist or is not used effectively because of an inability to solve the resulting combinatorial problems. The use of extended combinatorial auctions such as we propose in this dissertation is one approach to solving these problems. There are many difficulties yet to be overcome before this vision can be realized, however, not the least of which is that such auction-based markets would not be effective without wide adoption of new technology across an industry, and a willingness to delegate at least some level of autonomy and authority to that new technology.

7.1 Review

We have designed and implemented a testbed that has been used for the experimental work reported here, and we expect that it will continue to help us work out other aspects of our vision. We call this testbed MAGNET for Multi-AGent NEgotiation Testbed. It includes highly configurable, component-based implementations of a customer agent, a rudimentary market infrastructure, and a simple supplier agent. The customer agent implementation is designed so that virtually all behaviors can be specified and implemented in terms of responses to events. Events can be external occurrences, internal state changes, or the arrival of a particular point in time. Multiple threads of activity can be managed concurrently without interference. The market infrastructure is designed as a 3-level hierarchy of *sessions*, *markets*, and an *exchange*. Sessions encapsulate the interactions between a single customer agent and its suppliers during the negotiation and execution of a single customer plan. Markets provide a matchmaker service, helping customers find suppliers who can fulfill their needs. Markets also provide an *ontology* that defines the terms of discourse within the scope of the market. At a minimum, this includes common definitions of all the task types that can be sold within the scope of the market. In addition, markets capture ongoing transaction data and reduce it to statistical data that agents can use to support their decisions. The Exchange hosts a set of markets, and provides basic identity and finder services. The MAGNET software package is being made available to the research community under an open-source license.

In order to carry out its functions as an autonomous, rational economic agent in a market environment, a customer agent must be able to maximize its utility, or the utility of its principal, at each of a number of decision points. Because the outcomes of an agent's actions are uncertain, we must consider alternative possible outcomes for each action. The theory of *Expected Utility*, a well-developed if somewhat controversial theory in Economics, provides us with a way of dealing with the risk posture of a real-world decision-maker. Generally, we expect that aversion to loss will be greater than desire for an equal gain; this is known as a *risk-averse* posture.

When a goal arises, the agent and its principal must develop a plan, in the form of a *task network*. If any portions of the plan may be satisfied by market-based negotiation, then the plan must be composed of task types that are known in the market. The market's ontology serves this purpose. Once a plan is available, a *bid-process plan* must be developed to guide the negotiation process. The bid-process plan specifies which tasks are to be offered in which markets, allocates time to the bidding process and to the plan execution, and may split the bidding into phases in order to mitigate risk. For each bidding step in the bid-process plan, time must be allocated to the customer to compose its RFQ, to the supplier to compose bids, and to the customer to evaluate bids. This need to allocate time drives the need to characterize the winner-determination search process. For each auction episode specified in the bid-process plan, a RFQ must be composed. The RFQ specifies a subset of tasks in the task network, and for each task, it specifies a *time window* within which that task must be accomplished. The setting of time windows is critical, because it influences the likelihood that bidders will bid, the prices bidders are likely to charge, and the difficulty of the resulting winner-determination process. If the time windows specified in the RFQ allow task precedence relationships to be violated, then the winner-determination process will need to choose

a set of bids that can be composed into a feasible schedule. Once the RFQ has been issued and bids received, the agent must choose the "best" set of bids to carry out its plan. It is possible that no set is acceptable, and it is possible that some tasks will not receive any bids. The winner-determination process treats bid evaluation as a combinatorial optimization problem. If the final decision is to be made by a human user, then several runs of the winner-determination process may be required, possibly using different sets of additional constraints, before a final decision can be made. After the winner-determination process has recommended a set of bids to accept, the agent must generate a final schedule so that it can advise suppliers of actual start times for the awarded tasks. This is necessary because suppliers are expected to offer ranges of task-execution times in their bids in order to make the bids more attractive to the customer, but in general they will not be willing to reserve resources for a large range of task start times, once the bid-award time has passed.

We then focused our attention on the winner-determination problem. We have developed three different solutions to the extended winner-determination problem. The first is an Integer Programming (IP) model, based on the model presented by Andersson in [Andersson *et al.*, 2000]. A straightforward extension of the Andersson model, in which the precedence constraints are encoded directly in the model, proved to be extremely unwieldy, and so we also present a simplified model that depends on preprocessing. The preprocessing step must walk every path of length ≥ 2 in the precedence network, but the end result is much improved performance over the original model. Second, we introduce a stochastic search formulation based on a queue-based Simulated Annealing (SA) model. The SA search is incomplete; it is not guaranteed to find a solution if one exists, and even though it often finds optimal solutions, it cannot prove that they are optimal. A characteristic of most SA implementations is that there are many tuning parameters, and our version is no exception. Finally, we have extended Sandholm's bidtree-based Iterative Deepening A* (IDA*) winner-determination method [Sandholm, 2002] to handle the additional constraints in the MAGNET problem. In the process, we discovered that the order in which tasks are selected to build the bidtree makes a large difference in the expected complexity of the search. We defined and implemented both A* and IDA* versions of the bidtree algorithm.

Because the winner-determination problem must be solved within a predetermined period of time, it is important to have a clear idea of how much time to allocate to it, and to know what parameters to use in predicting its run time. We therefore conducted a series of experiments to characterize the performance of all of our winner-determination methods over a variety of problem sizes and shapes. The goal was to determine probability distributions, so that we could allocate the time necessary to achieve a known probability of solving the problem. We found that the lognormal distribution was a good model of search performance for both the IP and the IDA* methods, but not for the SA method. All methods were observed to scale exponentially with respect to number of bids, and with respect to the ratio of task count (the number of tasks in the task network) to bid size (the mean number of tasks specified in each bid). We also characterized the performance of the SA method on several individual problem, to get a clear picture of the variability of this stochastic algorithm independent of problem variation. We observed that the Weibull distribution could be used to model this variation closely. We compared the different winner-determination methods to each other both

in aggregate and problem-by-problem. In aggregate terms, the IDA* search outperforms IP, which outperforms SA. On a problem-by-problem basis, we observed very little correlation between the performance of IP and SA, or between IP and IDA*. This can be interpreted as telling us that a problem that is especially hard for IDA*, for example, is not necessarily hard for IP, and vice versa. As a consequence, it appears that one could run two searches with different characteristics in parallel, stopping when the first search produced an answer, thus reducing overall variability.

Finally, we used a set of linear regression formulas to develop predictive models for performance of the IP and IDA* searches. We did not attempt this with the SA search, because we have not found a good model for its overall variability, and because its performance is sensitive to a large number of parameters, each of which would likely have to participate in such a model. Because the runtimes of these search method scale exponentially with respect to most parameters, we used log runtime as the dependent variable. For both the IP and IDA* searches, we found coefficients for combinations of bid count, task count, bid size, and fan-in (the average number of predecessors of a task) that resulted in a standard error of at most 12% over all experiments. At the time the search runtime must be predicted, we have task count directly from our task network, and we can use market statistics to estimate the bid count and bid size parameters.

7.2 Future research

Much work remains to be done before the vision of the MAGNET project is fully realized. Some of that work, particularly with respect to the supplier agent and its decision processes, is already under way by other members of the team.

With respect to the customer agent, most of the decision processes outlined in Chapter 4 still need to be worked out and tested. The present work has resulted in models for the auction winner-determination problem and the time that must be allocated to it. For the remainder of the decisions, we need models that will maximize the expected utility of the agent or its principal. These include composing the plan, developing the bid-process plan, allocating time to the deliberation processes of the customer and suppliers, balancing negotiation time against plan execution time, setting the time windows in the RFQ, scheduling the work in preparation for awarding bids, and dealing with unexpected events during plan execution.

The language we currently use for plans and bids treats tasks as simple atomic objects, without attributes. There are many real-world problems in which attributes are important, both for specifying tasks and for expressing offers in bids. Examples include colors, quantities, dimensions, and quality attributes. In addition, many real-world operations operate on a "flow" basis. This includes the wine-making example we used in Chapter 4, in which realistically, the precedence between filling bottles and applying labels would normally be applied bottle-by-bottle, and not at the batch level. In addition, the expressivity of our bidding language is limited. A number of proposals have been made for more expressive bidding languages in combinatorial auctions, including those of Nisan [Nisan, 2000a] and Boutilier and Hoos [Boutilier and Hoos, 2001]. Bidding can also be done with *oracles*, which are functions passed from bidder to customer that can be evaluated to produce bid condi-

tions. Some features of a more expressive bidding language would likely have minimal impact on the winner-determination process (parameterized quality values, for example), while others, including the use of oracles, could require wholesale re-invention.

Because rational economic agents will likely be acting on behalf of real people and organizations, making financial and resource commitments, it is important that users be given the level of control and transparency that will make them comfortable. On the other hand, the reason agents are interesting in the first place is to give users and their organizations a level of market presence and an ability to evaluate alternatives that are well beyond unaided human performance levels. This argues for a mixed-initiative approach to the human-agent interaction, with the agent alerting the decision-making person to opportunities and alternatives, and providing a rich decision support environment, while the human responds, focuses attention, and makes the final decisions in many cases.

There remain several open problems before the performance data we have developed can be used in a practical, deployed agent, especially if the agent is to operate autonomously. These have to do with measuring and adapting to its computing environment, and scheduling its own computing resources across potentially multiple threads of negotiation. Many of these are design and deployment issues, but some, including issues around maintaining state and commitments across restarts on potentially different platforms, and making users aware of performance issues, may merit further research.

Our experimental work so far has used randomly-generated abstract problems, with only a minimal effort to make the problems reflect "real-world" situations. Our initial attempt to gather such real-world data from a shipping company was not successful; the data was not designed to support the types of statistics we are interested in, and it was not possible to massage it into a form that would support statistical study. In the future, it will be useful to find industrial and commercial partners who can work with us to realize the vision of the MAGNET project.

Bibliography

[Andersson *et al.*, 2000] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *Proc. of 4th Int'l Conf on Multi-Agent Systems*, pages 39–46, July 2000.

[Babanov *et al.*, 2002] Alex Babanov, John Collins, and Maria Gini. Risk and expectations in a-priori time allocation in multi-agent contracting. In *Proc. of the First Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July 2002.

[Bakos, 1998] Yannis Bakos. The emerging role of electronic marketplaces on the Internet. *Comm. of the ACM*, 41(8):33–42, August 1998.

[Bigus, 2002] Joe Bigus. Personal communication, March 2002.

[Biswas, 1997] Tapan Biswas. *Decision-Making Under Uncertainty.* St. Martin's Press, Inc., 1997.

[Boddy and Dean, 1994] Mark Boddy and Thomas Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–286, 1994.

[Boddy, 1993] Mark S. Boddy. Temporal reasoning for planning and scheduling. *ACM SIGART Bulletin*, 4(3), 1993.

[Boutilier and Hoos, 2001] Craig Boutilier and Holger H. Hoos. Bidding languages for combinatorial auctions. In *Proc. of the 17th Joint Conf. on Artificial Intelligence*, pages 1211–1217, 2001.

[Bradshaw, 1997] Jeffrey M. Bradshaw, editor. *Software Agents.* AAAI/MIT Press, 1997.

[Bresina, 1996] John L. Bresina. Heuristic-biased stochastic sampling. In *Proc. of the Thirteenth Nat'l Conf. on Artificial Intelligence*, 1996.

[Chavez and Maes, 1996] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. of the First Int'l Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.

[Cohen, 1995] Paul R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, Cambridge, MA, 1995.

[Collins *et al.*, 1997] John Collins, Scott Jamison, Maria Gini, and Bamshad Mobasher. Temporal strategies in a multi-agent contracting protocol. In *AAAI-97 Workshop on AI in Electronic Commerce*, July 1997.

[Collins *et al.*, 1998] John Collins, Maxim Tsvetovat, Bamshad Mobasher, and Maria Gini. MAGNET: A multi-agent contracting system for plan execution. In *Proc. of SIGMAN*, pages 63–68. AAAI Press, August 1998.

[Collins *et al.*, 2000a] John Collins, Corey Bilot, Maria Gini, and Bamshad Mobasher. Mixed-initiative decision support in agent-based automated contracting. In *Proc. of the Fourth Int'l Conf. on Autonomous Agents*, pages 247–254, June 2000.

[Collins *et al.*, 2000b] John Collins, Rashmi Sundareswara, Maria Gini, and Bamshad Mobasher. Bid selection strategies for multi-agent contracting in the presence of scheduling constraints. In A. Moukas, C. Sierra, and F. Ygge, editors, *Agent Mediated Electronic Commerce II*, volume LNAI1788. Springer-Verlag, 2000.

[Cox and Sadiraj, 2001] James C. Cox and Vjollca Sadiraj. Risk aversion and expected utility theory: Coherence for small- and large-stakes gambles. Working paper, http://w3.arizona.edu/ econ/working_papers/riskavers.pdf, 2001.

[de Vries and Vohra, 2001] Sven de Vries and Rakesh Vohra. Combinatorial auctions: a survey. Technical report, Techniche Universität München, 2001. http://www-lit.ma.tum.de/veroeff/quel/019.90003.pdf.

[Faratin *et al.*, 1997] Peyman Faratin, Carles Sierra, and Nick R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1997.

[Fox, 1996] Mark S. Fox. An organization ontology for enterprise modeling: Preliminary concepts. *Computers in Industry*, 19:123–134, 1996.

[Fujishjima *et al.*, 1999] Yuzo Fujishjima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proc. of the 16th Joint Conf. on Artificial Intelligence*, 1999.

[Glass and Grosz, 2000] Alyssa Glass and Barbara J. Grosz. Socially conscious decision-making. In *Proc. of the Fourth Int'l Conf. on Autonomous Agents*, pages 217–224, June 2000.

[Gomes *et al.*, 1998] Carla Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proc. of the Fifteen Nat'l Conf. on Artificial Intelligence*, pages 431–437, 1998.

[Greenwald and Dean, 1995] Lloyd Greenwald and Thomas Dean. Anticipating computational demands when solving time-critical decision-making problems. In K. Goldberg, D. Halperin, J.C. Latombe, and R. Wilson, editors, *The Algorithmic Foundations of Robotics*. A. K. Peters, Boston, MA, 1995.

[Gruninger and Fox, 1994] M. Gruninger and M. S. Fox. An activity ontology for enterprise modelling. In *Workshop on Enabling Technologies - Infrastructures for Collaborative Enterprises*. West Virginia University, 1994.

[Guttman *et al.*, 1998] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated electronic commerce: a survey. *Knowledge Engineering Review*, 13(2):143–152, June 1998.

[Harvey and Ginsberg, 1995] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proc. of the 14th Joint Conf. on Artificial Intelligence*, pages 607–613, 1995.

[Helper, 1991] S. Helper. How much has really changed between US manufacturers and their suppliers. *Sloan Management Review*, 32(4):15–28, 1991.

[Hillier and Lieberman, 1990] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 1990.

[Hooker, 2000] John Hooker. *Logic-Based Methods for Optimization*. John Wiley & Sons, 2000.

[Hoos and Boutilier, 2000] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence*, pages 22–29, 2000.

[Hoos and Stützle, 1998] Holger H. Hoos and Thomas Stützle. Evaluating Las Vegas algorithms – pitfalls and remedies. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 238–245. Morgan Kaufmann Publishers, 1998.

[Hoos, 1998] Holger H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, the Darmstadt University of Technology (Germany), 1998.

[Hunsberger and Grosz, 2000] Luke Hunsberger and Barbara J. Grosz. A combinatorial auction for collaborative planning. In *Proc. of 4th Int'l Conf on Multi-Agent Systems*, pages 151–158, Boston, MA, 2000. IEEE Computer Society Press.

[Jose and Ungar, 1998] Rinaldo A. Jose and Lyle H. Ungar. Auction-driven coordination for plantwide optimization. In *Proc. Foundations of Computer-Aided Process Operation FOCAPO*, 1998.

[Josephs, 2001] Leonard Josephs. Market architecture for multi-agent contracting: an internet standards based approach. Master's thesis, University of Minnesota, Minneapolis, MN, 2001.

[Jullien and Salanié, 2000] Bruno Jullien and Bernard Salanié. Estimating preferences under risk: The case of racetrack bettors. *The Journal of Political Economy*, 108(3):503–530, June 2000.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of the Thirteenth Nat'l Conf. on Artificial Intelligence*, volume 2, pages 1194–1201. MIT Press, 1996.

[Keller, 1996] Arthur M. Keller. Smart catalogs and virtual catalogs. In R. Kalakota and A. Whinston, editors, *Readings in Electronic Commerce*. Addison-Wesley, Reading, Mass., 1996.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, May 1983.

[Klein *et al.*, 2002] Mark Klein, Peyman Faratin, Hiroki Sayama, and Yaneer Bar-Yam. Negotiating complex contracts. In *Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, 2002. AAAI Press.

[Korf, 1985] Richard E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[Kreps, 1990] D. M. Kreps. *A Course in Microeconomic Theory*. Princeton University Press, 1990.

[Kuokka and Harrada, 1995] Daniel Kuokka and Lawrence Harrada. On using KQML for matchmaking. In *Proc. 3rd Int'l Conf on Information and Knowledge Management*, pages 239–245, 1995.

[Langley, 1992] Pat Langley. Systematic and nonsystematic search strategies. In *Proc. Int'l Conf. on AI Planning Systems*, pages 145–152, College Park, Md, 1992.

[Law and Kelton, 1991] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, second edition, 1991.

[Leyton-Brown *et al.*, 2000a] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proc. of ACM Conf on Electronic Commerce (EC'00)*, Minneapolis, MN, October 2000.

[Leyton-Brown *et al.*, 2000b] Kevin Leyton-Brown, Yoav Shoham, and Moshe Tennenholtz. An algorithm for multi-unit combinatorial auctions. In *Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence*, Austin, Texas, 2000.

[Maes *et al.*, 1999] Pattie Maes, Robert H. Guttman, and Alexandros G. Moukas. Agents that buy and sell: Transforming commerce as we know it. *Comm. of the ACM*, 42(3), March 1999.

[Mas-Colell *et al.*, 1995] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, January 1995.

[McAfee and McMillan, 1987] R. McAfee and P. J. McMillan. Auctions and bidding. *Journal of Economic Literature*, 25:699–738, 1987.

[McConnell *et al.*, 1997] S. McConnell, M. Merz, L. Maesano, and M. Witthaut. An open architecture for electronic commerce. Technical report, Object Management Group, Cambridge, MA, 1997.

[Nisan, 2000a] Noam Nisan. Bidding and allocation in combinatorial auctions. In *Proc. of ACM Conf on Electronic Commerce (EC'00)*, pages 1–12, Minneapolis, Minnesota, October 2000. ACM SIGecom, ACM Press.

[Nisan, 2000b] Noam Nisan. Bidding and allocation in combinatorial auctions. Technical report, Institute of Computer Science, Hebrew University, 2000.

[Oddi and Smith, 1997] Angelo Oddi and Stephen F. Smith. Stochastic procedures for generating feasible schedules. In *Proc. of the Fourteenth Nat'l Conf. on Artificial Intelligence*, pages 308–314, 1997.

[Ontology.org, 2002] Enabling virtual business. http://www.ontology.org, 2002.

[Parkes and Ungar, 2000] David C. Parkes and Lyle H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence*, pages 74–81, 2000.

[Parkes and Ungar, 2001] David C. Parkes and Lyle H. Ungar. An auction-based method for decentralized train scheduling. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 43–50, Montreal, Quebec, May 2001. ACM Press.

[Pollack, 1996] Martha E. Pollack. Planning in dynamic environments: The DIPART system. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, 1996.

[Rabin, 2000] Matthew Rabin. Risk aversion and expected utility theory: a calibration theorem. *Econometrica*, 68:1281–1292, 2000. http://iber.berkeley.edu/wps/econ/E00-279.pdf.

[Reeves *et al.*, 2001] Daniel M. Reeves, Michael P. Wellman, and Benjamin N. Grosof. Automated negotiation from declarative contract descriptions. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 51–58, Montreal, Quebec, May 2001. ACM Press.

[Reeves, 1993] Colin R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, New York, NY, 1993.

[Rodriguez *et al.*, 1997] J. A. Rodriguez, P. Noriega, C. Sierra, and J. Padget. FM96.5 - A Java-based electronic auction house. In *Second Int'l Conf on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*, London, April 1997.

[Ronén *et al.*, 1998] Yagíl Ronén, Daniel Mossé, and Martha E. Pollack. Using value-density algorithms to handle transient overloads in deliberation scheduling. *IEEE Intelligent Systems*, 13(4), 1998.

[Rosenschein and Zlotkin, 1994] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter*. MIT Press, Cambridge, MA, 1994.

[Rothkopf *et al.*, 1998] Michael H. Rothkopf, Alexander Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[Russell and Norvig, 1995] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, NJ, 1995.

[Sadeh *et al.*, 1999] Norman M. Sadeh, David W. Hildum, Dag Kjenstad, and Allen Tseng. MASCOT: an agent-based architecture for coordinated mixed-initiative supply chain planning and scheduling. In *Workshop on Agent-Based Decision Support in Managing the Internet-Enabled Supply-Chain, at Agents '99*, pages 133–138, May 1999.

[Sandholm and Lesser, 1995] Tuomas W. Sandholm and Victor Lesser. On automated contracting in multi-enterprise manufacturing. In *Distributed Enterprise: Advanced Systems and Tools*, pages 33–42, Edinburgh, Scotland, 1995.

[Sandholm *et al.*, 2001] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *Proc. of the 17th Joint Conf. on Artificial Intelligence*, Seattle, WA, USA, August 2001.

[Sandholm, 1996] Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents*. PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1996.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for winner determination in combinatorial auctions. In *Proc. of the 16th Joint Conf. on Artificial Intelligence*, pages 524–547, 1999.

[Sandholm, 2002] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.

[Schlenoff *et al.*, 1998] Craig Schlenoff, Rob Ivester, and Amy Knutilla. A robust process ontology for manufacturing systems integration. National Institute of Standards and Technology, 1998.

[Singh, 1999] Munindar P. Singh. An ontology for commitments in multiagent systems. towards a unification of normative concepts. *AI and Law*, 7(1):97–113, 1999.

[Smith, 1980] R. G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Trans. Computers*, 29(12):1104–1113, December 1980.

[Stone *et al.*, 2002] Peter Stone, Robert E. Schnapire, Micheal L. Littman Janos A. Csirik, and David McAllester. ATTac-2001: A learning, autonomous bidding agent. Submitted to the Eigtheenth National Conference on Artificial Intelligence (AAAI-2002), January 2002.

[Sycara and Pannu, 1998] Katia Sycara and Anandeep S. Pannu. The RETSINA multiagent system: towards integrating planning, execution, and information gathering. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 350–351, 1998.

[Sycara *et al.*, 1997] Katia Sycara, Keith Decker, and Mike Williamson. Middle-agents for the Internet. In *Proc. of the 15th Joint Conf. on Artificial Intelligence*, pages 578–583, 1997.

[Sycara *et al.*, 1999] Katia Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record (ACM Special Interests Group on Management of Data)*, 28(1):47–53, March 1999.

[Sycara, 1988] Katia Sycara. Resolving goal conflicts via negotiation. In *Proc. of the Nat'l Conf. on Artificial Intelligence*, pages 245–250, 1988.

[TAC-02, 2002] TAC-02. Trading agent competition 2002. http://www.sics.se/tac/, 2002.

[Tate, 1996] Austin Tate. Towards a plan ontology. *Journal of the Italian Artificial Intelligence Association*, January 1996.

[Tennenbaum *et al.*, 1997] J. M. Tennenbaum, T. S. Chowdhry, and K. Hughes. eCo System: CommerceNet's architectural framework for internet commerce. Technical report, Object Management Group, Cambridge, MA, 1997.

[Tsvetovatyy *et al.*, 1997] Maxsim Tsvetovatyy, Maria Gini, Bamshad Mobasher, and Zbigniew Wieckowski. MAGMA: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, 11(6):501–524, 1997.

[Varian and MacKie-Mason, 1995] Hal R. Varian and Jeffrey K. MacKie-Mason. Generalized vickrey auctions. Technical report, University of Michigan, 1995.

[Varian, 1995] Hal R. Varian. Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce*, New York, NY, July 1995.

[Vickrey, 1961] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[Walser, 1999] Joachim Paul Walser. *Integer Optimization by Local Search*, volume 1637 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1999.

[Walsh *et al.*, 2000] William E. Walsh, Michael Wellman, and Fredrik Ygge. Combinatorial auctions for supply chain formation. In *Proc. of ACM Conf on Electronic Commerce (EC'00)*, October 2000.

[Wellman and Wurman, 1998] Michael P. Wellman and Peter R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115–125, 1998.

[Wellman *et al.*, 2001] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[Wellman, 1993] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[Wilkins and Myers, 1998] David E. Wilkins and Karen L. Myers. A multiagent planning architecture. In *Proc. Int'l Conf. on AI Planning Systems*, pages 154–162, 1998.

[Wurman *et al.*, 1998] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Second Int'l Conf. on Autonomous Agents*, pages 301–308, May 1998.