1 ON LANGUAGES FOR DYNAMIC RESOURCE SCHEDULING PROBLEMS

Warren B. Powell

Abstract: The modeling of very complex problems requires the participation of people with diverse backgrounds. The combined efforts of these people, properly coordinated, can contribute to the solution of complex problems, but only if we facilitate the process of communicating. In this paper, we highlight the importance of languages, illustrated by a discussion of the "languages" used by four core groups of people. We show how the choice of language can either disguise similarities or hide important differences. Most significantly, we illustrate how the language we speak can color our view of the problem, leading, in some cases, to poor representations of a problem.

"The elevator lift is being fixed for the next day. During that time we regret that you will be unbearable." - Sign in a Bucharest business.

1.1 INTRODUCTION

Consider the situation of an American businessman early in the century who is interested in making money by selling Arab oil in the United States. The first problem is designing the chemical process needed to convert the type of oil in Saudi Arabia to the kind of gasoline needed in the U.S., a problem that has been solved by a French chemist. The American businessman needs to determine if global energy prices will rise to the point to make the enterprise profitable, a problem that is well understood by an English economist. The economist needs to understand how this new source of oil will affect the other oil markets, which can be solved by a Russian mathematician.

It turns out that the easiest part of this problem is translating American English to British English to Russian to French to Arabic. The general problem of translating between spoken languages has been resolved centuries ago. Much more intractable is the problem of translating between the fields of business, economics, mathematics, chemistry and the oil industry. Particularly in today's relatively specialized economy, it is difficult to get people with different training to understand the subtle languages of different fields. The American businessman wants to make money in a way that will show up on a corporate balance sheet, reflecting local tax laws. The economist might consider only the costs of production and shipping. The Russian mathematician might formulate a spatial price equilibrium problem that uses simplistic game theoretic models to describe global oil trade. The chemist will focus on the subtle differences in the quality of the oil and the nature of the production process needed to produce appropriate types of gasoline. And the Arab oil sheik wants to be sure that he is selling more oil than Libya.

We face a similar problem when we want to formulate real-time logistics problems as mathematical models. The benefits seem apparent - we want to lower our costs and make more money for the company. This requires formulating our problem as a mathematical model (the equivalent of the economist) which can then be solved using a mathematical algorithm, designed by mathematicians. The model and algorithm must be implemented by a software engineer (analogous to our chemist above) who needs to design code that can be used by someone in operations, with an efficient implementation of the mathematician's algorithm, and which provides the necessary diagnostics to be useful to the modeler.

The challenge of building a production-quality real-time control system for complex operations, such as arise in transportation, requires the participation of different groups of people who contribute different skill sets. At a high level, we can identify four major skill groups that need to contribute to the design and implementation of a real-time control system:

- Industry
- Modeling
- Algorithms
- Software

Within each of these groups are numerous subgroups, speaking what we view as dialects of a major language. Within industry, the major subgroups are: management, operations, marketing and human resources. Management thinks of itself as setting company strategy, although its impact on day to day operations is often minimal. Operations is the group that really controls the company on a day-to-day basis, and the biggest challenge is developing a system that is accepted by operations. A real-time dispatch system, however, invariably has an impact on customer service, which requires the participation of marketing. Finally, a dispatch system will impact both the people using the tool (dispatchers, who may need additional training) and the drivers themselves, requiring the input of human resources.

The modeler is an important, emerging group which we believe will become a distinct specialty. While industrial participants will reflect a business background

and algorithmic specialists will draw from mathematics, we believe that modelers will come from engineering and, as such, will evolve to become separate population. A good modeler will combine a knowledge of business processes and the mathematical theory of algorithms, his/her training will be more familiar to those from science and engineering, with a strong emphasis on experimental skills. A good modeler will focus on developing efficient mathematical models which capture the important physical laws of a process, balancing the benefits of modeling accuracy with the cost of collecting the necessary data.

Specialists in algorithms are primarily mathematicians working on well-defined problems. As a result of the variety of different problem structures, subcommunities of specialists in areas such as linear, nonlinear, and integer programming have arisen. Other important groups include stochastic programming and dynamic programming. As a rule, these groups speak a reasonably common language, with strong dialectical differences among the subgroups.

Finally, the software engineer has emerged as a critical player. Here too, we find important subgroups, covering the model (problem representation), algorithm, user interface, database, and communication.

Our decision to separate modeling and algorithms represents a departure from standard practice. Historically, professionals within operations research focused on the design and implementation of efficient algorithms, with an appreciation of the importance of modeling, but relatively little science in the process. Increasingly, modeling is taking on the characteristics of a classical experimental science, and the design and testing of models is starting to look like other activities within engineering.

The role of languages should be fundamental to the field of modeling, but its importance has only really emerged in the area of real-time logistics as models have tried to step up to the role of actually running a company. We believe that the modeling of real-time systems, or more precisely, *operational planning*, is fundamentally different than classical strategic models operating on a static dataset. There are two reasons for this claim. First, the primary user of a strategic model is rarely someone from operations; most commonly, management is using the model to make decisions about infrastructure or markets. Such an individual does not understand operations at a level to criticize some of the finer assumptions of a strategic model. In fact, most strategic models do not produce highly detailed outputs, making it impossible to even criticize some of the assumptions of the model.

The second reason is that a strategic model is out of *context*. Real-time operational decisions are made against a rich fabric of information that is available only at the time that the decision is being made. This information includes not only the data that is in the computer, but other information obtained through visual inspection and telephone conversations (often referred to as *head knowledge*). The decision maker will know the weather, the current business climate (e.g. should he be emphasizing cost or service?), and recent historical events ("I just have to serve this customer on time today - I missed his last appointment!"). He may even know that the data in the computer was entered by someone who is notoriously sloppy and not reliable ("The driver is supposed to be available at 3pm, but he is never on time."). This background of contextual information makes it difficult for even a knowledgeable operations manager to critique decisions after the fact.

Our desire to study languages is motivated by a desire, ultimately, to build better operational planning tools than are available today. While classical research into mathematical models and algorithms will continue, improvements in operational models will, in our view, come from two directions. First, we need a broader set of skills to be brought to bear on the problem. These skill sets can be broadly organized under the four headings described above, realizing that each heading may include people from different fields. Second, we need better information about the problem we are optimizing, which requires the more active participation of different groups from within the company. Both of these directions involve the participation of people from fundamentally different backgrounds, who invariably speak different languages. If we want to solicit broader participation, we need to articulate a language to describe the problems that we are working on.

The goal of this paper is to focus attention on the language issue, and provide structure to the discussion. We make the argument that an understanding of the languages will do more than raise our sensitivity to the issue, but will actually change the way we solve certain problems. Most importantly, we will seek to facilitate discussions between groups with complementary skill sets. Our focus is on large, complex problems, and the design and construction of production-caliber systems that will work in the real world. Thus, we take as a starting point that we will not be identifying problems with closed-form solutions, nor are we interested in relatively simpler problems where existing models and algorithms have proved satisfactory.

Our eventual goal is the development of an optimization environment that not only solves large, complex problems, but which is also transferable to other applications. We have seen the benefits of having general purpose solvers such as modern linear programming codes such as MINOS and Cplex; the value of these codes were significantly enhanced with the development of front-end interpreters such as GAMS and AMPL, which allow modelers with good math skills to "talk" to the solver through standard interfaces such as MPS-format files. This technology, however, is not up to the task of solving genuinely large, complex dynamic resource scheduling problems.

Our presentation begins in section 1.2 with a discussion of specific applications, which defines the scope of problems that we are trying to solve. Here, we list not only specific examples, but propose a structure for what a problem instance is composed of. Next, section 1.3 suggests a language for problem. This language, which is a form of taxonomy, allows the modeler to step away from the minutiae of specific problem instances and focus on the fundamental structure of a problem. Then, section 1.4 suggests a notational language for representing dynamic resource scheduling problems in an abstract way, that is the first step to identifying appropriate solution algorithms and coding the model on the computer. We make the argument that our mathematical formulation facilitates the communication between modelers, algorithmic specialists, software engineers, and industry. Section 1.5 addresses the representation of our mathematics in software. The software implementation also needs to address the interface with the mathematical algorithm, and the communication of the results back to the modeler and the user in the form of diagnostics. Finally, section 1.6 offers a strategy for implementing models in the context of very complex problems.

"We will take your bags and send them in all directions." - Airline office in Copenhagen.

1.2 THE LANGUAGE OF APPLICATIONS

The first step in our process is to developing an understanding of the types of applications we wish to consider. Section 1.2.1 starts by giving a list of specific examples. Section 1.2.2 discusses in more general terms the structure of application–specific languages, since our eventual goal is to develop tools that cut across different application areas. Finally, section 1.2.3 proposes a structure of an application, which forms the basis for our classification in section 1.3.

1.2.1 Examples

- Inventory distribution - Classical inventory problems involve moving units of inventory to anticipate future demands, balancing the cost of distributing product to distribution points such as warehouses.
- Short-haul routing and scheduling - Drivers need to be scheduled through a sequence of loads with real-time updates, taking into consideration work rules, driver attributes, and load attributes.
- Load matching for long-haul trucking - Long-haul truckload motor carriers have the problem of matching drivers to loads, where the loads typically take one to three days to move.
- Driver/load management over a linehaul relay network - A tactical planning system manages over 6000 drivers and 10,000 loads per week over a national linehaul relay network.
- Tactical management of rail flatcars - A major railroad must optimize the repositioning of its fleet of 10,000 flatcars to move intermodal trailers and containers over its rail network. There are over 50 types of flatcars, each holding between one and eight of the over 40 types of trailers and containers.
- Routing and scheduling for chemical distribution - A system has been implementing for designing driver routes and schedules to deliver chemicals to customers which use the product at varying rates. Some customers have small tanks, which need to be clustered with other tanks to fully utilize the vehicle .
- Pipe cutting - A pipe manufacturer keeps stocks of pipes of different lengths in different bins. If an order comes in for a pipe of length, say, 12 feet, and there are no pipes in inventory of this length, the manufacturer needs to decide whether to take a 14 foot pipe out of inventory, leaving a 2 foot piece of scrap, or a 20 foot pipe, leaving an 8 foot section that may still be useful.
- Personnel training - A medical manufacturing company needs to send a group of experts around the country to train hospitals in the use of a new diagnostic equipment. After a hospital purchases the equipment, it calls to arrange for training, often with short notice.

- Machine scheduling - Machines, which might be drilling or stamping presses or similar equipment, needs to be scheduled to handle a sequence of demands.
- Ambulances - An ambulance is a type of machine that needs to respond to most requests very quickly.
- Personnel planning - Most companies face the problem of assigning people to projects that fit their skills, anticipating the needs of future projects, and realizing the effect of a project on a person (positive, as in training and experience, and negative, if the project exerts high demands on the individual, such as travel and time away from home).

1.2.2 Languages

Each of the examples above represents an application with a distinct language. A specific application is generally characterized by certain characteristics that describe the resources being managed. An obvious question arises: when are two applications really different, and when are they structurally the same with only cosmetic differences? Is managing a fleet of taxis different than managing a fleet of trailers? Is truckload trucking really different than rail? Is making cars different than making oil? At one level, all of these problems are completely different. A truck would never be used to pick up a passenger at an airport. But we are modelers, and we are interested in how these problems differ in terms of our modeling approach.

Differences in vocabulary can even make two companies in the same industry look different. A long time ago, I had been working for a national less-than-truckload motor carrier, and started working for a direct competitor. One carrier moved freight out of satellites into "mother breaks," possibly incurring "linehaul variance" in order to make service. The competing carrier moved freight out of end-of-lines into "primary breaks," which possibly involved "moving air" in trucks in order to make service. The seemingly insignificant differences in vocabulary proved, initially, to be a major roadblock in my relationship with the new carrier.

Such contextual languages can magnify insignificant and meaningless differences. An alternative is to completely drop the language of the application and move to an entirely new vocabulary. For example, trucking, rail, ocean shipping and air freight are all industries that pose complex dynamic resource scheduling problems, where resources (trucks, boxcars, containers, aircraft) are resources that have to be scheduled to serve tasks (loads, shipments, boxes, customers). Such a vocabulary strips away the visual images that accentuate differences that contribute nothing to the actual structure of the problem. In the process, however, we discover that there are other differences that make boxcars different from truckload trailers, which are different from taxis.

These differences represent contextual data that is associated with each type of resource when placed in the setting of an application. One of the challenges of a language is making this contextual information explicit, so we can separate what is and is not important. At the same time, we need to appreciate the simplicity of saying "boxcar" instead of saying "a large reusable resource with relatively low value, which can serve a certain range of high volume products that generally have low service expectations, and which may enter and leave the system randomly."

1.2.3 Elements of an application

The first step in the development of a taxonomy is to describe the characteristics of a dynamic resource scheduling problem. A DRSP, in our view, is comprised of three fundamental elements:

- Resources - These are specific objects which need to be managed.
- Processes - Here we capture the physics of the problem, comprised of the laws that govern how the system evolves over time, and physical constraints that must be observed.
- Communication and control - Communication and control describes what decisions need to be made, who makes them, how information is provided to the decision maker, and how the decision maker separates good from bad decisions.

In the remainder of this section, we give examples of each of these. This structure becomes the basis of our taxonomy in section 1.3.

Resources

Examples of different resources include:

- People - Drivers, crews (e.g. pilots), doctors (and other specialists), airline passengers, low income commuters.
- Equipment - Aircraft, locomotives, tractors, trailers.
- Conduits - Highways, pipes, conveyor belts.
- Goods - Food, computers, fertilizer, cigarettes.

It has been common in other fields to separate the *demands* placed on the system (referred to as customers, jobs, or loads, among others), from the systems that serve these customers. In our representation, the distinction between a customer and a server (in the language of queueing theory) or a job and a machine (in the language of machine scheduling) is somewhat artificial. We do view jobs and machines as distinct resources that need to be managed, with different attributes, but we have not found it useful to put them into fundamentally different categories. This does not mean that we will not explicitly recognize an entity called a customer (or job, or load); in fact, we have found it useful to refer to an object we call a *task*. However, in terms of our problem categorization, and our eventual mathematical model, the separation between resources such as machines, trucks, planes and people, representing resources managed by a company, and tasks, representing resources that serve an external customer, proved to be artificial, and as a result, we group them together.

Processes

Processes describes the physics of a problem, the laws that are not up to a manager or dispatcher, but rather are determined by the technology of the process in question.

- Physical constraints - These constraints always apply to a process at a point in time.
- Dynamics - These are the laws that govern the evolution of the system over time.

Examples of physical constraints include:

- Cannot have two objects in the same place at the same time.
- A single object cannot be in two places at the same time.
- Rate of process transformation - This describes the speed with which a resource may be modified over time.

Communication and control
Elements of the communications and control structure include:

- Controls - What decisions are made.
- Control structure - Who makes each decision.
- Information structure - Who has access to what information, and when.
- Measurement and evaluation - How do we determine when one solution is better than another, and who is performing the evaluation?

Controls can be discrete (assign a driver to a load) or continuous (restrict the flow of a liquid from one container to another). The control structure captures the hierarchy of decision makers, and the information structure represents who has access to what information, and when.

The issue of measurement and evaluation is a particularly challenging one. Most companies want to maximize profits, but the process of doing this requires managing a range of individual elements that are more easily measurable. Examples of goals that commonly arise in this problem class might include:

- Operating revenue, costs and profits.
- Speed and reliability of service to the customer.
- Utilization of equipment and other physical assets.
- Employee happiness and productivity.
- Return on financial investment.

In any setting, the issue is not which one of these to worry about, but what weight to put on each measurement, since all of them are important.

1.2.4 Sources of information

An important aspect of an application is the *sources* of information. We have identified the elements of an application as consisting of resources, processes and controls. A fourth class of data might be called *context*, capturing the rich tapestry of information that is not explicitly tied to resources, processes or controls, but which which does nonetheless affect the decisions that are made. We view contextual data as a type of miscellaneous category, representing information that is not (at the moment) in the computer but which affects, usually in an ill-defined way, the decisions that are being made.

It is useful to briefly summarize the sources of data for each of our three major data classes:

- Resources - Most of the time, information on resources comes from computer databases, although it may be supplemented by other information (the computer tells you the aircraft is a Boeing 727, but a separate document tells you the speed of the aircraft). The process of implementing a model often forces the addition of data to a database that is important but which was never captured properly in the computer. In some cases, it is unavoidable that the knowledge of a resource exists only in someone's head, who probably got the information by looking out the window or talking on the phone (two forms of communication that will be outside the scope of a computer for the forseeable future).

- Processes - Information on processes generally comes from verbal descriptions from experienced operations professionals. Some information may come from the computer in terms of historical databases which capture past performance, such as travel times, frequency of machine breakdowns, and so on. Typically, information regarding processes must be accumulated over time, since it is virtually impossible for any group of people to fully describe all the physical operations of a complex process.

- Controls - One of the most difficult sources of information to collect is the nature of the control process. The easiest information to collect is who makes what decision, although these lines of responsibility are often blurred. The most difficult piece of information is the process by which we determine the quality of a solution. It is relatively easy to compare apples and apples, particularly when the apples represent hard, measurable dollars. Real decisions, however, have to consider other issues, such as:

  - How to trade off noncomparable quantities, such as cost, service to the customer, handling of employees, and the productivity of the equipment?
  - How to trade off the quality of the information being used to make a decision?

  It is, of course, both useful and important to simply ask decision makers how they trade off different dimensions of the problem. While informative, these discussions are rarely accurate or complete, highlighting the simple fact that most people do not know how they make these trade offs. In particular, they do not always like to admit some of the trade offs they have to make when faced with difficult choices. For this reason, the theory of revealed preference can be especially valuable. Historical datasets can be used to record what decisions were actually made (and under what circumstances). In the process, it may be possible to learn about biases and attitudes toward risk.

It is the diversity of sources of information that creates the emphasis in this paper on languages. The participation of different groups of people, who together contribute the definition of the problem we are solving, requires the development of languages that facilitate this communication.

"You are invited to take advantage of the chambermaid." - A Japanese hotel.

1.3 A REPRESENTATION LANGUAGE

The goal of a representation system is to provide a means of communication between specific problem instances, and the structure of the problem in a way that is meaningful to a modeler. A successful architecture should allow a modeler to capture all the important elements of a problem in an elegant, compact manner. Our goal in developing a representational system is to provide the basis for a flexible classification system which will allow us to organize problems into fundamental classes with different structures.

An important side benefit of a taxonomy is to identify problems which may be superficially very different, but which share a common structure. Over time, a modeler will learn which problem structures lend themselves to specific solution approaches, as well as to recognize when a problem falls in a class for which there is no good solution. In this sense, the modeler is playing the role of the physician looking at the symptoms of a patient, first to determine what disease the patient may have, and then to decide on a proper course of treatment.

Our goal is not to present a complete taxonomy; the complexity of problems that we consider in this paper make a full classification beyond the scope of this paper. However, we do feel it is useful to indicate the structure of a classification system, and indicate how this system can be used to facilitate the communication of results between application areas.

Classification systems have been used successfully in the past. The "$M/G/k$" paradigm for queueing systems, introduced by Kendall (see Gross and Harris **?** for a summary and references), served for decades to define the field of queueing theory. This framework actually takes the form "$A/B/X/Y/Z$" where A and B capture the arrival and service processes, while X, Y and Z represent the number of services, system capacity, and queue discipline. The machine scheduling literature uses "$\alpha|\beta|\gamma$ to represent, respectively, the machine environment, processing characteristics, and the number of machines (see Pinedo **?**). Eiselt *et al.* **?** propose a "I / II / III / IV / V" taxonomy for classifying location problems, and use this to organize the extensive literature in location theory.

Following the structure of section 1.2, we organize our representation system along three primary dimensions: resources, processes, and controls. We use the general style of queueing theory and machine scheduling, and propose that dynamic resource allocation problems be identified using the notation:

$$Resource||Process||Control$$

A complete specification of a taxonomy for this large problem class is beyond the scope of this paper, where our interest is more on highlighting the need for a classification system. In the remainder of the section, we highlight some of the key characteristics of these systems, followed by an illustration of how the general vocabulary of a classification system allows us to identify common properties of different problems.

1.3.1 Resources

The fundamental object that we are managing is called a resource, which can be machines, trucks, people, plants, or highways. We propose that some of the core characteristics of a resource can be divided along the following lines:

- Resource layering - The management of resources often requires the coupling of multiple types of resources in order to perform work, as in a driver and tractor pulling a trailer. In our view, a customer is a type of resource to be managed. Assigning a "job" to a "machine" represents a coupling of two resource layers to do "work."
- Resource attributes - Some resources are simple (a molding machine, a taxi, a trailer), some are more complex (aircraft, locomotives, industrial robot) while others are very complex (people). Some resources have static attributes (the molding machine) which do not change over time, while others have highly dynamic attributes that are always changing (people).
- Density of attribute space - We may have lots of resources with the same attributes (inventories of boxcars for a railroad) or lots of resources with very different attributes, so that there are rarely more than one or two with the same attributes.
- Resource availability - Resources may enter and leave the system according to different physical processes. Customers often enter the system randomly and leave when we are finished, while other resources leave of their own accord (machine breakdowns).
- Value of time - Different types of resources may have highly different time values, suggesting that the scheduling of one or more resources dominates the scheduling of others.

1.3.2 Processes

Processes describe the physical laws that govern the management of the resources. A driver can get in a tractor and drive it, but a trailer cannot move by itself. It is here that we specify the constraints and structure of the system.

- Resource bundling - It might be that more than one resource is needed at the same time (two pilots in an aircraft, three locomotives pull a train). In the same category, we might say that one truck can hold several shipments, or one taxi might carry several customers at the same time.
- Reward structure - Often, more than one resource is required to accomplish a specific goal (satisfying a customer, or performing maintenance on a machine). The cost structure describes whether rewards (or costs) are additive or are more complex functions.
- Arrivals (A) and departures (D) of resources to and from the system - Here we capture the rules governing the arrival of resources to the system, or their departure. These processes may be fixed or stochastic, exogenous or controlled.

- Transformation controls - We may have operational goals that restrict the rate at which we are willing to transform resources.
- Physical constraints - Finally, we have physical constraints that restrict processes, such as conservation of mass, or technological limits like the speed of a machine or size of a buffer.

1.3.3 Communication and control

The last dimension of our taxonomy captures the communication and control structure, the process by which we make and evaluate decisions. Elements of this aspect of a system include:

- Actions - Actions may be discrete or continuous. For discrete actions, we may characterize decisions at different levels. The lowest level might represent simple steps such as coupling, uncoupling and transforming. Higher levels capture tightly coordinated decisions or broader strategies.
- Information profile - This dimension captures the difference between when we know about random events, and when we have to make decisions involving these events. Systems where decisions are being made only as we learn about random events are highly dynamic and real-time, while other systems require decisions to be made well in advance, producing scheduled, predictable systems.
- Control structure - Complex systems are often controlled by different people, sometimes working at the same level of authority, or working at multiple levels.
- Evaluation - We have to represent the process of determining when one solution is better than another. Typically, we like to use a single-dimensional utility function such as costs or profits (or equivalent costs), but sometimes we need to more explicitly recognize multicriteria aspects, and possibly the presence of nonquantifiable qualities.

1.3.4 An illustration

Part of the value of a classification language is to identify similarities that are not immediately apparent on the surface. One example is a cutting stock problem. Consider a situation where you have bins of pipes of different lengths, ranging, say, from 1 to 20 feet. Orders come in randomly for lengths of pipe, and it is often necessary to take a longer length of pipe and cut it down. So, if we need a 12 foot length of pipe but do not have one, we can look at the different bins and choose something longer to cut down. We can choose a 14 foot piece, which will satisfy the demand and leave a two foot section as a leftover. Or, we can choose a 20 foot piece, leaving an eight foot section. It might be that there is a greater need for the eight foot section than the two foot, which might easily be discarded as waste.

While this problem sounds very different from a fleet management problem, in fact, the two are very similar when stated using similar terms. Both trucks and pipes are resources with different attributes. A truck in Chicago may be like the 14 foot section of pipe, while the truck in Cleveland is the 20 foot section of pipe. Assigning the 14 foot resource to the customer demand produces a two foot resource, just as assigning a

Chicago truck to a load going from Chicago to Atlanta produces a resource in Atlanta. Thus, in both cases, we are choosing a resource to handle a task, which changes the attributes of the resource. The choice of the best resource to use depends on the value of the resource that is produced after the task is completed. The vocabulary of pipes and trucks serves only the disguise the similar qualities of the two problems.

" I don't do 'x' " - Logistics manager.

1.4 A MATHEMATICAL LANGUAGE

Mathematics is the language through which we communicate with mathematicians who develop algorithms, and communicate to software engineers who represent our data and implement the algorithms. As a rule, modelers adopt the language of a particular algorithmic approach when representing a problem. When we wish to optimize a problem, we usually find ourselves working within the framework of:

$$\min c^T x$$

subject to:

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

This paradigm requires us to model decisions in the context of a vector x, goals and objectives must be captured in the cost vector c, and the physics of our underlying process must be represented in the system of linear equations $Ax = b$. The success of linear programming is an indication that there is a wide variety of problems for which this language is sufficient. For more complex problems, it is too hard to use.

Linear programming, of course, is not the only way to solve a problem. In fact, there are a series of modeling paradigms that have been developed, each of which has evolved its own distinct language. Examples include:

- Constrained optimization - the mathematics of coupled decisions.
- Control theory - constrained optimization over time.
- Simulation - the mathematics of working with realizations.
- Markov decision processes - the mathematics of optimizing with expectations.

Constrained optimization is an especially powerful tool, but it does not deal well with uncertainty and random outcomes, and tends to be focused on making a decision at a single point in time. Classical control theory addresses the problem of optimization over time, but normally in a deterministic setting with relatively simple physics governing the evolution of the process over time (e.g. linear equations). Simulation and stochastic processes handle random outcomes extremely well, but normally works with simple rules and policies to handle the problem of making decisions within the process, hence its success in manufacturing processes and queueing networks. Markov

decision processes embeds decision-making within a stochastic process, but depends on the language of state spaces and expectations, which generally cannot be used for large, complex problems.

A good analogy of mathematicians solving complex operational problems is the parable of the blind men and the elephant, represented in the poem by John Godfrey Saxe and reprinted in figure 1.4. Specialists in a particular modeling technique invariably see a problem in the language of their training. The phrase most often used to communicate this pattern is a "hammer looking for a nail." Anyone who has ever modeled a complex problem has been guilty of this; in short, all of us are limited by the languages we speak. Speaking a language, and interpreting the world through this language, is not in and of itself a problem, as long as we are aware of the potential limitations. If we are successful at solving a problem using a particular language, then we have nothing to be concerned about. On the other hand, if we are finding ourselves unsuccessful at solving a problem, we should be aware that the solution may involve a restatement of the problem in a different language.

It is common in the field of modeling to start with an idea of the solution algorithm we want to use, and then formulate a problem in the language of this algorithm. The reason for this practice is simple: it is easy to formulate a problem in a way that can not be solved with current technology. At the same time, the practice can severely distort our view of the problem, because we have formulated it in the language of a particular technology. It is the premise of this paper that a more fundamental mathematical representation is needed, one which is not tied to a particular algorithmic approach, but which offers the flexibility of using a variety of different approaches. Most important, we feel that we need a mathematical representation that can handle the complexity of real problems, without creating an overly complex mathematical language that disguises the fundamental structure of the problem.

In this section, we highlight some of the issues that arise when translating our problem into mathematical notation. In keeping with the theme of this paper, we need to focus on *why* we are doing this; more specifically, with whom do we propose to communicate with this abstract language? There are three sets of users of a mathematical representation of a problem. First are the modelers, who have to translate the original problem into mathematics. Second are the algorithmic specialists, who can identify search algorithms when the problem is recognized to have a specific algebraic structure. Third are the software engineers, who need to explicitly represent data internal to the computer in the form of variables. Interestingly, software engineers have identified weaknesses in standard mathematical conventions and have, out of necessity, invented new conventions that make software cleaner and more elegant. We propose to adopt some of these conventions in our own mathematical representation, in part to improve the clarity of our presentation, and in part to communicate more clearly with the software engineers who need to implement our expressions.

The goal of this section is to indicate that it is possible to develop a mathematical vocabulary that provides a more natural interface between the modeler, who needs to work with operations people from industry, and the more classical mathematical formulations around which algorithms are designed.

It was six men of Indostan
To learning much inclined,
Who went to see the elephant,
(Though all of them were blind).
That each by observation
Might satisfy his mind.

The First approached the elephant,
And, happening to fall,
Against his broad and sturdy side,
At once began to bawl:
"God bless me! but the elephant
Is nothing but a wall!"

The Second, feeling of the tusk,
Cried: "Ho! what have we here
So very round and smooth and sharp?
To me 'tis mighty clear
This wonder of an elephant
Is very like a spear!"

The third approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
"I see," quoth he, "the elephant
Is very like a snake!"

The Fourth reached out his eager hand,
And felt about the knee:
"What most this wondrous beast is like
Is mighty plain," quoth he
" 'Tis clear enough the elephant
Is very like a tree."

The fifth, who chanced to touch the ear,
Said "E'en the blindest man
Can tell what this resembles most;
Deny the fact who can,
This marvel of an elephant
Is very like a fan!"

The Sixth no sooner had begun
About the beast to grope,
Than, seizing on the swinging tail
That fell within his scope,
"I see," quoth he, "the elephant
Is very like a rope!"

So, oft in theologic wars
The disputants, I ween,
Rail on in utter ignorance
Of what each other mean,
And prate about an elephant
Not one of them has seen!

John Godfrey Saxe

Figure 1.1 The Blind Men and the Elephant

1.4.1 Resources

The foundation of our notation begins with a representation of the physical objects in the system, which we call resources. Our first step is to define the set of resources in the system at a point in time. For this, we define:

$\mathbf{a}_t$ = The vector of attributes of a resource at time t.
$\mathcal{A}$ = The space of possible values of $\mathbf{a}$

We need to provide for the presence of resource layering. This can be handled mathematically by defining subvectors and subspaces:

$\mathbf{a}_t^{(\ell)}$ = The vector of attributes of a resource in layer ℓ at time t.

$\mathcal{A}^{(\ell)}$ = The subspace of possible values of $\mathbf{a}^{(\ell)} \subseteq \mathcal{A}^{(\ell)} \subseteq \mathcal{A}$.

Hence we may write:

$$\mathcal{A} = \mathcal{A}^{(1)} \times \mathcal{A}^{(2)} \times \cdots \times \mathcal{A}^{(L)}$$

We use a superscript on $\mathbf{a}$ to indicate which subspace it belongs in. Thus, $a^{(1)} \in \mathcal{A}^{(1)}$, $a^{(2)} \in \mathcal{A}^{(2)}$ and $a^{(12)} \in \mathcal{A}^{(1)} \times \mathcal{A}^{(2)}$. All actions take place at some time t, so we sometimes drop the subscript t when there is no ambiguity.

We count the number of resources with a particular attribute using:

R_{at} = The number of resources in the system at time t with attribute $\mathbf{a}$

$R_t = \{\ldots, R_{at}, \ldots\}$

The vector R_t, then, combined with the space $\mathcal{A}$, captures the state of all the resources in our system at a particular time t. To solve a problem, we require as input R_0. This style, however, does not capture the availability of resources in the future in a natural way. For this reason, we generally find it convenient to let R_t^0 represent the vector of resources that first become available at time t. If we allow R_t^0 to be negative, then it represents exogenous changes in the availability of resources in the future.

This notation interfaces easily with standard databases, which store the status of each asset or resource and its attributes. It is the job of the modeler to determine which attributes are important, balancing realism with model complexity.

1.4.2 Controls

In classical optimization, decision variables are represented as x. In real applications, decisions are formulated in considerably different terms. For this reason, we suggest an approach that begins with a formalism that is much closer to a real application, and then demonstrate the relationship between this more natural formalism and the standard mathematical representation of a problem.

We begin by characterizing three mechanisms for making decisions:

- Elementary decisions, or *actions* - This is the most basic step required to change the state of the system.
- Compositions - These represent small groups of elementary decisions that are often coupled together (such as, "every time we do X, we also do Y").
- Strategies - A strategy represents a larger pattern of coordinated activities that influence, but do not directly control, elementary decisions.

Optimization models often represent only elementary decisions, and depend on the mathematics of the problem to devise composite decisions and strategies (of course, a decision variable may represent a composition of several elementary decisions). Thus, these are *outputs* rather than inputs. In our approach, we believe the user should be able to specify the presence of certain types of compositions and strategies, and let the mathematics work *with* them.

The first step is to formalize the representation of elementary decisions. We propose that any action (in the class of DRSP's) can be represented as consisting of one of three fundamental transformations to the system:

- Couple - Two resources (in the same or different layers) are combined into a single, more complex resource.
- Uncouple - A complex resource is broken down into its elementary parts.
- Transform - A single elementary resource is modified to a resource with different attributes (but in the same layer).

In any system, resources are typically transformed using a small set of decisions available to the user. These are normally expressed using words like *"move driver empty to. . .", "set up the machine for ...", "train a person to do ...", or "deliver product to the customer ..."*. Expressed in these terms, such sets of instructions are normally very small. Applied to specific situations, on the other hand, they grow dramatically. For this reason, we use the concept of a *context dependent* set, borrowing a concept from object-oriented programming called function overloading. For example, we might define:

$\mathcal{D}^e$ = Set of elementary decisions available to the decision maker, containing basic steps that would modify a process.

$\mathcal{D}^e(\mathbf{a})$ = The set of decisions associated with transforming a single resource with attribute $\mathbf{a}$.

$\mathcal{D}^e(\mathbf{a}^{(1)}, \mathbf{a}^{(2)})$ = The set of decisions associated with coupling two resources with attributes $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$.

We may need to capture the presence of multiple agents, at different levels in a hierarchy. Let $\mathcal{N}$ represent the set of decision makers. If we wish to capture a hierarchical ordering of decision makers, define the subset $\mathcal{N}_k \in \mathcal{N}$ as the set of decision makers at the k^{th} level. Then, we write:

$\mathcal{D}^{(n)}$ = The set of decisions that may be made by decision maker $n \in \mathcal{N}_k$, where the level k is predefined for a particular model.

To avoid ambiguity, we will normally model the sets $\mathcal{D}^{(n)}$ as being mutually exclusive and, of course, collectively exhaustive. In some complex operations, this may not be the case, but we would propose that it would still be best to model each decision as being "owned" by a particular position or individual.

We need to capture the impact of a decision d on the state of the system. For this we define:

$$\delta_{at'}(d_t) = \begin{cases} 1 & \text{if decision } d_t \text{ made at time } t \text{ increases the number of resources with attribute } a \text{ at time } t' \geq t \text{ by 1} \\ -1 & \text{if decision } d_t \text{ made at time } t \text{ decreases the number of resources with attribute } a \text{ at time } t' \geq t \text{ by 1} \\ 0 & \text{otherwise} \end{cases}$$

We also need to *count* how often a decision is made, which we do using:

$x_t(d)$ = the number of times decision d is executed at time t

Any optimization specialist will easily recognize $x_t(d)$ as a standard decision vector in classical math programming, while the elements $\delta_{at}(d)$ are the elements of a constraint

matrix. There is an important difference, however, in the modeling approaches. In the normal methodology of math programming, a modeler has to explicitly define the constraint coefficients $\delta_{at}(d)$ and the decision vector d, a process that may be simplified using interfaces such as AMPL or GAMS. In our approach, we ask the user to provide *elementary* decisions, such as are contained in the set $\mathcal{D}$, and then *derive* the resulting decision variables. The elements $\delta_{at}(d)$ are *endogenous* to the modeling process, not an input.

We have, at this point, represented elementary decisions using a new, highly flexible notation. It is important to realize, however, that while notation is an extremely important method of representing a problem, it is not the only, nor the best, method. Networks emerged in the 1970's as a popular modeling tool in part because they offered much faster solution times, and for certain problems, yielded integer solutions. But without question, one of the most widely appreciated dimensions of networks was that they were visual and, for many, much more accessible than algebraic representations. Of particular value was that a graphical depiction of a network was unambiguous, and represented a precise method of communicating the structure of a problem. The only limitation was that network diagrams could only be illustrative, given that it is impossible to actually draw networks with 50,000 arcs.

Our problem class also lends itself to a graphical depiction, even though our problems may not be pure networks. When we consider elementary decisions, it is clear that a resource can be represented as a node, and transformations as arcs. Coupling and uncoupling, on the other hand, requires a slightly richer visual vocabulary. In figure 1.2, we suggest a simple way of visually communicating the coupling of flows. We believe that a richer visual vocabulary can be developed to extend the scope of problems that can be covered in this way.

1.4.3 Processes

We present a mathematical model of two aspects of processes. The first is the representation of the transformation of resources as they are acted on. Then, we discuss the modeling of physical constraints.

Transformations. For a given decision $d \in \mathcal{D}^e(a)$ or $\mathcal{D}^e(a^{(1)}, a^{(2)})$ we identify an action that changes the system. Consider a resource transformation decision $d \in \mathcal{D}^e(a)$. As a result of this decision, we obtain a new resource with attributes a'. The process of making this transformation generates a cost or contribution to the system, which we represent by:

$c(d)$ = the cost or contribution to the system resulting from a decision d, where $c \in \mathcal{C}$

Furthermore, this transformation takes a certain amount of time, represented using:

$\tau(d)$ = the amount of time required to complete decision d, where $\tau \in \mathcal{T}$

We capture all of these changes in a single mapping function:

$$\mathcal{M} : \mathcal{A} \times \mathcal{D} \rightarrow \mathcal{A} \times \mathcal{C} \times \mathcal{T}$$

For the case of a resource transformation, $\mathcal{M}$ produces the ordered triplet:

$$\mathcal{M}(a, d) \rightarrow (\mathbf{a}', c, \tau) \tag{1.1}$$

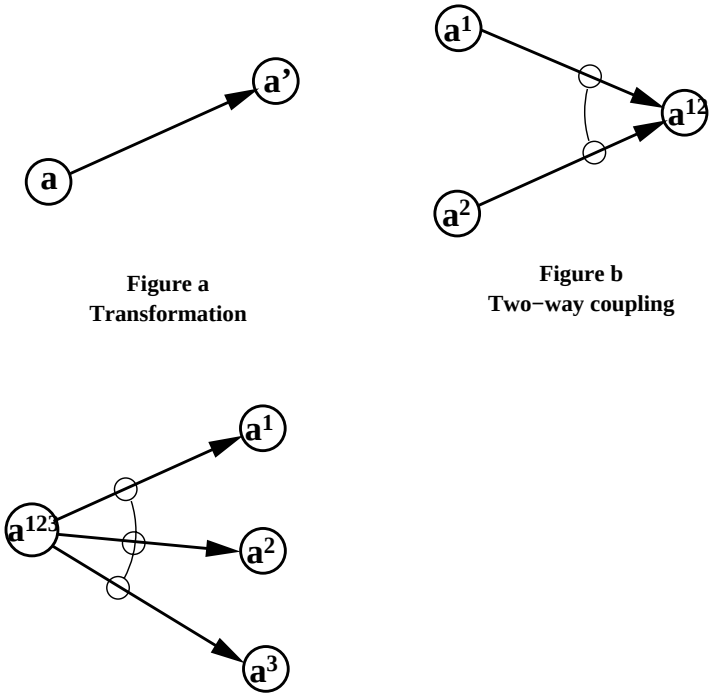

Figure 1.2 Visual depictions of decisions.

Using the concept of variable overloading, we can use the mapping $\mathcal{M}$ to handle other settings. In its most general form, it defines a mapping on a subset of resources with attributes $a \in \hat{\mathcal{A}}$:

$$\mathcal{M}(\hat{\mathcal{A}}, d) \rightarrow (\hat{\mathcal{A}}', c, \tau) \tag{1.2}$$

The strength of this representation is that it is closer to the form with which real problems are expressed. Given a resource vector R_{at}, and attribute space $\mathcal{A}$ and a transfer mapping $\mathcal{M}$, we can derive the constraints of a linear program.

Physical constraints. The standard math programming paradigm uses a set of constraints, often expressed as linear inequalities, that limit the set of feasible outcomes of a process. These constraints eliminate solutions that are both physically impossible, as well as those that are simply undesirable. In our view, we use the term physical constraint to eliminate the consideration of outcomes that are physically impossible, and use other means (in particular, the cost function c(d)), to limit the consideration of undesirable solutions.

We divide physical constraints into two broad groups:

- Physics - This captures basic laws governing conservation of mass, such as: a discrete resource cannot be in two places at the same time (conservation of resources); mass cannot be created or destroyed.
- Technology - These laws describe what can be accomplished within the limits of a particular technology, including the rate at which resources can be transformed (equation (1.1)), and what can feasibly be coupled or uncoupled. It is in this category that we would capture, for example, the size of a truck.

Conservation of mass may be represented using:

$$R_{at} - \sum_{d \in \mathcal{D}} x_t(d)\delta_{at}(d) \quad = \quad 0 \tag{1.3}$$

Equation (1.3) places a requirement that we make at least some decision about everything. We may equivalently use:

$$R_{at} - \sum_{d \in \mathcal{D}} x_t(d)\delta_{at}(d) \quad \geq \quad 0 \tag{1.4}$$

which allows us to "do nothing" to a particular resource. However, "doing nothing" is implicitly a decision, for which there must exist an appropriate mapping.

Technological restrictions come in two forms. First, they limit what resources can be coupled (and uncoupled). For example, you can couple a pilot with a jet, but you cannot couple two jets. These restrictions are best handled using a set of feasible couplings. The second restriction is in the rate at which resources are transformed. Consider a resource with attribute a; if we act on it with decision $d \in \mathcal{D}^e(a)$, then we produce a resource with attribute a'. We can represent the "flow" of resources from a to a' using:

$x_{a,a',t}$ = the number of resources with attribute a transformed to a' starting at time t

One way to capture technological limitations is to restrict this flow, using:

$$x_{a,a',t} \quad \leq \quad u_{a,a',t}$$

where:

$u_{a,a',t}$ = the maximum number of transformations of time $a \rightarrow a'$ that can occur at one point in time

1.4.4 Controls

We cover two topics under the category of controls. The first is the important issue of evaluation – How do we determine when one answer is better than another? Then, we present a metastrategy for solving large-scale, complex DRSP's.

Evaluation. One of the most difficult problems is evaluating a solution, or comparing two solutions to determine which one is better. For small problems, we may have well defined objective functions that reflect cost or service. As the problems become

larger and more complex, we need to explicitly recognize that the real objective function is complex and poorly specified, and that any objective function we choose is at best a surrogate for the real problem. For example, most companies cannot clearly specify the proper tradeoff between cost and service. Instead, they are constantly guessing at the right tradeoff, and then looking for indications that they are making the right guess.

We can capture this process by proposing that we are trying to solve:

$$\min_{d \in \mathcal{D}} F(d)$$

where $F(d)$ is an unknown function, representing, for example, expected corporate profits over some time horizon. Since we cannot specify $F(d)$, we propose instead to solve its surrogate:

$$\min_{d \in \mathcal{D}} G(d)$$

where $G(d)$ is a measurable, possibly multidimensional, function. A central part of the language issue is realizing that companies really want to minimize $F(d)$, but the best we can do is solve $G(d)$. This means that part of our optimization problem is finding the right function $G(d)$.

If we wish to capture the presence of multiple decision makers, we would write:

$$\min_{d \in \mathcal{D}^{(n)}} G(d)$$

to represent decisions made by decision maker $n \in \mathcal{N}$. When this happens, we open the realm of interactions between and competition among different individuals. It is easy to find situations where individuals cooperate, and others where they compete. Thus, game theory may play a central role in these problems.

The question now is, what is $G(d)$? In most situations, a decision maker works with a set of functions $G_i(x)$ that represent specific statistics that they calculate to manage their job. The difficulty is that most of the time, we ignore these "user defined" statistics. Instead, we impose our own cost function, which, most of the time, looks something like:

$$g_t(x_t) = \sum_{d \in \mathcal{D}} c_t(d) x_t(d) \tag{1.5}$$

We suggest that (1.5) should represent one of the elements that are considered, but that the other pet statistics of a decision maker should also be included. If we let $\mathcal{I}$ represent a set of such statistics, calculated using functions $g_i(x), i \in \mathcal{I}$, then we propose that these can be combined into a very general utility function. For example, we might define:

$$U(\mathbf{p}, g) = p_1(g - g_0) + p_2 e^{p_3(g-g_0)} \tag{1.6}$$

where $\mathbf{p} = (g_0, p_1, p_2, p_3)$ are parameters that control the shape of the function. Note that we use a special parameter g_0 that performs a scaling of g; we suggest that g_0 represents a target value for a particular quantity being measured. Now we may write:

$$g_t(\mathbf{w}, x_t) = \sum_{i \in \mathcal{I}} w_i U\left(g_{it}(x_t)\right) \tag{1.7}$$

where w_i is the weight given to dimension i, and $\mathbf{w}$ is the vector of weights. (In a multiagent environment, we would have a utility function $U^{(n)}$, weights $\mathbf{w}^{(n)}$ and a function $g_t^{(n)}(\mathbf{w}^{(n)})$ for each agent.) There is the modeling challenge of optimizing $G(\mathbf{w}, d)$ for a given $(\mathbf{w}, \mathbf{p})$, and the larger problem of determining $(\mathbf{w}, \mathbf{p})$, or more generally, determining the functional form of $g(\mathbf{w}, x)$ in (1.6) and (1.7).

Interestingly, for most companies, the functions g_i are quite well defined. It is only because our models require a cost function that operations research professionals tend to ignore these statistics, and focus instead on "our own" cost functions, which are often nothing more than different statistics. While we may argue that these are better in some way, they generally suffer from two limitations: first, they are new, so we do not have any sense what the numbers should look like (quick: what is the latest value of the S&P 500?) and second, we often cannot readily take a cost and translate it to an action I should be taking.

Finally, we have to reflect our desire to minimize costs over some horizon. Using a discount factor α to reflect the time value of money, we may write:

$$G(\mathbf{x}) = \sum_{t=0}^{T} U(\mathbf{p}, g_t(x_t))\alpha^t \tag{1.8}$$

More often, we need to explicitly recognize the presence of random elements, so we wish to solve:

$$G(\mathbf{x}) = E\left\{\sum_{t=0}^{T} U(\mathbf{p}, g_t(x_t))\alpha^t\right\}$$

Solution approach. The criticism of general "formulations" is that they are ultimately unsolvable, reflected in the common practice of starting with a solution technique and then trying to fit the problem into that approach. This process ensures that if the modeler can formulate a problem, there will be an algorithm available to solve it. This is especially true in the context of large, dynamic systems.

We propose an alternative, *metastrategy* for solving large, complex, dynamic systems. Our strategy is based in part on two observations:

- Greedy strategies generally work well in dynamic settings.
- Any large problem can be decomposed into problems that are sufficiently small that they can always be quickly and easily solved.

These observations are best applied to problems that are dynamic (which is the class of problems we are considering here), stochastic (virtually all dynamic problems are fundamentally stochastic) and with imperfect data (which applies to almost all real systems). In short, as the problem becomes messier, the solution strategies become simpler.

A measure of the success of this approach is given in a series of recent papers (Powell and Carvalho ?; ?). In ?, a large multicommodity network flow problem, with GUB constraints on certain arcs, was solved within three percent of an optimal linear programming relaxation, by solving nothing more than thousands of simple sorts. In Powell and Shapiro ?, an ultra-large scale dynamic crew scheduling problem was solved by iteratively solving 20,000 very small transportation problems.

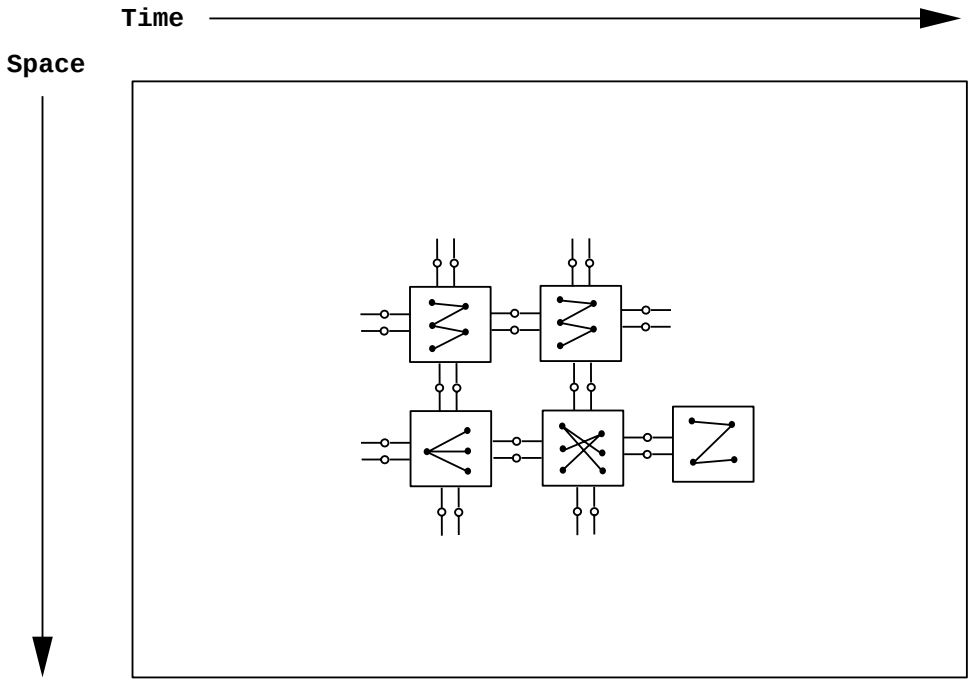

Figure 1.3 An illustrative decomposition.

We call this solution approach *massive dynamic decomposition*. It is fundamentally different than classical decomposition methods, which attempt to solve a single, large-scale problem to optimality. In our method, we solve sequences of very small subproblems, that are linked together in a special way. The concept is illustrated in figure 1.3. The large box represents the original optimization problem (1.8) that solves the problem over the entire attribute space $\mathcal{A}$, and over the entire planning horizon T. Each small box represents a specific optimization problem that is easily solved; in particular, we design these so that integer solutions are easy to obtain. Examples would be a simple sort, an assignment problem, a transportation problem or a small transshipment problem. They might also include small machine scheduling problems, or small set partitioning problems.

To develop our method, we begin by writing (1.8) recursively:

$$G_t(R_t) = \max_{x_t} g_t(x_t) + G_{t+1}(R_{t+1}) \tag{1.9}$$

subject to flow conservation, technology limitations and system dynamics, where we assume that R_t captures the state of the system. Because we do not know G_{t+1}, we replace it by an approximation that is adaptively estimated. Our approximation, which we denote $\hat{G}_t^k$ at iteration k, is defined recursively by:

$$\tilde{G}_t^k = \max_{x_t} g_t(x_t) + \hat{G}_{t+1}^k(R_{t+1}) \tag{1.10}$$

We further breakdown the problem by solving sequences over subsets of the attribute space. Let $\mathcal{A} = \mathcal{A}(1) \cup \mathcal{A}(2) \cup \ldots \cup \mathcal{A}(M)$ represent a partitioning of the attribute space. An appropriate partitioning might represent spatial boundaries or divisions of labor (different decision makers working at the same level of authority). For a given subspace $\mathcal{A}(m)$, we define the subvector $R_t(m)$ which counts the number of resources with each attribute in that subspace. We can now write:

$$\tilde{G}_t^k(m) \quad = \quad \max_{x_t} g_t(x_t, m) + \hat{G}_{t+1}^k(R_{t+1}) \tag{1.11}$$

subject to appropriately restricted versions of the constraints for flow conservation, technology and dynamics. Let $\pi_t^k(m)$ be the dual variable associated with the flow conservation constraint (1.3) for subproblem m. We may now use this dual to update our approximation $\hat{G}_t^k(R_{t+1})$ (note that this function is defined over the entire vector R_{t+1}, and not just over $R_{t+1}(m)$). Several methods can be used to accomplish this updating, depending on the specific form of approximation chosen for $\hat{G}$, hence we represent the updating abstractly using the mapping:

$$\mathcal{Q}(\hat{G}_t^k, \pi_t^k) \rightarrow \hat{G}_t^{k+1}$$

It is important to choose a functional form for $\hat{G}$ so that (1.11) is easy to solve (and yields integer solutions). Particularly nice functional forms are linear, which tends to be unstable, and piecewise linear, convex and separable. Both forms may result in (1.11) reducing to a pure network. If this is not the case, it may be necessary to solve an integer programming problem, but if the decomposition is chosen well, the problem is likely to be extremely small and easy to solve.

"Place orders early because in a big rush we will execute customers in strict rotation."
- A sign in a tailor shop

1.5 A SOFTWARE LANGUAGE

Ultimately, the entire modeling process is designed to transform a problem into a form that can be captured on the computer, a process that requires the translation of the problem to software. For this, we must speak the language of software engineers.

A software implementation of a problem must accomplish three core functions:

- Problem representation - It must capture the elements of any problem, as represented by the resources, processes and controls.
- Problem solution - It must be able to "solve" the problem, typically by interfacing with one or more optimization solvers.
- Communication and feedback - It must be able to communicate the results of the solution procedure in languages that make sense to the different users of the system.

The best phrase describing the computer representation of a problem is a "computer model." This tired and well-worn phrase is, somewhat unfortunately, appropriate;

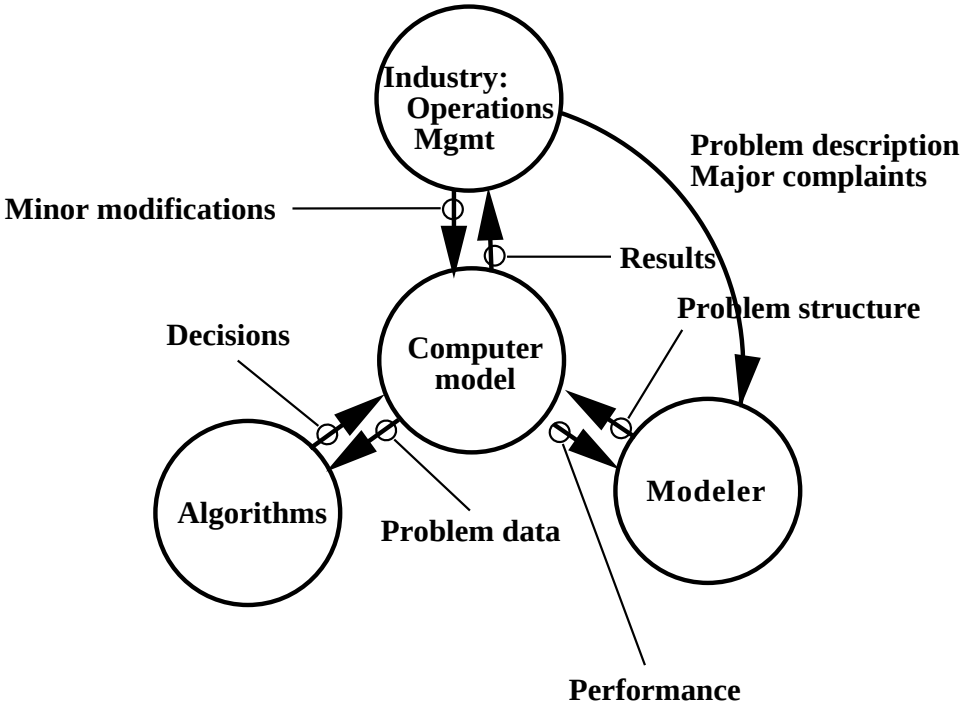

Figure 1.4 Lines of communication

unfortunately, because it is often misapplied. For many, the computer model and optimization solver are one and the same. In our view, the optimization solver (or solvers) are a part of a model, but are generally a small part. The more important part of a computer model is the representation of the problem itself.

Figure 1.4 provides a simplified depiction of the flow of information between different groups, covering "industry", modellers, software engineers, and mathematicians (algorithms). The challenge of software is to speak the languages of modellers, algorithms and industry. The major oversimplification of this simple illustration is the notion that industry can be represented using a single circle.

1.5.1 Problem representation

Problem representation, or, in the parlance of computer science, *domain knowledge encapsulation*, is the task of capturing the problem within software. Using our taxonomy, this means representing resources, processes and controls within software. This information should be captured in the form of software *objects*, each of which is comprised of certain data, and *methods* that act on that data. We can view resources, processes and controls as *classes*, which we may further subdivide in some cases into subclasses.

- Class: resources
  - Data:
    - Elements of the attribute space $\mathcal{A}^{(\ell)}$ for each layer ℓ.
    - Number of resources R_{at} of with attribute a at time t.
  - Methods:
    - Read and write routines.
    - Elementary assign, access and allocate routines.
- Class: processes
  - Data:
    - Various parameters required to describe physical and technological processes.
  - Methods:
    - The transfer function $\mathcal{M}$
    - The cost function $c(d)$
    - The time function τ
- Class: controls
  - Data:
    - Structure of the control process
    - Parameters governing the tradeoff of different dimensions of performance.
  - Methods:
    - Functions to rank two solutions - $G_t(x_t)$
    - Functions to provide the gradient of a solution with respect to specific parameters - $\nabla G(x)$.

Figure 1.5 Illustrative set of high-level object classes

The notion of subclasses is especially valuable in the context of describing resources. If a resource is a class, resources divided into different layers (such as people, airplanes, tractors and trailers) are subclasses. They share basic attributes such as being discrete objects in time and space, but otherwise are substantially different. Complex resources such as people and airplanes are probably conveniently divided into further subclasses.

The representation of resources as software objects is, by now, widespread. Schrijver **?**, for example, describes a class library for the load matching problem of truckload trucking. We imagine that there may easily be hundreds (if not thousands) of programming shops that are solving transportation and logistics problems using an object-oriented methodology. The designs of these class libraries are rarely published, so it is hard to assess the state of the field. Most importantly, this elegant and powerful way of writing software is now an industry standard, and represents the

"language" that software engineers speak. It is this reason that we have adopted an object style in our classification language, and in our mathematical notation.

1.5.2 Problem solution

The actual solution of our model is comprised of three steps:

- Design a suitable decomposition of the problem, considering the structure of the control system, the quality of the data, and the mathematical structure of the resulting subproblem.
- Identify the mathematical structure of the subproblem and an appropriate solver.
- Design and implement a computational strategy for solving the subproblems.

The first step is the most difficult, since it forces us to balance the capabilities of our library of solvers for subproblems with the organizational structure of the company, and the quality of the data. The designer needs to consider issues of diagnosability and data quality, which generally push for smaller decompositions, against solution quality, which generally encourages larger decompositions.

Given a particular decomposition, we must draw from our library of subproblem solvers. A simple library might include:

- Sort routines, hash tables
- Network solvers
  - Assignment problem
  - Transportation problem
  - Transshipment problem
- Specialized machine scheduling solvers
- Crew scheduling systems
- General linear programming
- General integer programming

Ferland *et al.* ? propose a class library of solvers for discrete optimization problems. The goal is to develop a class library of solvers that can be easily incorporated into specific applications. We support this concept, and suggest that the design of a general class library for representing DRSP's may enhance our ability to design optimization objects for solving them.

The last step is to design an computational strategy. The entire system might be implemented on a single processor, or spread among multiple processors on the same machine, different machines in the same location, or even spread around a series of machines located in different parts of the country. We propose using the strategy of massive dynamic decomposition to break large problems into large numbers of small problems. These small problems can be solved using our solver library, and if they are sufficiently small, there should be no significant computational problems. Furthermore, this solution technique will scale linearly with problem size, suggesting that we will not have difficulty solving large problems.

1.5.3 Communication

The last step is to communicate our results for evaluation and implementation. The choice of information to present depends, of course, on who we are presenting to. In our simple paradigm, the software needs to present results that are meaningful to three groups:

- Software engineer - Are there any bugs? Is the model being executed properly? Is it efficiently coded?
- Modeler - Is the system producing high quality solutions in a reasonable amount of time? Was the problem modeled correctly, in terms of making the right approximations, and using suitable decompositions? Are there errors in the data or the representation of the problem?
- Industry-operations - Is the system doing what we expect? Is the solution "high quality"? Is the real problem being represented correctly? Am I getting useful information? Can I explain why it is doing something?

Considerable expertise exists in the area of instrumentation to help software engineers. The bigger challenge for the modeling community is to address the combined questions of the modeler and industry representative. These questions can be broadly divided into four groups:

- Action - What should I do?
- Explanation - Why should I do it? The most frequently asked question is why *didn't* the system recommend a specific action?
- Information - What additional information can the system provide to help me make a decision on my own, over and above information that I already have?
- Evaluation - Are the recommendations valid? Is the quality of the solution high?

In systems today, the communication of a recommended action is usually only the starting point of a larger interrogation. If the recommendation is something that I find inherently reasonable (and probably what I was going to do anyway) then I will probably implement it and move on. The interesting recommendations are those which run counter to my normal patterns, and where perhaps I should be changing what I would do. It is in these recommendations that potential benefits lie, but there are also many pitfalls, since the recommendation may be wrong. Since *all* recommendations must be considered in the context of poor data and a misspecified model, counterintuitive recommendations must be viewed skeptically, and primarily as an invitation to more questions.

With an appreciation of *what* people want to see from a software system, we have the challenge of *how* to display it. We propose that information from a model can be displayed graphically and textually. In so doing we are excluding other modes of communication such as sound, smell, taste, touch and ESP, although we acknowledge the power of intuition.

It has been our experience that modelers, with a modest understanding of the problem, are most comfortable with graphical displays for communicating activities,

and selected high level performance statistics such as the objective function, CPU time, and other aggregate measures capturing cost and service. Others, especially those in operations, benefit from the ability of graphics to communicate the big picture and identify problem areas for further analysis. Then, they depend on a variety of selected statistics that they have used over the years to guide certain decisions. For example, just as many of us have become accustomed to looking at the Dow Jones Industrial Average as a measure of stock market performance, people in operations have their own set of pet statistics that they are comfortable with. It is our experience that experienced people in operations become extremely proficient using measures of performance that, on the surface, can seem confusing and uninformative to us.

Rather than criticize these measures because they are less meaningful (or worse) to modelers, we need to acknowledge the value of the experience operations professionals have gained in using these statistics. This is particularly true when we consider the different modes with which people in operations will evaluate a solution. Typically, users look at the results of a model dynamically in real-time or statically in a test setting. We can label these two modes as *out of context* and *in context*. Out–of–context evaluations are the easiest because we control the information available to the user. Much more realistic are in–context evaluations, where the user is looking at a recommendation at the same point in time that all the other information about the problem is available (including data not in the computer).

The most important guideline that emerges from this discussion is the importance of communicating results in a familiar format. Graphics is a powerful language because we all immediately respond to pictures. The only guidelines here concern the consistent use of shapes (wide lines mean more flow), motion (to indicate activity) and color (green is good, red is bad). Even more important is to consistently display certain activities in a certain way; experienced users will become accustomed to almost anything, as long as it is consistent. Other powerful visual cues involve the use of filters to highlight change (show me what is different). A human is exceptionally good at identifying changes if he is allowed to see the evolution from one solution to the next. At the same time, he is very poor at identifying changes when he is shown one solution, and then another, without any sense of what changed.

"Please do not feed the animals. If you have any suitable food, give it to the guard on duty." - Sign at a Budapest zoo.

1.6 A MODELING PROCESS

This paper proposes an approach for modeling complex problems, that takes as a starting point that the problem is too complex to describe with any accuracy, and where collecting (accurately) *all* of the data that *might* be useful is simply impractical. In particular, we propose the following starting axioms in the modeling process:

- Data is not correct, because humans are inherently tolerant of bad data.
- Operations people cannot describe what they are doing or why.

- Management does not understand their business, and cannot articulate with any accuracy how the operations people are solving the problem, or how they should solve the problem (but they believe they can).
- No single group of people can agree on what is best.
- Almost everyone can contribute an important piece of some part of the problem.

These problems demand an iterative process that involves guessing at many elements of the problem, and then using the performance of the system to refine our understanding of what the issues are, and what data is needed to support the system.

The basic steps in our paradigm are roughly as follows:

- Start with industrial application.
- Use the taxonomy to identify the problem class.
- Use the mathematical notation to represent problem abstractly.
- Design an appropriate decomposition that captures the important tradeoffs and which yields a solvable subproblem.
- Use software objects to implement the math.
- Apply an optimization engine to solve the problem class.
- Use graphical and textual outputs to communicate results back to the users.
- Continuously modify the process to improve the results.

This process is then imbedded within a larger process that starts with an initial formulation of the problem, and then iteratively improves our model of the problem. This process may sound obvious to most modelers, but we would claim that most modelers will try, as an initial design, to formulate the most accurate, precise model that seems practical given available data and the capabilities of our solvers. In our view, we should start with a much simpler model with the goal of getting *something* up and running and in front of the user. We propose to use the *simplest* model that gives meaningful results, even though we may be painfully aware of the limitations of our initial model.

A simple example illustrates this process. Consider the problem of assigning a resource (such as a driver or locomotive) to a task (a load or a train). It would be very reasonable to formulate this problem initially as a myopic assignment model, as illustrated in figure 1.6. This model seems simple and appealing, and easy to solve using standard network algorithms. However, when viewed in the context of a problem with bad data, our ability to solve this problem globally to an optimal solution is actually an impediment to the process of diagnosing the model and identifying the sources of bad data. A better approach is to implement a procedure where every driver is greedily assigned to the "best" load. Such an approach, which seems to have major weaknesses, is much easier to design and implement, and easier to diagnose. Only when the data improves should we use an algorithm that integrates more of the data using mechanisms such as dual variables.

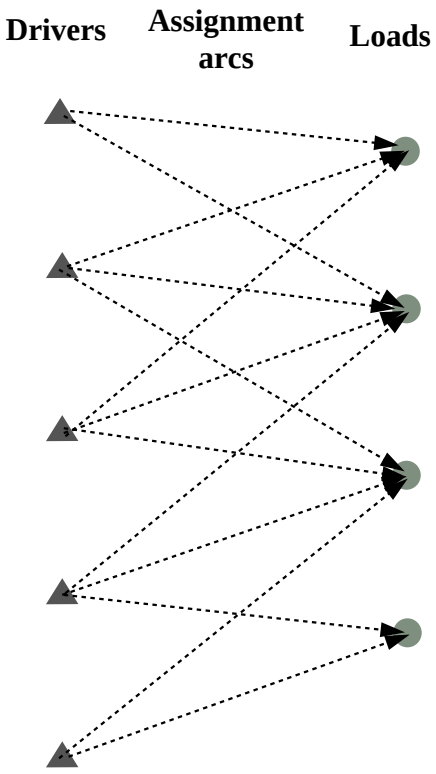

Figure 1.6 Global assignment network for assigning drivers to loads.

Once a simple model is up and running, it is important to focus on basics such as the accuracy of the data coming to the system and the accuracy of our modeling of the physics of the problem, and ignore in the first iterations the actual quality of the solution. In most cases, the solution will be poor, but not because of the algorithm. After iterating on the data requirements and the physical process, the time will come when the data appears to be right, but the answer is not very good, and a user can find better solutions than the computer using only the information available to the computer. This is the signal that an advancement in the search procedure is needed. This process then continues indefinitely (we assume that the problem is sufficiently complex that it can *always* be improved).

This strategy suggests the following process of implementing and improving a model:

- Start with the simplest possible model that represents a reasonable starting point to begin identifying data problems.
- Develop the model quickly to solicit user inputs as fast as possible.
- Design the software so that it is easy to change.

- Allow users to criticize specific actions in context.
- Respond to instance-specific criticisms.
- Adjust the model so that it fits within a well-defined, manageable work process.

In closing, we would claim that the paradigm suggested in this paper is evolutionary in nature, and integrates well-understood themes and concepts in the modeling community, but integrates them in a way that has not been proposed formally before.

"Ladies, leave your clothes here and spend the afternoon having a good time." A laundry in Rome.

References

H.A. Eiselt, G. Laporte, and J.-F. Thisse. Competitive location models: A framework and bibliography. *Transportation Science*, 27:44–54, 1993.

J.A. Ferland, A. Hertz, and A. Lavoie. An object-oriented methodology for solving assignment-type problems with neighborhood search techniques. *Operations Research*, 44:347–359, 1996.

D. Gross and C. Harris. *Fundamentals of Queueing Theory*. John Wiley and Sons, New York, 1985.

M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, New York, 1995.

W.B. Powell and T. Carvalho. Dynamic control of logistics queueing network for large-scale fleet management. SOR Report 96-01, Department of Civil Engineering and Operations Research, Princeton University, 1996.

W.B. Powell and T. Carvalho. Multicommodity logistics queueing networks. *European Journal of Operations Research*, 1996. (to appear).

W.B. Powell and J. A. Shapiro. A dynamic programming approximation for ultra large-scale dynamic resource allocation problems. SOR Report 96-06, Department of Civil Engineering and Operations Research, Princeton University, 1996.

P.R Schrijver. *Supporting Fleet Management by Mobile Communications*. P.R. Schrijver, The Netherlands, 1993.