INFORMS Journal on Computing
Vol. 18, No. 1, Winter 2006, pp. 43–60
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 1801 | 0043

DOI 10.1287/ijoc.1040.0111

A Metastrategy for Large-Scale Resource Management Based on Informational Decomposition

Joel A. Shapiro
Amazon.com, 1200 12th Avenue S., Suite 1200, Seattle, Washington 98144, USA, joel@alumni.princeton.edu

Warren B. Powell
Department of Operations Research and Financial Engineering, Princeton University,
Princeton, New Jersey 08544, USA, powell@princeton.edu

This paper addresses the solution of large, complex resource allocation problems, examples of which include large freight transportation companies and supply chain management. Some instances of these problems involve millions of constraints and tens of millions of variables. Classical formulations focus on modeling the physical problem alone. In this paper, we focus on modeling the organization of information and decisions, producing a natural decomposition based on how decisions are actually made. Restricting the size of a subproblem to the sizes of problems actually solved by real decision makers, we avoid the computational demands posed by large problems. The algorithmic challenge is producing high quality solutions that reflect the interaction between subproblems. Linear approximations have been a widely used tool for decomposition, but these can produce unstable solutions of only moderate quality. We introduce the concept of using nonlinear approximations, which creates special technical problems but also produces solutions of very high quality. The strategy is simulated on two problem classes (fleet management and supply chains) and compared against standard modeling strategies. Synchronous and asynchronous strategies are also compared.

Key words: multiagent; dynamic programming approximations; supply chain; transportation
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received July 1999; revised May 2001, June 2003; accepted July 2004.

1. Introduction

Large-scale, complex operational problems arise in a variety of settings. Railroads, airlines, and trucking companies routinely need to coordinate the flows of people and equipment over large networks. Manufacturing enterprises need to manage different elements of a supply chain. Attempts to model these problems using classical (usually deterministic) optimization methods have been successful only in isolated pockets (airline crew scheduling is a good example). The difficulty is that these problems are typically very large, usually exhibit integer variables, and almost always exhibit a variety of uncertainties.

Recently, Powell et al. (2001) introduced a flexible modeling paradigm designed to capture the complexities of large-scale operations. They formalize a problem class called the *dynamic resource transformation problem* (DRTP) with three primary dimensions: knowledge, processes, and controls. The modeling strategy emphasizes the representation of the organization and flow of information and decisions. However, a DRTP is a *model without an algorithm.* Since the problem class is so broad, it is not possible to offer a single algorithm. Instead, this paper suggests an *algorithmic metastrategy* that represents an approach that can be followed for a broad range of DRTPs. The focus of the strategy is to exploit the natural decomposition of complex problems into *informational subproblems* that capture how information and decisions are organized within a large operation. For example, large transportation companies are often regionally divided with an individual in charge of each region. In an operation such as a railroad, there might be one individual in charge of locomotives in one region, while another is in charge of a subset of boxcars for the entire country. Each of these individuals would constitute an informational subproblem. Thus, decision makers may be in the same company or in different organizations; they may be in the same place or spatially dispersed. There may be different people working on the same types of decisions in different regions, or on different types of decisions that affect the same region.

A challenge of any decomposition scheme is introducing a coordination strategy so that the individual components work together to mimic the results that might be achieved if the entire problem could be solved at once. The most popular scheme for achieving this is to use linear approximations to capture the impact of decisions on different subproblems (see, for example, Bertsekas and Tsitsiklis 1989 and the references cited there). While this is by far the easiest to implement, it can be unstable. We introduce the use of

nonlinear functional approximations, which produces much better results, but also introduces complications that we describe and resolve.

There are several key attractions of the metastrategy: (1) The subproblems are generally easy to solve (even hard integer programs can be easy to solve when they are small). (2) The method scales easily to ultra-large-scale problems, since the size of a subproblem is determined by how the problem is modeled (the organization of information and decisions is considered a part of the model, not a part of the algorithmic strategy). (3) The approach handles stochastic information in a simple and natural way. (4) The use of loosely coupled subproblems means that subproblems can be solved on different computers, without the fast communication requirements of traditional parallel computation. In fact, the subproblems may belong to different groups within a company, or operate across organizations.

A major contribution of this paper is the presentation of a methodology for handling nonlinear functional approximations. Using nonlinear functional approximations to coordinate different subproblems introduces special challenges that do not arise in the context of linear approximations (linear approximations include the entire class of algorithms based on Lagrangian relaxation). We can briefly illustrate the challenge using Figure 1, where we have four regions (A, B, C, and D) and four time periods (1–4). We are trying to capture the impact of different regions on region A at time period 4. Resources may be sent from region B at time period 1, arriving at time period 4, or from region C at the same time period (where we may solve region B before we solve region C). Or, we may encounter a region such as D that requires only two time periods to arrive at region A at time 4; as a result, we are interested in the impact of solving region D at time 2. Because the functional approximation is nonlinear, sending resources from region C, time 1 to region A (arriving at time 4) has an impact on the decisions made at region D, time 2. Sending an extra unit of flow from (C,1) to (A,4) may have an impact either on flows out of (A,4), or on other flows into (A,4).

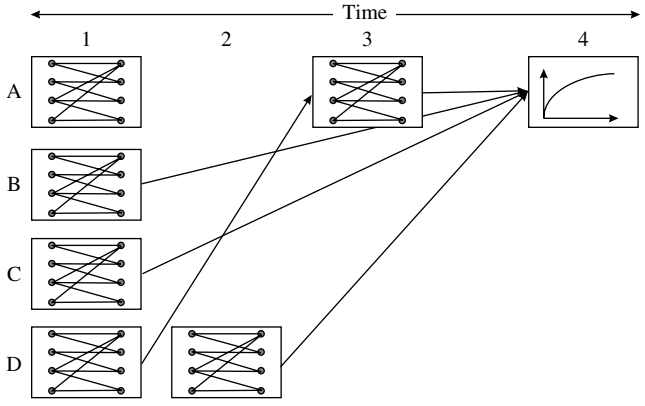

Figure 1 Coordinating Subproblems Through Nonlinear Functional Approximations

In our approach, we do not use a single nonlinear approximation for each subproblem, but a family of nonlinear approximations. We present the equations for updating these approximations, and show how well they work using problems that can be solved using classical tools. We also show that an apparent fix to the problem illustrated in Figure 1, which uses an estimate of the amount of flow into (A,4) from a closer subproblem such as (D,2), can work poorly.

This paper makes the following contributions. First, we introduce the concept of informational decomposition, where the decomposition of the problem into subproblems is a part of the specification of the model, not a part of the algorithmic strategy. The concept is similar to multiagent systems, but our representation of an informational subproblem is new, and we have applied it to problems in logistics that are classically approached as large-scale optimization problems. Second, we show how to represent the interaction between subproblems using nonlinear functional approximations that capture the impact of decisions of one subproblem on another. This strategy requires using a family of functions for each subproblem instead of a single approximation. This approach is new to the decomposition literature. Third, our metastrategy handles uncertainty without increasing the size of the problem since the effects of different scenarios are simply averaged into the functional approximations.

Finally, we study experimentally the performance of our algorithms on problems from freight transportation (which requires the movement of resources such as trucks and trains over a large area such as North America) and supply chain management. We compare our results against standard algorithms for these problems, not as a demonstration that we have a better algorithm, but to show that we can produce comparable results using a strategy that scales to much larger and harder problems (as well as one that easily handles uncertainty). The two problem classes are quite different, and also allow us to illustrate two very different views of communication between subproblems. We compare strategies for synchronous and asynchronous coordination of subproblems. We also study in some depth the complexities introduced by the use of nonlinear functional approximations. We show that a straightforward implementation of a nonlinear strategy works poorly, and that *obvious* approaches for fixing these problems do not work. We then introduce a specialized dual approximation strategy that works quite well.

Section 2 summarizes the mathematical notation required to model a DRTP. This representation emphasizes the organization and flow of information and

is central to the development of the metastrategy. Section 3 then explains our metastrategy and introduces the fundamental equation for linking subproblems. Following this, Section 4 suggests how to specialize the metastrategy to several important application classes. This is followed by Section 5, which discusses the impact of problem decomposition on the choice of value function. This Section in particular discusses the role of nonlinear approximations and gives the updating equations that motivate the use of a doubly-indexed functional approximation. Section 6 presents computational results to measure the effectiveness of the metastrategy under various decompositions, value function approximations, and problem classes. Our computational work is restricted to problem classes where we can also find optimal solutions using classical methods. This experimental work is intended only to provide a measure of the accuracy of our approach. The real value is its scalability; we do not attempt to solve any ultra-large-scale problems, nor any problems that involve stochastic elements, simply because we are not able to obtain tight bounds on performance. Finally, Section 7 concludes with directions for future research, including distributed computation.

2. A Mathematical Representation

In this section we briefly outline the mathematical notation for modeling a DRTP. A more comprehensive review is presented in Powell et al. (2001). What is most important about our representation, for the purposes of this paper, is the explicit modeling of information, the evolution of information, and the organization of information within the control structure.

Any DRTP can be classified along three principal axes.

1. Knowledge: This includes what we know about resources being managed (which may include drivers, loads, product, customers, and demands) as well as parameters that govern the behavior of the system.

2. Processes: The "physics" of the system. Processes include system dynamics (how the system evolves over time), physical constraints on the system, as well as the nature of the information stream arriving to the system.

3. Controls: The decision-making process. This covers how decisions are made.

Each of the following three subsections deals with one of these items. The presentation of the model is limited to what is needed to represent the organization and flow of information and decisions.

2.1. Knowledge

Our knowledge about our system can be divided into two groups: the resources that we are managing and parameters that govern the process being managed. Resources are represented using

$\mathscr{C}^R$ = the set of resource classes (e.g., drivers, trucks, product types, ...),

$\mathscr{R}$ = the set of resources in class $c \in \mathscr{C}^R$,

a_r = the attribute vector of a resource $r \in \mathscr{R}^c$,

$\mathscr{A}^c$ = the set of possible vector-values for attribute vectors in layer $c \in \mathscr{C}^R$,

R_{ta}^c = the number of resources with attributes $a \in \mathscr{A}^c$ at time t,

$R_t = (R_{ta}^c)_{c \in \mathscr{C}^R,\, a \in \mathscr{A}^c}$.

We represent parameters simply using

ρ_t = vector of parameters that govern the dynamics of the system.

Combined, we represent what we know about the system at time t using

$$K_t = (R_t, \rho_t).$$

As an example, take a freight application. In driver-scheduling problems, an important resource is a human driver whose specific *attributes* describe his state at any point in time:

$$a = \begin{bmatrix} a_{\text{actionable}} \\ a_{\text{location}} \\ a_{\text{domicile}} \\ a_{\text{duty\_hours}} \end{bmatrix} = \begin{bmatrix} \text{time} \\ \text{location of the driver} \\ \text{domicile} \\ \text{duty hours} \end{bmatrix}. \quad (1)$$

The element $a_{\text{actionable}}$ is special, capturing the time at which the rest of the vector a becomes *implementable*. Thus, representing a resource with attribute a implicitly includes the time dimension. This compact representation will simplify notation later.

Resources in a DRTP usually evolve over time through sequences of couplings (putting two or more resources together) and uncouplings (taking them apart). For example, we might have a^P be the attributes of a pilot and a^A be the attributes of an aircraft. The attribute vector a^A would include the information needed to determine whether the aircraft needs one pilot or two before it can fly.

2.2. Processes

In this section, we summarize the processes that govern the dynamics of the system: exogenous information processes, decisions, and system dynamics.

2.2.1. Exogenous Information Processes. Most DRTPs are not solved with static data. Instead, they face dynamic data that change over time through a sequence of information updates. How a model handles this sequence of information updates is a critical determinant of its success in any field implementation. Our principal interest is to investigate how a model can handle information updates generated by

sources outside the system:

κ_t = the set of all the information elements arriving to the system in time period t.

Assume that we are modeling over a planning horizon $\mathcal{T}^{ph}$. The information $\kappa_t, t \in \mathcal{T}^{ph}$ represents forecasts of future events. Since the future may be uncertain, there may be more than one set of potential future events. Following classical notation, we let

ω = a potential sequence of $\kappa_t, t \in \mathcal{T}^{ph}$,

Ω = the set of all potential outcomes over $\mathcal{T}^{ph}$.

As information κ_t arrives, we update our database, to which we refer as the *knowledge base* of the system. We assume that there is a function U^K for updating the knowledge base so that

$$K_{t+1} = U^K(K_t, \kappa_{t+1}). \tag{2}$$

Such an iterative updating scheme requires K_0 as input.

For the remainder of this paper, we consider only dynamic updates of resources. All other information is assumed static. Updates to the resource vector are represented using

$\hat{R}_{ta}$ = the change in the number of resources with attribute vector a due to exogenous information arriving at time t.

For the remainder of the paper, we let $\hat{R}_{ta}$ represent exogenous changes to our resource vector, while R_{ta} represents resources that are in state a at time t as a result of previous endogenous changes.

2.2.2. Decisions. Specifying the types of controls in a DRTP determines the makeup of the umbrella set $\mathcal{D}$ of decisions that we have been using up to this point. The elements of this set are decisions to which we refer as *primal controls*. Decisions can generally be divided into specific classes based on domain-specific classification schema. Accordingly, we define

$\mathcal{C}^D$ = the set of decision classes,

$\mathcal{D}^c$ = the (finite) set of possible decisions d in class $c \in \mathcal{C}^D$,

$\mathcal{D}_a^c$ = the subset of $\mathcal{D}^c$ that can be applied to a resource with attribute a.

In our freight example, we may define $\mathcal{C}^D = \{\text{move}, \text{sleep}\}$ and $\mathcal{D}^{\text{move}} = \{\text{move\_to\_Chicago}\}$. Also as a matter of computational convenience, sets like $\mathcal{D}_a^c$ allow us to specify a set of feasible decisions contingent on the state of the resource a. As a matter of mathematical necessity, however, we require that $\bigcup_{c \in \mathcal{C}^D} \mathcal{D}_a^c$ be nonempty for every a, containing at least the sentinel decision d^ϕ. The interpretation of d^ϕ is domain-specific but is generally meant to represent the "do-nothing" option. For instance, in a freight context, d^ϕ generally requires a driver to sit idle at his current location in space.

For accounting purposes, we need to represent the number of times a decision is executed:

x_{tad} = the number of times decision d is applied to resource a at time t.

Recall that one of the elements of the attribute vector is $a_{\text{actionable}}$. We can use this element to determine the time at which an action takes place, allowing us to use x_{ad} instead of x_{tad}. We assume that $a_{\text{actionable}}$ refers to a point in time within our planning horizon, defined by

$\mathcal{T}^{ph}$ = set of time instances within our planning horizon.

Thus, we consider only those attributes a where $a_{\text{actionable}} \in \mathcal{T}^{ph}$.

2.2.3. System Dynamics. The changes induced by a decision d can be elegantly modeled using the *modify function* M. Given an attribute vector a, the application of the control d modifies the system as in

$$M(a, d, K_t) \mapsto (a', c, \tau), \tag{3}$$

where

a' = the attribute vector of resource a *after* control d is applied to it,

c = the cost/benefit resulting from the modification,

τ = the time required to complete the modification.

We need to consider the impact of a decision d on other parts of the system. We capture this using the *δ-function*

$\delta_{a'}(a, d, K_t)$ = the impact on resource a' if decision d is implemented on resource a,

$$= \begin{cases} 1 & \text{if } M(a, d, K_t) \mapsto (a', \cdot, \cdot), \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the $\delta_{a'}$ function captures the effect of the endogenous decision d on the system.

2.3. Controls

All that remains to be defined is the last dimension of our DRTP paradigm, controls, which comprises the following three elements.

1. Control structure: Who can make what types of decisions, and with what information?

2. The decision function: How are decisions made, and with what information?

3. Measurement and evaluation: What is the economic impact of implementing a control?

2.3.1. Control Structure. We represent our control structure as a set of subproblems denoted by

$\mathcal{Q}$ = set of subproblems that encompass the problem.

There are three dimensions to a subproblem: the attribute subspace, which determines what resources are being controlled; a subset of decision classes, which determines what types of decisions belong to a subproblem; and a set of time periods over which the

subproblem has control. These are represented using:

$\mathcal{A}_q$ = subset of the attribute space for subproblem q, where $\bigcup_{q\in\mathcal{Q}} \mathcal{A}_q = \mathcal{A}$ and $\mathcal{A}_{q_1} \cap \mathcal{A}_{q_2} = \varnothing$ when $q_1 \neq q_2$. Implicit in the definition of the attribute space is:

$\mathcal{T}_q^{ih}$ = the *implementation horizon* for subproblem q. This is the set of time periods during which subproblem q controls the decisions. Since time is a dimension of the attribute vector, we may state that $a \in \mathcal{A}_q \Rightarrow a_{\text{actionable}} \in \mathcal{T}_q^{ih}$.

$\mathcal{C}_q^D$ = set of control classes associated with subproblem q. As a rule, a subproblem is formulated for a specific type of decision, so the set $\mathcal{C}_q^D$ is implicit in the formulation of the problem.

The set of decisions in subproblem q can now be expressed by

$$\mathcal{D}_q = \text{subset of decisions in subproblem } q = \{d \in \mathcal{D}_a^c,\ c \in \mathcal{C}_q^D,\ a \in \mathcal{A}_q,\ a_{\text{actionable}} \in \mathcal{T}_q^{ih}\}.$$

We require that the subsets $\{\mathcal{A}_q\}$, $\{\mathcal{D}_q\}$, and $\{\mathcal{T}_q^{ih}\}$ be, in their respective dimensions, mutually exclusive and collectively exhaustive.

Later we will need to capture the impact of decisions in one subproblem on another. For this, we define the following.

Definition 2.1. The forward-reachable set $\vec{\mathcal{M}}_q$ of subproblem q is the set of subproblems q' with resource states $a' \in \mathcal{A}_{q'}$ that can be reached by implementing a single, feasible decision d on at least one state $a \in \mathcal{A}_q$. More precisely, the forward-reachable set of subproblem q is

$$\vec{\mathcal{M}}_q = \{q' \in \mathcal{Q}\backslash q \mid \exists\, a \in \mathcal{A}_q, d \in \mathcal{D}_q \text{ where } M(a, d, \cdot) \mapsto (a', \cdot, \cdot), a' \in \mathcal{A}_{q'}\}. \quad (4)$$

Definition 2.2. The backward-reachable set $\overleftarrow{\mathcal{M}}_q$ of subproblem q is the set of all subproblems for which subproblem q is forward-reachable. More precisely, the backward-reachable set of subproblem q is

$$\overleftarrow{\mathcal{M}}_q = \{q' \in \mathcal{Q}\backslash q \mid q \in \vec{\mathcal{M}}_{q'}\}. \quad (5)$$

It is important to observe that the definition of the forward-and backward-reachable sets implies a master/slave relationship between subproblems. Later we provide two very different illustrations of this relationship. In Section 4 we describe a resource allocation problem where the master sends resources to the slave, and a multistage lot-sizing problem where the master requests resources from the slave.

2.3.2. The Decision Function. Earlier we introduced the decision variable x_{ad}, which is the number of times that decision d is applied to resources of type a. We now define

$X_q^\pi(I_q)$ = a function that determines x_{ad} for $a \in \mathcal{A}_q$, $d \in \mathcal{D}_q$ where:

Π = a set of different possible decision functions, where $\pi \in \Pi$,

I_q = the information content of subproblem q (for policy π, which we suppress for simplicity).

The information set I_q (which we can think of as the "IQ" of our subproblem) is composed of three basic classes of information: what we know now, what we forecast to become known in the future, and the impact of decisions made in subproblem q on other subproblems. As before, what we know now can be divided into what we know about resources and what we know about other problem parameters (costs, speeds, times, and other physical parameters). Following our earlier notation, we let

K_q = the information in subproblem q $= \{R_q, \rho_q\}$,

ω_q = updates to the set K_q.

A myopic decision function uses $I_q = K_q$. Rolling-horizon models use $I_q = (K_q, \Omega_q)$. If $|\Omega_q| = 1$, then we are using a deterministic forecast; if $|\Omega_q| > 1$, then we have a stochastic model.

In this paper, we assume that all the information in a subproblem becomes available at the same time. Thus, an element $\omega_q \in \Omega_q$ would be a sample of all the information that would be used in subproblem q. In stochastic models, it is often convenient to assume that $|\mathcal{T}_q^{ih}| = 1$; when this is not the case, we assume that all the information for the subproblem becomes known at once.

Using this assumption, we can introduce notation to capture the property that the modify function $M(a, d, K_q)$ is known given ω_q. This means that the outcome $a'(\omega)$ is known given (a, d, K_q). It is convenient, then, to introduce the notation

$$x_{aa'}(\omega_q) = \sum_{d\in\mathcal{D}_q} x_{ad}(\omega_q)\delta_{a'}(a, d, K_q),$$
$$x_{qa'}(\omega_q) = \{x_{aa'}(\omega_q), a \in \mathcal{A}_q\},$$
$$x_{qq'}(\omega_q) = \{x_{qa'}, a' \in \mathcal{A}_{q'}\},$$
$$x_q(\omega_q) = \{x_{qq'}(\omega_q), q' \in \vec{\mathcal{M}}_q\}.$$

Thus, $x_{qq'}(\omega_q)$ is the vector of all the flows from subproblem q to q'; $x_q(\omega)$ is the vector of all the flows out of subproblem q.

2.3.3. The Objective Function. We define the objective function of a subproblem q as C_q. This is simply the cost generated by a decision, as returned by

the modify function. The total costs generated by subproblem q are given by

$$C_q(x_q, \omega_q) = \sum_{a \in \mathcal{A}_q} \sum_{d \in \mathcal{D}_q} c_{ad}(x_{ad}, \omega_q).$$

We assume that new information arriving to subproblem q, represented by ω_q, is known before we make decisions in this subproblem. This information may affect costs, new arrivals, and upper bounds. If subproblem q encompasses points in time in the future, we feel that this is a reasonable approximation. If subproblem q captures "here and now" then we are assuming that ω_q is a deterministic, short-term forecast, which may be updated as new information arrives that affects q.

If we were to ignore the impact of our decisions on other subproblems we could solve a sequence of local subproblems given by the following definition.

Definition 2.3. The local subproblem LSP_q for subproblem q is the following system:

$$\min_{x_q(\omega_q)} C_q(x_q, \omega_q) \tag{6}$$

subject to:

$$\sum_{a' \in \mathcal{A}_q} x_{aa'}(\omega_q) - \sum_{a' \in \mathcal{A}_q} x_{a'a}(\omega_q) = R_a + \hat{R}_a(\omega_q) \quad \forall a \in \mathcal{A}_q \tag{7}$$

$$x_{aa'}(\omega_q) \le u_{aa'}(\omega_q) \quad \forall a, a' \in \mathcal{A}_q \tag{8}$$

$$x_{aa'}(\omega_q) \ge 0 \quad \forall a, a' \in \mathcal{A}_q \tag{9}$$

where C_q is the objective function of subproblem q.

Note that our knowledge base, K_q, is also a function of ω_q by virtue of our updating scheme. The constraints (7) represent flow conservation. For later time periods, R_a is determined by decisions made in earlier time periods.

Problem LSP_q encompasses a rich array of problems. The underlying problem may be a simple matching of drivers to loads; it may be a set-partitioning problem involving the assignment of a pilot to a sequence of flights; it could include the coupling of drivers, tractors, and trailers, which then have to pick up goods for delivery to a set of customers. We do not claim that LSP_q is a simple problem, but we do assume that it is relatively small. Generally, we assume that it is no larger than what a human is probably solving by hand. Most importantly, the size of a subproblem is limited by the amount of information available to the subproblem at a given point in time.

3. The Metastrategy

The metastrategy requires that we generalize the system (6)–(9) defining LSP_q to one that considers the impact of decisions in subproblem q on other subproblems. The optimization problem for subproblem q can be rewritten as the global subproblem GSP_q. The major difference between LSP_q and GSP_q is that the global subproblem captures the effect of decisions in subproblem q on other subproblems. We may now define

Definition 3.1. The global subproblem $\text{GSP}_\mathbf{q}$ for subproblem q is the following system:

$$\tilde{V}_q(R_q) = E\left\{ \max_{x_q} C_q(x_q, \omega_q) + \sum_{q' \in \vec{\mathcal{M}}_q} \bar{V}_{qq'}(x_{qq'} \mid \omega_q) \right\} \tag{10}$$

where $\bar{V}_{qq'}$ is a family of functional approximations that capture the impact of decisions made by subproblem q on subproblem q'. Equation (10) must be solved subject to the local constraints (7)–(9) and equations that handle flow conservation between subproblems:

$$R_a = \sum_{q' \in \overleftarrow{\mathcal{M}}_q} x_{q'a} \quad a \in \mathcal{A}_q. \tag{11}$$

Constraint (11) defines the flows from one subproblem to the next. All the remaining constraints are buried in (7)–(9), which are part of the subproblem.

In special cases, we can compute the expectation in (10) exactly (examples can be found in Frantzeskakis and Powell 1990 and Powell and Cheung 1994). However, general problems will require choosing a sample $\omega_q \in \Omega_q$ and then solving $\text{GSP}_q(\omega_q)$ for a sample realization. In this case, we would drop the expectation and represent the solution as $\tilde{V}_q(\omega_q)$. For the remainder of the paper, we assume that a sampling-based strategy is required.

Remark. Equation (10) is the heart of our metastrategy. There are obvious parallels with dynamic programming (the recursive computation of value functions). The most obvious difference is the double indexing (qq') of the value function. The use of a doubly-indexed functional approximation in the context of multiagent control is new (but motivated by the work on multiperiod travel times in Godfrey and Powell 2002b) and is required only when we use nonlinear functional approximations for the value function. We show in Section 5 how these functions are updated, and in particular the step that produces the doubly-indexed approximation.

The design of the function $\bar{V}_{qq'}(x_{qq'} \mid \omega_q)$ is essential to the metastrategy, which is intended to approximate the impact of subproblem q on subproblem q'. The challenge is to find an approximation that keeps GSP_q tractable, but provides an accurate approximation of the impact of one subproblem on another. The function $\tilde{V}_q$ is partly a placeholder that we use to capture information about subproblem q. This information (which might be a function estimate, or a gradient) is then used to update our approximation for subproblem q. We represent this updating process using the mapping

$$\bar{V}_q \leftarrow U^V(\tilde{V}_q(\omega_q), \bar{V}_q). \tag{12}$$

An important dimension of the metastrategy is the order in which certain operations take place. In particular, we have some flexibility in designing the order

in which subproblems are solved, and the order in which value functions are updated. We assume that implicit in the set $\mathcal{Q}$ is an ordering of subproblems, so that $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$. In addition, let

$$\mathcal{V} = \text{the set of value function approximations} = \{\bar{V}_1, \bar{V}_2, \ldots, \bar{V}_{|\mathcal{V}|}\}.$$

We assume throughout that subproblems are solved in the order specified in $\mathcal{Q}$, while value functions are updated in the order specified in $\mathcal{V}$. In the simplest updating strategy, we would update $\bar{V}_q$ at the same time that we solve $\tilde{V}_q$, but this will not generally be the case. In fact, we could consider strategies where the orderings change as the algorithm progresses, implying an ordering $\mathcal{Q}^k$ at iteration k.

Our metastrategy, then, is a high-level mathematical framework for developing specific solution strategies for specific DRTP problem instances. To use the metastrategy on a specific problem instance, the modeler must perform the following two tasks.

1. Specialize the metastrategy: Fill in the four-tuple $(Q, \bar{V}_q, X^\pi, U^V)$.
2. Execute the metastrategy: Iteratively solve GSP_q until a stopping criterion is met.

Specializing the Metastrategy. The steps involved in specializing the metastrategy involve (i) identifying an appropriate decomposition $\mathcal{Q}$; (ii) designing a suitable value-function approximation $\bar{V}$; (iii) developing the decision function (both the model and an algorithm to solve it), and (iv) designing the updating strategy U^V.

Executing the Metastrategy. Because we cannot generally "guess" the correct parameters for $\bar{V}_q$, we solve GSP_q iteratively from a reasonable initial solution, updating our guess using U^V at each iteration, as outlined in Figure 2. The termination criteria are left to the modeler's discretion. Note that in online modeling situations, the termination check is skipped.

Section 4 presents two illustrations of the metastrategy on important practical problems. Examples of decompositions include decomposition by time (temporal), time and space (temporo-spatial), and by resource layer (layered). Different forms of $\bar{V}$ are explored. Common forms include linear and separable, nonlinear approximations. The decision function X^π ranges in complexity from simple sort procedures to Wagner-Whitin (1958) dynamic programming to the full network simplex. Stochastic linearization as in Ermoliev (1988) may be sufficient for U^V. In general, by smoothing $\bar{V}_q$ in the dual space we do not require as frequent dual updates as do other decomposition and value function approximation techniques (e.g., Bertsekas and Tsitsiklis 1996).

Generally, the forward-reachable sets $\vec{\mathcal{M}}_q$ control the construction and ordering of $\mathcal{Q}$. Similarly, ordering the updating of $\bar{V}_q$ is generally controlled by each subproblem's backward-reachable set $\overleftarrow{\mathcal{M}}_q$. However, different strategies can be devised. Section 6.2 develops and tests a number of policies for constructing the functions $\bar{V}$ and ordering the set $\mathcal{Q}$.

STEP 0: Initialize $\bar{V}_q$ for all subproblems $q \in \mathcal{Q}$.
STEP 1: Choose a $q \in \mathcal{Q}$ following a predetermined ordering $\{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$:
  STEP 1a: Choose a sample $\omega_q \in \Omega_q$.
  STEP 1b: Solve GSP_q.
STEP 2: Update the value function approximations, $\{\bar{V}_q: q \in \mathcal{Q}\}$ using U^V.
STEP 3: Terminate if stopping criterion is satisfied. Otherwise go to STEP1.

Figure 2 Steps in the Metastrategy

4. Applying the Metastrategy to Important Problem Classes

In this section we show how the metastrategy can be applied to two important and very different classes of DRTPs: fleet management and multistage dynamic lot-sizing. Section 4.1 decomposes a large-scale fleet management problem using nonlinear functional approximations to capture the impact of decisions on the rest of the system. Nonlinear functional approximations introduce special challenges that have never been addressed before. Then, Section 4.2 introduces a completely different problem class (multistage lot-sizing), which involves a different type of problem structure (nonconcave but monotone value functions), as well as a different type of master/slave structure that determines the construction of the forward-reachable sets.

4.1. Fleet Management by Nonlinear Approximation

Fleet management problems have been studied for decades as applications of optimization, starting with early models for freight car distribution (see Dejax and Crainic 1987 for an excellent review of this literature). These models typically focus on formulating the fleet assignment problem as a single large linear (or integer) program. Recent work (for example, Powell and Carvalho 1998, Godfrey and Powell 2002a) suggests strategies for decomposing problems over time, which facilitates incorporating uncertainty. In the case of problems arising in freight transportation (major truckload and LTL carriers, large railroads, and container shipping companies), these models still fail to capture the organization of decisions. For example, it is common for companies to divide responsibility into smaller regions. We can capture this division of responsibility using our notation, which specifies who (which subproblem q) makes what decisions (specified in the set $\mathcal{D}_q$) and to what resources these are

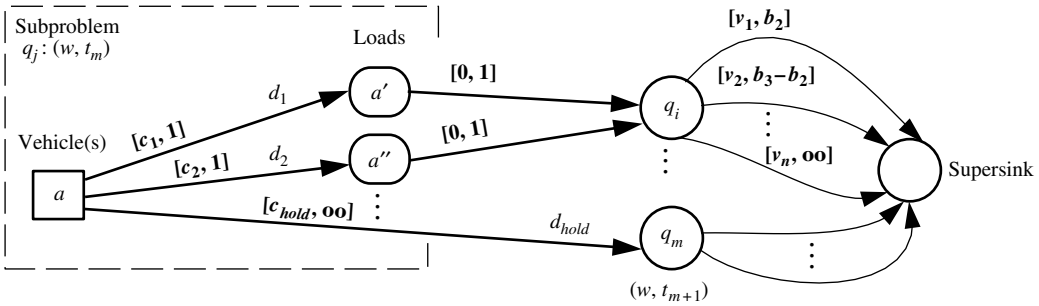

Figure 3 **Fleet Management Subproblem q_j with Piecewise Linear $\bar{V}_{q_i}$**

applied (defined by resources with attributes in the set $\mathcal{A}_q$).

The specialization, $(Q, \bar{V}_{q_i}, X^\pi, U^{\bar{V}}) = (\mathcal{W} \times \mathcal{T}$, linear, sort, exponential smoothing) has been shown to work well for moderate and large-scale problems with real-world data (Powell and Carvalho 1998). However, linear approximations can be unstable, and will not provide high quality results for all problems. An alternative is to use a nonlinear approximation, which may be a continuously differentiable function if integrality is not an issue, or a piecewise linear function if integer solutions are needed. In the latter case, we might write $\bar{V}_q(R_q)$ as

$$\bar{V}_q(r) = \sum_{m=1}^{n} v_m 1_{\{m \le r\}},$$

where $1_{\{X\}} = 1$ if X is "true", and v_m is the incremental value of the mth unit of flow. Such a function is useful when we can guarantee concavity, where we would require $v_m \ge v_{m+1}$. Normally, we divide the functions into segments, where values of r between, say, b_m and b_{m+1} carry the same marginal cost.

When we use a nonlinear function, problem GSP_q can be reduced to a pure network (this is especially useful if integrality is an issue). A sample network is illustrated in Figure 3. The supply for node a is $r_a \ge 0$. The cost c and upper bound u of each arc are shown as the pair $[c, u]$ in bold above the arc. Each decision is modeled as a separate arc. Of particular interest is the network representation of $\bar{V}_{q_i}$: a single node is used to represent $\bar{V}_{q_i}$ for each $q_i \in \vec{\mathcal{M}}_q$. The n outbound arcs from node q_i each capture a single segment of $\bar{V}_{q_i}$, where the arc labeled $[v_k, b_{k+1} - b_k]$ offers the cost v_k to $b_{k+1} - b_k$ units of flow. Because we require $v_{k-1} \ge v_k$, flow will enter arc $[v_k, b_{k+1} - b_k]$ only if arc $[v_{k-1}, b_k - b_{k-1}]$ is saturated.

4.2. Multistage Dynamic Lot Sizing

In this section we consider the classical multistage lot-sizing problem. The lot-sizing problem is interesting in part because it is an example of a nonconcave problem (as a result of the integer variables). Also, it provides an illustration of the forward-reachable set, which is quite different from the other applications.

The dynamic lot-sizing problem arises in the context of an inventory system that handles N different types of items. The goal of controlling such a system is to determine the replenishment quantities for each item that satisfy demand requirements over some finite, discrete time horizon $\mathcal{T} = \{1, \ldots, T\}$ at minimum system cost. We are given the following basic data:

c_i^s = setup cost for item i,
c_i^h = per-period holding cost for item i (charged against end-of-period inventory),
c_{ti}^v = variable production cost for item i at time t, and
$\beta_{i,i-1}$ = number of units of item i required for each unit of item $i-1$ (for $i > 1$).

To demonstrate how we can solve the multistage lot-sizing problem using our metastrategy, we must first formulate it as a DRTP.

We begin with layer-one resources, to which we refer to as "items." Each item has a "type." The inventory system has N distinct stages, each stage i capable of storing only items of type i, which means that the physical location of the item is synonymous with its type. The attribute vector of an item is quite simple, taking the form

$$a = \begin{bmatrix} a_{\text{actionable}} \\ a_{\text{type}} \end{bmatrix}, \tag{13}$$

where $a_{\text{actionable}} \in \mathcal{T}$ and $a_{\text{type}} \in \{1, \ldots, N\}$. The count of items at each stage i is recorded as

R_{ti}^1 = the inventory of items of type i at the end of period t.

All units in inventory start off as items of type N. An item of type i can proceed to its successor stage $i-1$ only by undergoing some transformation at stage i (e.g., assembly or machining). Items at stage i are subjected to demand at each time period t, which must be satisfied without backordering.

Demands are represented as layer-two resources, which we call "orders." The attribute vector for an

order is also simple, taking the form

$$a = \begin{bmatrix} a_{\text{actionable}} \\ a_{\text{type}} \\ a_{\text{source}} \end{bmatrix}, \tag{14}$$

where $a_{\text{actionable}}$ and a_{type} have the same range as for item attribute vectors and a_{source} indicates whether the demand for item i stems from an exogenous source (external customer demand) or an endogenous source (internal demand from a successor stage). Exogenous demand is denoted as

$$\hat{R}^2_{ti} = \text{the count of exogenous demand for items of type } i \text{ at the end of period } t.$$

In our representation, ω_{q_i} would represent a realization of the exogenous demands $\{\hat{R}_{1i}, \hat{R}_{2i}, \ldots, \hat{R}_{Ti}\}$. In our approach, we assume that all the realizations for a subproblem become known at once. If we wanted to model the staging of information more carefully, we would have to create individual subproblems for each time period as well.

Endogenous demand is denoted as

$$R^2_{ti} = \text{the count of endogenous demand for items of type } i \text{ at the end of period } t,$$

where $R^2_{t1} = 0$ for all t since there is no successor to stage 1. Since we consider only serial assembly systems, endogenous demand represents orders from stage $i-1$.

A natural way to decompose the original problem into subproblems is according to stage. In other words, $\mathcal{Q} = \{q_1, \ldots, q_N\}$, where each subproblem q_i corresponds to stage i, *across the entire time horizon* $\mathcal{T}$. The decision problem for subproblem q_i consists of the vector of production decisions $x_{q_i} = \{x_{1i}, x_{2i}, \ldots, x_{ti}, \ldots, x_{Ti}\}$. Because no backordering is allowed, we require $R^1_{ti} \geq 0$. We assume zero production lead time and $\hat{R}^2_{ti} = R^1_{ti} = 0$ for $i = 1, \ldots, N$ and $t \leq 0$. Stage $N+1$ can be thought of as an infinite supply of raw materials. The role of the decision function X^π at each stage i at time t is to choose x_{ti}, the number of items of type i to produce (or buy). Feasibility at stages $1 \leq i \leq N$ requires that we satisfy demand $\hat{R}^2_{ti}$ fully, so that the system dynamics defining R^1_{ti} are

$$R^1_{ti} = \begin{cases} R^1_{t-1,i} + x_{ti} - \hat{R}^2_{ti} - \beta_{i,i-1} x_{t,i-1} & \text{if } 1 < i \leq N, \\ R^1_{t-1,i} + x_{ti} - \hat{R}^2_{ti} & \text{otherwise.} \end{cases} \tag{15}$$

Since producing a unit at stage $i-1$ implies sending an order for $\beta_{i,i-1}$ units to stage i, we have to enforce the requirement

$$R^2_{ti} = \beta_{i,i-1} x_{t,i-1}. \tag{16}$$

We may then write

$$\tilde{R}^2_{ti} = \hat{R}^2_{ti} + R^2_{ti} \tag{17}$$

as the "modified demand." Consequently, we can rewrite (15) as

$$R^1_{ti} = R^1_{t-1,i} - \tilde{R}^2_{ti} + x_{ti}. \tag{18}$$

Let $X^\pi_{q_i}$ be the decision function for subproblem q_i. This function determines how to satisfy the demands of subproblem q_{i-1} and sends orders to subproblem q_{i+1}. In our earlier applications, the forward-reachable set was the set of subproblems to which resources (vehicles, drivers) were sent. In this application, the forward-reachable set of subproblem q_i is q_{i+1}, which is the subproblem to which we are sending *orders*. This application, then, provides a nice illustration of the master/slave relationship between a subproblem and the forward-reachable set. So, $\vec{\mathcal{M}}_{q_i} = q_{i+1}$. The reason q_{i-1} is not in the forward-reachable set of q_i is because we are sending items from q_i to q_{i-1} in response to a *request* for items made by q_{i-1}. Thus, we see an instance of the master/slave relationship implied by the forward-reachable set. Subproblem q_i is placing a demand on q_{i+1}, which makes q_i the master and q_{i+1} the slave. Note that our formulation assumes that we satisfy the entire order placed by q_{i-1}; if this were not the case, we would be sending a flow of missed requests from q_i to q_{i-1}, implying that both q_{i+1} and q_{i-1} would be in the forward-reachable set of q_i.

In classical MRP logic, the production decisions in stage i ignore their impact on stage $i+1$. If we ignored the impact of decisions in stage i on stage $i+1$, the problem could be solved optimally using a minor variant of Wagner-Whitin (1958) developed by Eppen et al. (1969) ("modified Wagner-Whitin") to handle time-variant instead of constant marginal production costs c^v_{ti}. Let $WW(\tilde{R}^2_{q_i}, c^s_{q_i}, c^h_{q_i}, c^v_{q_i} \mid \omega_{q_i})$ denote a Wagner-Whitin procedure for subproblem q_i using demands $\tilde{R}^2_{q_i}$, and setup, holding, and variable production costs of $c^s_{q_i}$, $c^h_{q_i}$, and $c^v_{q_i}$, respectively. We also express the conditioning on ω_{q_i} to emphasize the presence of sampling. Our decision function can be written as the mapping $X^\pi : WW(\tilde{r}^2_{q_i}, c^s_{q_i}, c^h_{q_i}, c^v_{q_i} \mid \omega_{q_i}) \mapsto x_{q_i}(\omega_{q_i})$.

It is well known that dynamic programming works well for single-stage (and single-commodity) problems but does not generalize to multiple stages (Zangwill 1969). For this purpose, we turn to our metastrategy. The procedure we outline here was originally proposed by Graves (1981), purely in the context of deterministic problems with a single item per stage and zero travel times. Our development makes it much clearer that the method could be extended to more general problems with uncertainties in demands, multiple items per stage, and multiperiod travel times. For example, Papadaki and Powell (2003) show, in the context of a batch-service problem, that the use

of functional approximations actually produces better results when the underlying problem is stochastic (uncertainty has the effect of smoothing out the function that captures the impact of decisions on other subproblems).

Applying the metastrategy, we will make decisions in stage i using $\bar{V}_q, q \in \vec{\mathscr{M}}_{q_i}$ to capture the impact of decisions in q_i on other subproblems. In the simplest case, q_{i+1} is a different company, in which case $\bar{V}_{q_{i+1}}$ is simply the pricing structure of the enterprise represented by q_{i+1}. When it is a different factory within the same company, then we have access to the production process, and can develop our own approximation. Here, we consider only a simple linear approximation of the form $\bar{V}_q = v_q x_q$ where both v_q and x_q are vectors defined over the planning horizon. The real function is neither concave nor convex, but it is monotone (Puterman 1994 and Papadaki and Powell 2002), which suggests that a linear approximation might work well. Papadaki and Powell (2003) show that linear approximations do, in fact, work quite well for batch-service problems, especially when the underlying problem is stochastic.

In contrast to the earlier examples, which were all linear programs where slopes could be obtained from dual information, this problem involves discrete setup variables. We could approximate a slope using finite differences. A more elegant alternative suggested by Graves (1981) is to assume that a unit change in a requirement will not result in a change in any setups. In this case, the marginal impact of an incremental demand is easy to calculate. For example, assume we wish to estimate $v_{t,i+1}$, which is the cost of an incremental demand in stage $i+1$ at time t. Let t'_{i+1} be the last time period prior to t in which there is production of item $i+1$:

$$t'_{i+1} = \max_{t' \in \mathscr{T}} \{t' \mid x_{it'} > 0\}. \tag{19}$$

Then

$$\begin{aligned} v_{t,i+1} &= c^v_{t'_{i+1},i+1} + (t - t'_{i+1})c^h_{i+1} \\ &= \text{marginal cost of increasing } R^{1k}_{t,i+1} \text{ by} \\ &\qquad \text{one unit.} \end{aligned} \tag{20}$$

The use of a linear approximation for the value function creates a very easy subproblem. Recall that c^v_{ti} is the variable production cost for stage i at time t. If we produce an additional unit x_{ti}, then we incur a variable production cost c^v_{ti}, and we induce a demand $\beta_{i+1,i}$ on stage $i+1$. Using our value function approximation, the cost of this is $v_{i+1,t}$ per unit of item $i+1$. We can now define a modified variable production cost for stage i using

$$\tilde{c}^v_{ti} = \begin{cases} c^v_{ti} + \beta_{i+1,i} v_{t,i+1} & \text{if } 1 \le i < N, \\ c^v_{ti} & \text{otherwise.} \end{cases} \tag{21}$$

STEP 0: Let $\tilde{c}^v_{ti} = c_{ti}$ and $v_{ti} = 0$ for $i = 1, \ldots, N$ and $t \in \mathscr{T}$.
STEP 1: For $i = 1, \ldots, N$ do:
  STEP 1.1: Sample the exogenous demands ω_{q_i}.
  STEP 1.2: For $t \in \mathscr{T}$ compute $R^2_{ti} = \beta_{i,i-1} x_{t,i-1}$ and $\tilde{R}^2_{ti} = \hat{R}^2_{ti} + R^2_{ti}$.
  STEP 1.3: Solve $WW(\tilde{R}^2_{q_i}, c^s_{q_i}, c^h_{q_i}, \tilde{c}^v_{q_i} \mid \omega_{q_i})$ for $x_{q_i}(\omega_{q_i})$.
  STEP 1.4: For $t \in \mathscr{T}$ update $R^1_{ti} = R^1_{t-1,i} - \tilde{R}^2_{ti} + x_{ti}$.
STEP 2: For $i = N-1, \ldots, 1$ and $t = 1, \ldots, T$ set v_{ti} by (20) and then $\tilde{c}^v_{ti}$ by (21).
STEP 3: If convergence criterion is satisfied, STOP; otherwise goto STEP 1.

Figure 4 Metastrategy Steps for Multistage Dynamic Lot-Sizing

A key insight is to approach the original problem iteratively, solving subproblem q_i at each iteration using the modified Wagner-Whitin algorithm, $WW(\tilde{R}^2_{q_i}, c^s_{q_i}, c^h_{q_i}, \tilde{c}^v_{q_i} \mid \omega_{q_i}) \mapsto x_{q_i}(\omega_{q_i})$, independently of all other stages. Note that we have replaced the original variable cost $c^v_{q_i}$ with the modified variable cost $\tilde{c}^v_{q_i}$. This allows subproblems at different stages to remain *loosely coupled* when compared to a simultaneous formulation across all stages and times, such as a raw integer program. The advantage of such a loose coupling is that the information set $\mathscr{I}_{q_i}$ of each subproblem q_i is more tightly constrained, making x_{q_i} easier to deduce.

As with any application of the metastrategy, some number of iterations are required to solve a dynamic lot-sizing problem effectively. At each iteration k we define $\mathscr{Q} = \{q_1, q_2, \ldots, q_N\}$ and $\mathscr{V} = \{q_N, \ldots, q_2, q_1\}$. In other words, subproblems are solved by traversing forward-reachable sets in reverse (proper) order, while value function approximations are updated by traversing backward-reachable sets in proper order. The steps for implementing the metastrategy in this fashion are summarized in Figure 4. Graves (1981) shows that for deterministic problems, the procedure converges monotonically, but not necessarily to an optimal solution. The problem is interesting from the perspective of our metastrategy because the use of a linear approximation for a nonconvex problem is not immediately obvious. Later, we present experimental results that confirm Graves' earlier work, that the method does produce better solutions than a pure MRP solution.

5. Practical Value Function Approximations

The quality of the solution produced by the metastrategy is, of course, tied to the accuracy of the value function approximations. We can reasonably divide value functions into three broad classes: zero (that is, ignoring the impact of decisions on other subproblems), linear, and nonlinear. Within the class of nonlinear approximations, important subclasses include

separable concave (or convex for minimization problems), nonseparable concave, and nonconcave. Within the class of nonconcave functions would be discrete problems (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998). The simplest case, $\bar{V} = 0$, may seem trivial and uninteresting, but it is often the case that this is what is being used in practice, and hence comparisons against this null choice are meaningful.

Of all the nonzero choices, a linear approximation is always the simplest. It is the easiest to estimate and never destroys local problem structure. For example, if LSP_q is a network problem, then GSP_q will be a network problem if $\bar{V}_{q'}, q' \in \vec{\mathcal{M}}_q$ is linear.

The cardinality of the backward-reachable set, $\overleftarrow{\mathcal{M}}_q$, plays an important role in the choice of value function approximation. If $|\overleftarrow{\mathcal{M}}_q| > 1$, then linear approximations provide a natural decomposition. Also, we would be able to replace the doubly indexed function $V_{qq'}(R_{qq'})$ in (10) with a singly indexed one, which further simplifies the method. If a nonlinear approximation of subproblem q is used, then it complicates the problem of solving the subproblems in the set $\overleftarrow{\mathcal{M}}_q$. Given the importance of this issue, we divide our discussion between problems where $|\overleftarrow{\mathcal{M}}_q| \leq 1$ and $|\overleftarrow{\mathcal{M}}_q| > 1$.

5.1. Subproblems q with $|\overleftarrow{\mathcal{M}}_q| \leq 1$

Subproblems with backward-reachable sets of low cardinality are common in coarse-grained decompositions $\mathcal{A}_q, q \in \mathcal{Q}$. Consider, for example, a temporal decomposition of a transportation problem, $\mathcal{Q} = \{q_0, q_1, \ldots, q_T\}$, with unit travel times and a finite time horizon $\mathcal{T} = \{0, 1, \ldots, T\}$. In this case, each subproblem q_t corresponds to a single time period t and the set $\overleftarrow{\mathcal{M}}_{q_t}$ is the singleton $\{q_{t-1}\}$. Subproblems for times $s < t - 1$ are not present in $\overleftarrow{\mathcal{M}}_{q_t}$ because of the single-period travel times. Another example of such subproblems occurs in serial assembly systems such as in multistage dynamic lot-sizing problems. In these cases, each item must pass through stages $\{N, \ldots, 1\}$ in strict sequence, so that the decomposition $\mathcal{Q} = \{q_N, \ldots, q_1\}$ assigns a subproblem q_i to each stage i. In this case, the backward-reachable set of subproblem q_i is the singleton $\{q_{i+1}\}$.

The case when $|\overleftarrow{\mathcal{M}}_q| = 0$ is not interesting. It may occur in real problems, for instance for q_0 in a temporal decomposition or q_N in a stage decomposition, but because $\overleftarrow{\mathcal{M}}_q = \phi$ we may set $\bar{V}_q(R_q) = 0$ for all values of R_q without penalty.

The case when $|\overleftarrow{\mathcal{M}}_q| = 1$ is of particular interest, because we can replace (10) with

$$\tilde{V}_q(R_q) = E\left\{\max_{x_q} C_q(x_q, \omega_q) + \bar{V}_{q'}(R_{q'} \mid \omega_q)\right\}, \qquad (22)$$

where $q' = \vec{\mathcal{M}}_q$, conditioned on ω_q. The variable $R_{q'}$ is determined completely by x_q, so we do not need the doubly indexed formulation of (10), considerably simplifying the solution and updating process, even if $\bar{V}_q(R_q)$ is nonlinear.

5.2. Subproblems q with $|\overleftarrow{\mathcal{M}}_q| > 1$

Subproblems with backward-reachable sets of high cardinality are common in fine-grained decompositions of the attribute space $\mathcal{A}$. A typical example is a temporo-spatial decomposition $\mathcal{Q} = \{\mathcal{W} \times \mathcal{T}\}$ of a transportation problem with terminals $\mathcal{W}$ and time horizon $\mathcal{T}$. If travel times are deterministic and stationary, the backward-reachable set $\overleftarrow{\mathcal{M}}_q$ contains an element for each terminal $\mathcal{W}$ that permits travel to the terminal of subproblem q. In the case of random or nonstationary travel times, the same terminal may appear in $\overleftarrow{\mathcal{M}}_q$ at different points in time.

When $|\overleftarrow{\mathcal{M}}_q| > 1$ we can use the natural decomposition of linear functions to produce a subproblem of the form

$$\tilde{V}_q(R_q) = E\left\{\max_{x_q} C_q(x_q, \omega_q) + \sum_{q' \in \vec{\mathcal{M}}_q} \bar{V}_{q'} x_{qq'}\right\}, \qquad (23)$$

where $\bar{V}_{q'}$ and $x_{qq'}$ are vectors defined over the elements $a' \in \mathcal{A}_{q'}$. Here, the value function approximation is singly indexed, but we only need to consider the flow from q to q', contained in the vector $x_{qq'}$.

Linear value function approximations can provide very good results but are often unstable unless they are coupled with a strategy to stabilize flows from one iteration to the next. The alternative to linear approximations are nonlinear functions. Assume for simplicity that we are able to work with separable nonlinear functions so that they are reasonably tractable and do not significantly complicate the solution of GSP_q. One algorithm with which we have worked is the CAVE algorithm, which iteratively estimates a concave, separable approximation of the value function (Godfrey and Powell 2001), adaptable to both continuous and piecewise linear functions. Our discussion below, however, would apply to any nonlinear-approximation method.

When we use a nonlinear value function approximation, we have to face the problem that all the subproblems in $\overleftarrow{\mathcal{M}}_q$ are sending flow into subproblem q. In this case, we can consider two strategies.

1. Finer-grained primal information content: Use the more fine-grained primal information content $x_{qq'}$ when approximating the impact on q' *in addition to* the aggregate vector $R_{q'}$.

2. Finer-grained dual information content: Store the dual information content for subproblem q' in a finer partition $\bar{V}_{qq'}$ *instead of* $\bar{V}_{q'}$.

We discuss each of these modifications.

Finer-Grained Primal Information Content. We try an obvious modification of CAVE, which we call CAVE-P (short for CAVE-PRIMAL), because it differs from CAVE in how primal information content is indexed. CAVE-P records the state of subproblem q at each iteration k for future use in two forms: R_q^k (which gives the total flow into q from all subproblems) and $x_{q'q}^k$, $q' \in \overleftarrow{\mathcal{M}}_q$ which gives the flow from each subproblem. At iteration $k+1$, the evaluation of $\bar{V}_q$ in the optimality recursion (10) of subproblem q_i is performed on $R_q^k - x_{q'q}^k + x_{q'q}^{k+1}$. In effect, we use the primal state from iteration k to "forecast" the primal state at iteration $k+1$. This has the effect of communicating the decisions made in subproblems $q'' \in \overleftarrow{\mathcal{M}}_q$ at iteration k to subproblem q' in iteration $k+1$ with the hope of avoiding resource flooding. The procedure for updating $\bar{V}_q$ is the same as the $|\overleftarrow{\mathcal{M}}_q| \le 1$ case.

Finer-Grained Dual Information Content. We build a value function $\bar{V}_{q_i q}$ for each $q_i \in \overleftarrow{\mathcal{M}}_q$. We call this modification "CAVE-D" (short for CAVE-DUAL) because it differs from CAVE principally in how dual information content is indexed. Instead of using a function $\bar{V}_q(R_q)$, we use a function $\bar{V}_{q'q}(x_{q'q})$. We do not need the flows $x_{q''q}$ from any other subproblem $q'' \in \overleftarrow{\mathcal{M}}_q$. The computational implications of this modification are that we must now build $|\overleftarrow{\mathcal{M}}_q|$ value function approximations for subproblem q. The procedure for updating $\bar{V}_{q'q}$ is more complicated than updating $\bar{V}_q$ in the CAVE-P or $|\overleftarrow{\mathcal{M}}_q| \le 1$ case. The intuitive approach is to use the subproblem q' duals $p_{q'q}^+$ and $p_{q'q}^-$ to update $\bar{V}_{q'q}$. Unfortunately, this approach cannot work because these dual values only indirectly measure the slope values of $\bar{V}_{q'q}$. To capture the true future impact, we use the following adjusted duals to update $\bar{V}_{q'q}$:

$$\tilde{p}_{q'q}^+ = \max_{q'' \in (\overleftarrow{\mathcal{M}}_q \cap q) \setminus q'} \{p_{q''q}^+\}, \tag{24}$$

$$\tilde{p}_{q'q}^- = \min_{q'' \in (\overleftarrow{\mathcal{M}}_q \cap q) \setminus q'} \{p_{q''q}^-\}, \tag{25}$$

where $p_{qq}^{+/-}$ are the duals of the constraint (7) in subproblem q. Notice the difference between $p_{q'q}^{+/-}$ and $\tilde{p}_{q'q}^{+/-}$. The individual duals $p_{q'q}^{+/-}$ estimate the local impact of a decision made in subproblem q' while the adjusted duals $\tilde{p}_{q'q}^{+/-}$ measure the impact of a decision made in subproblem q'' on all subproblems $q'' \in (\overleftarrow{\mathcal{M}}_q \cap q) \setminus q'$. Taking a maximum to compute $\tilde{p}_{q'q}^+$ ensures that it measures the maximal impact of adding a resource to q from q'. Similarly, the minimization to determine $\tilde{p}_{q'q}^-$ ensures that it measures the minimal impact of removing a resource from q. If $\tilde{p}_{q'q}^+ > \tilde{p}_{q'q}^-$, we preserve concavity by setting

$$(\tilde{p}_{q'q}^+, \tilde{p}_{q'q}^-) \leftarrow \begin{cases} (\tilde{p}_{q'q}^-, \tilde{p}_{q'q}^+) & \text{if } x_{q'q} > 0, \\ (\tilde{p}_{q'q}^+, \tilde{p}_{q'q}^+) & \text{otherwise.} \end{cases} \tag{26}$$

This also avoids using the ill-defined value of $\tilde{p}_{q'q}^-$ when $R_{q'q} = 0$ (the function $\bar{V}_{q'q}$ is only defined on $\Re_+$). It is not immediately clear whether we should prefer CAVE-D over CAVE-P for a given problem.

6. Computational Experiments

Our metastrategy offers an approach that is scalable to very large problems (such as a railroad or trucking company), as well to as stochastic problems. We reduce problems that are computationally intractable purely because of their scale to smaller problems that can, in principle, be solved using known algorithms. Our claim is that through the use of value functions, we can obtain solutions that approach those obtained by global-optimization formulations. Not surprisingly, we cannot test this hypothesis on either ultra-large-scale problems, or on stochastic problems. Instead, we test our method on deterministic problems for which we can find an optimal solution. We focus on a fleet management problem (the driver-scheduling problem considered earlier is much too large to solve to optimality), and the dynamic lot-sizing problem, which exhibits very different properties. Using the fleet management problem, we also investigate the behavior of the algorithm to different types of decomposition (temporal, spatial/temporal) as well as both synchronous and asynchronous computation.

We aim to answer three key research questions about the metastrategy

1. Sensitivity to decomposition: How sensitive is the metastrategy to the decomposition used? What modifications, if any, are necessary to $\bar{V}$ and X^π to ensure consistent solution quality across decompositions. These questions are addressed in Section 6.1.

2. Sensitivity to asynchronous computation: Is the metastrategy suitable for asynchronous, distributed computation? We investigate in Section 6.2 how solution quality and computational effort are affected by the scheme used to construct and order $\mathcal{Q}^k$ and $\mathcal{V}^k$. Most importantly, how is solution quality affected when subproblems are solved out of order?

3. Sensitivity to problem class: How sensitive is the metastrategy to the problem class on which it is used? What modifications, if any, are necessary to Q, $\bar{V}$, and X^π to ensure consistent solution quality across problem classes? This question is addressed in Section 6.3. It is important to emphasize that our experiments are not intended to show that our algorithm is better than specialized algorithms that have already been developed for these problems. They are designed only to provide a measure of solution quality by comparing the results against an optimal solution.

All computational experiments were performed on a 300 MHz Pentium II processor with 256 MB of RAM. Code for the experiments was written in Java 1.1.5

and compiled using Symantec's JIT-compiler version 3.00.029. The object-oriented framework for DRTPs described in Shapiro (1999) was used for rapid prototyping of the code.

6.1. Sensitivity to Decomposition: $\mathcal{Q}$

We claimed in the introduction that the metastrategy could be used to effectively solve a DRTP under a variety of decompositions $\mathcal{Q}$. The first step to validating this claim is to study the performance of the metastrategy under various decompositions of a fleet management problem (see Figure 5). For example, transportation companies are often decomposed spatially (5a), but the evolution of information typically imposes a temporal decomposition (5b), which can be even more pronounced if different people handle decisions at different times during a week (companies often require two or four separate shifts to cover operations over an entire week). Figure (5c) depicts a hybrid of these two. Finally, the decomposition may be by the type of resource (5d), as might happen in a railroad where different groups handle different types of freight cars, while another group handles locomotives for the entire company.

The sensitivity of metastrategy performance to the type of decomposition used is an important issue. If it were to perform well only on certain decompositions, the metastrategy would not be broadly applicable.

For instance, in fleet management problems it is common to use a temporal decomposition for computational reasons. In this case $\mathcal{Q} = \{q_t, t \in \mathcal{T}\}$ where q_t is the subproblem for time t *across all space*. Not only does this handily support a rolling-horizon implementation, but it also makes each subproblem simple. Solving one time period at a time eliminates coupling constraints on load coverage allowing us to solve $|\mathcal{T}|$ network-type problems of moderate size. This invariably requires less total computational effort than does solving one giant IP across time. In other cases, the nature of the decomposition to be used is not determined by computational considerations. For instance, in Section 4.1 we described a temporo-spatial decomposition $\mathcal{Q} = \{\mathcal{W} \times \mathcal{T}\}$ that creates a single subproblem for each geographical location at each time. This would arise when a company plans operations locally, or if a central planner works on one location at a time. Although a total of $|\mathcal{W}||\mathcal{T}|$ subproblems result, each is smaller and easier to solve than in the pure temporal decomposition.

To test the metastrategy thoroughly, we ran experiments on a total of nine fleet management test problems of varying size and difficulty, as indicated in Table 1. The loads in these problems are negatively correlated, which means that locations with many loads terminating at them have few loads originating at them. Generally, this leads to either a net deficit or surplus of vehicles at a given location. As shown in Cheung and Powell (1996) and Godfrey and Powell (2002a), negatively correlated demands require more repositioning moves to obtain a high-quality solution than do demands that are independent or positively correlated, so that these problems are actually harder to solve than their size would indicate.

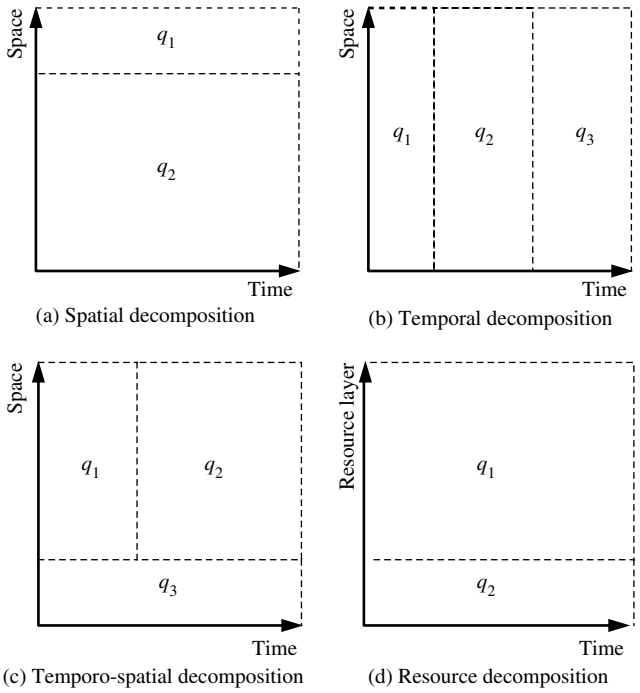

Figure 5 Potential Decompositions $\mathcal{Q}$ for Fleet Management

Temporal Decomposition. This is the simplest case, with $|\vec{\mathcal{M}}_q| \leq 1$ for all $q \in \mathcal{Q}$. We investigate the two forms of $\bar{V}_q$ suggested in Section 5.1: LINEAR and CAVE (piecewise linear and separable). We are interested in the best "OPT ratio" achieved over 100 iterations of the metastrategy. The OPT ratio is computed as the ratio, in percent, of a metastrategy objective-function value to an LP-optimal upper bound generated using CPLEX. Table 2 summarizes the results

Table 1 Fleet Management Test Problems (20 Locations, Six-Period Load Pickup Time Windows, \$0.50/Loaded Mile Revenue; LP Upper Bounds from Godfrey and Powell 2002b)

Problem	Repositioning cost/mile	Vehicles	Loads	$\lvert\mathcal{T}\rvert$	LP Upper bound
P1	\$0.40	100	1992	30	\$604,421
P2	\$0.70	100	1992	30	\$534,117
P3	\$1.00	100	1992	30	\$495,740
P4	\$0.70	200	1019	15	\$282,610
P5	\$0.70	200	1992	30	\$543,703
P6	\$0.70	200	4024	60	\$1,080,698
P7	\$0.40	400	1992	30	\$627,643
P8	\$0.70	400	1992	30	\$562,947
P9	\$1.00	400	1992	30	\$528,783

Table 2 Best OPT Ratio for the Temporal Decomposition

Problem	LINEAR	CAVE
P1	98.5	98.6
P2	98.6	98.8
P3	98.8	98.9
P4	97.6	99.1
P5	96.9	99.3
P6	98.0	99.5
P7	94.6	99.4
P8	95.3	99.6
P9	93.9	99.5

Table 3 Best OPT Ratio for the Temporo-Spatial Decomposition

Problem	LINEAR	CAVE	CAVE-P	CAVE-D
P1	97.7	95.7	95.4	98.4
P2	97.7	95.9	95.3	98.3
P3	97.8	94.0	94.4	98.3
P4	96.9	88.0	87.8	97.9
P5	97.1	89.7	89.2	98.7
P6	97.7	93.2	92.0	98.8
P7	94.7	66.3	66.3	99.2
P8	93.0	78.5	79.0	99.0
P9	96.5	82.5	82.5	98.7

for the temporal decomposition. The CAVE form of $\bar{V}_q$ achieves a higher ultimate OPT ratio than does the LINEAR form on the temporal decomposition. Evidently the ability to model more than one slope $(\partial \bar{V}_q / \partial R_q)$ across the domain of $\bar{V}_q$ allows the CAVE value function approximation to reflect more accurately the benefit of various decisions. Moreover, the performance of the CAVE approximation is much more stable than that of the LINEAR approximation as indicated in Figure 6. The LINEAR approximation estimates the same marginal benefit for an extra vehicle at a given location and time regardless of how many vehicles may already be there. In effect, this approximation lacks restraint with respect to repositioning moves, which leads to the wild fluctuations in solution quality.

Temporo-Spatial Decomposition. This is the more difficult case, with $|\vec{\mathcal{M}}_q| > 1$ for all $q \in \mathcal{Q}$ (except at time 0 when $|\vec{\mathcal{M}}_q| = 0$). We investigate four forms of $\bar{V}_q$: LINEAR and CAVE from Section 5.1, and CAVE-P and CAVE-D from Section 5.2. The best OPT ratio achieved over 100 iterations is given in Table 3. Both CAVE and LINEAR do worse on the temporo-spatial decomposition than they did on the temporal decomposition. CAVE's drastic decline in performance is caused by over-saturation of the high-slope initial break segments (i.e., resource flooding). The CAVE-P approximation was designed to prevent resource flooding. Unfortunately, as Table 3 indicates, the performance of this approximation is no better than that of CAVE. A further modification of CAVE-P considered terminals in random space order at a given time. Although this modification was intended to allow greater exploration of the state space on R_q, it produced an OPT ratio not significantly better than CAVE-P. Consequently, CAVE-P is dropped from further discussion. The best approximation for the temporo-spatial approximation turns out to be CAVE-D, which uses more fine-grained dual information $\bar{V}_{q_i q}$. In fact, the temporo-spatial performance of CAVE-D is comparable to the performance of CAVE under the simpler temporal decomposition. Figure 7 plots the OPT ratio, by iteration, of each approximation. Notice how CAVE-D achieves asymptotic near-optimality and remarkable stability for the temporo-spatial

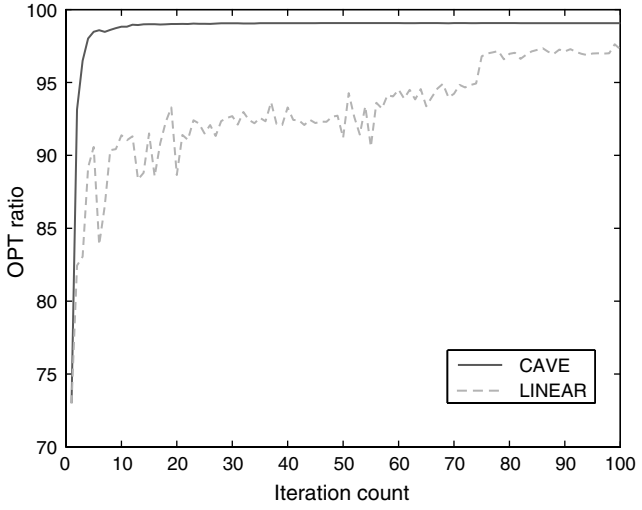

Figure 6 OPT Ratio of CAVE vs. LINEAR: Temporal Decomposition of P4

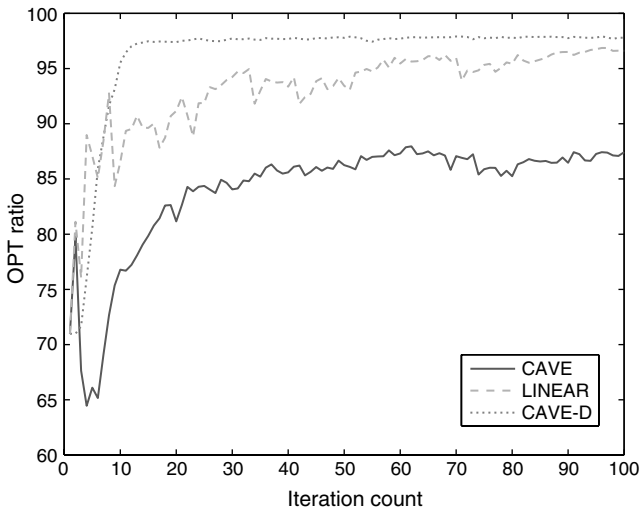

Figure 7 OPT Ratio for the Temporo-Spatial Decomposition of P4

decomposition that only CAVE had previously displayed under the temporal decomposition.

6.2. Sensitivity to Asynchronous Computation

This section concentrates on the construction and ordering of $\mathcal{Q}$ and $\mathcal{V}$. The set $\mathcal{Q}$ describes the order in which subproblems should be solved, while $\mathcal{V}$ describes the order in which value functions should be updated. Our aim is to determine the sensitivity of the metastrategy to asynchronous computation, as indicated by experiments on fleet management problems. To begin, we must define the following terms on the decomposition $\mathcal{Q}$.

Definition 6.1. A decomposition $\mathcal{Q}$ has a *full order* if it can be uniquely ordered such that any adjacent elements q_i and q_j, $i \neq j$, satisfy:

1. Either q_i always strictly precedes q_j, written $q_i \prec q_j$, or
2. q_i always strictly succeeds q_j, written $q_i \succ q_j$.

We interpret "precedes" to mean "should be solved first" when applied to $\mathcal{Q}$ and "should be updated first" when applied to $\mathcal{V}$.

Definition 6.2. A decomposition $\mathcal{Q}$ has a *spartial order* if it can be ordered such that any adjacent elements q_i and q_j, $i \neq j$, satisfy:

1. Either q_i can precede q_j, written $q_i \preceq q_j$, with no q_k for which $q_i \prec q_k \preceq q_j$, or
2. q_i can succeed q_j, written $q_i \succeq q_j$, with no q_k for which $q_j \prec q_k \preceq q_i$.

Writing $q_i \prec q_j$ implies that q_i must come before q_j, while writing $q_i \preceq q_j$ implies that q_i may come before or after q_j. Hence, every full order defines one or more partial orders, but the converse does not hold. A temporal decomposition, $\mathcal{Q} = \{q_t, t \in \mathcal{T}\}$, always possesses a full order by time, where $q_t \prec q_s$ when $s > t$. A temporo-spatial decomposition, $\mathcal{Q} = \{\mathcal{W} \times \mathcal{T}\}$, possesses only a partial order, because subproblems cannot generally be ordered across space $\mathcal{W}$.

In a completely synchronous environment it makes sense to use a single ordering of $\mathcal{Q}$ and $\mathcal{V}$. In an asynchronous environment, the ordering may change from one iteration to the next. For this setting, we let $\mathcal{Q}^k$ and $\mathcal{V}^k$ be the ordering of each set at iteration k. Thus, at iteration k, we would solve the subproblems in the order in which they appear in $\mathcal{Q}^k$, and update the value functions in the order in which they appear in $\mathcal{V}^k$. To study the impact of an asynchronous environment, we choose these sets randomly from one iteration to the next. By studying the performance of the metastrategy under different schemes for constructing and ordering $\mathcal{Q}^k$ and $\mathcal{V}^k$, we hope to establish its viability as a distributed-optimization algorithm. In a real distributed computing environment, we can generally only observe $\mathcal{Q}^k$ and $\mathcal{V}^k$—their construction and ordering will be fixed by the communications protocol and computer network topology in effect. In a research paper, we have the luxury of directly constructing and ordering $\mathcal{Q}^k$ and $\mathcal{V}^k$ to simulate different distributed computing environments artificially. We assume that all decompositions possess at least a partial order. This requirement will become particularly important when computing the objective function because it is not clear how to compute such values for iterations performed out of order.

Recall that the construction phase involves selecting a subset of subproblems from $\mathcal{Q}$ to be solved at iteration k, while the ordering phase determines the order with which that subset should be solved. An obvious analogy can be made with the elements of a queueing system. We can imagine $\mathcal{Q}^k$ as a queue of subproblems (customers) waiting to be solved (served) by the decision function X^π (the server). Adding elements to $\mathcal{Q}^k$ during the construction phase constitutes an arrival phase of the queueing system. The order in which subproblems are solved during the ordering phase constitutes the queue discipline of the queueing system. In optimization over the Internet this analogy becomes entirely concrete. Our metastrategy naturally extends to multiagent decision systems, and we can extend the queueing analogy to incorporate multiple decision functions in parallel or tandem.

In our experiments, we assume, without loss of generality, a single construction and ordering phase at each iteration. We investigate only one scheme for governing the construction phase of an iteration, which we call RANDOM, whereby $\mathcal{Q}^k = \mathcal{Q}$ but in random order (e.g., $\{q_1, q_{17}, q_3, \ldots\}$ in a temporal decomposition). This simulates an asynchronous distributed implementation where each subproblem is solved on a separate host with possible communication link failures or drastic differences in speed between hosts. The single scheme for governing the ordering phase of an iteration is FIFO. This is classic first-in-first-out, whereby subproblems in $\mathcal{Q}^k$ are solved in order of insertion.

We test a single class of construction phases for $\mathcal{V}^k$ that we call PRIMAL-LINKED, whereby $q \in \mathcal{Q}^k \Rightarrow \bar{V}_q \in \mathcal{V}^k$. The motivation for this class is that solving subproblem q will generally reveal new information about $\partial \bar{V}_q^k / \partial R_q^k$ that we can use to update $\bar{V}_q^k$. We order $\mathcal{V}^k$ using the FIFO scheme, whereby elements of $\mathcal{V}^k$ are updated in order of insertion.

An important question remains: how do we measure the objective function value at iteration k? When $\mathcal{Q}^k$ maintains at least a partial order defined on $\mathcal{Q}$ (e.g., $\mathcal{Q}^k = \{q_1, q_2, \ldots, q_T\}$ in a temporal decomposition), this is straightforward—use $\bar{V}_0(R_0)$ from (10). However, when subproblems are solved out of order, their primal states may not always be jointly feasible, so that adding their objective functions is like adding apples

and oranges. We propose two strategies for measuring the objective function.

Audit Iterations: With period $\eta \in \mathbb{Z}_{++}$ we perform an iteration k with $\mathcal{Q}^k = \mathcal{Q}$, in order. The purpose of such an iteration is merely to estimate the quality of the current solution produced by the metastrategy.

- Random: We prohibit dual updates so as not to bias future iterations in case the audit iteration was to be done offline in an asynchronous environment.
- Spacer: We allow dual updates.

The point of the spacer iterations is to perform periodic "correct" iterations that both solve the subproblems and perform the dual updates based on a globally-feasible primal state. We use $\eta = 5$ throughout.

Under an asynchronous temporal decomposition, CAVE does not do terribly well when dual updates are completely random, as indicated in Table 4. However, the asynchronous spacer version of CAVE does better than even the best synchronous LINEAR implementation (except on P1 and P3). Moreover, the performance of CAVE remains stable across both synchronous and asynchronous implementations, while LINEAR appears decidedly unstable in the asynchronous implementation. Figure 8 shows the performance of the linear approximation under an asynchronous updating strategy, while Figure 9 shows the same plot for CAVE. These results shows that the LINEAR strategy is much more dependent on spacer iterations for its success than is its CAVE counterpart. Because spacer steps require synchronization (and the concomitant synchronization penalties, Bertsekas and Tsitsiklis 1989) this dependence makes LINEAR less suitable for asynchronous temporal decompositions than CAVE. Note that our asynchronous iterations have all been of the Gauss-Seidel variety. It is not clear what the communications overhead would be for the LINEAR and CAVE approximations. Moreover, the density of the dependency graphs (essentially $\overleftarrow{\mathcal{M}}_q$ in dual space and $\vec{\mathcal{M}}_q$ in primal space) may have a significant effect on the speedup obtained.

Table 4 Best OPT Ratio: Asynchronous Temporal Decomposition

	Synchronous		Asynchronous random		Asynchronous spacer	
Problem	LINEAR	CAVE	LINEAR	CAVE	LINEAR	CAVE
P1	98.5	98.6	76.9	89.2	81.9	98.4
P2	98.6	98.8	83.7	93.1	84.6	98.6
P3	98.8	98.9	81.0	94.5	86.4	98.7
P4	97.6	99.1	78.9	96.4	84.1	98.6
P5	96.9	99.3	78.9	93.2	82.5	98.9
P6	98.0	99.5	78.7	91.3	81.7	99.1
P7	94.6	99.4	54.7	90.2	71.1	99.2
P8	95.3	99.6	65.4	95.1	69.0	99.1
P9	93.9	99.5	63.6	95.1	70.9	99.0

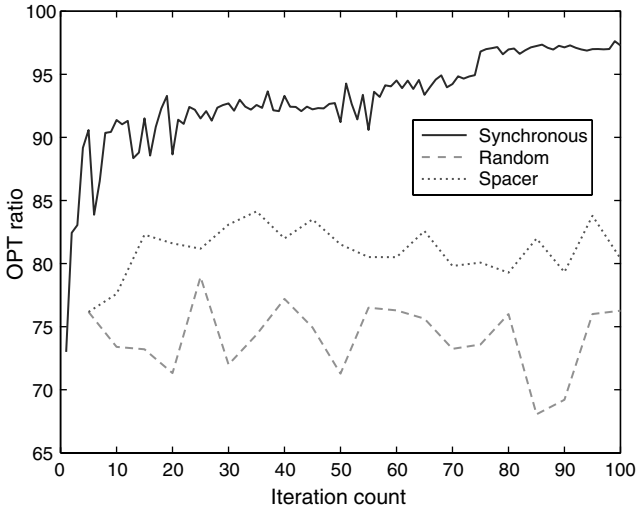

Figure 8 OPT Ratio of LINEAR: Asynchronous Temporal Decomposition of P4

6.3. Extension to Multistage Dynamic Lot-Sizing Problems

Sections 6.1 and 6.2 studied the issues of metastrategy performance under various decompositions and asynchronous computation schemes on a single problem class. The goal of this section is not to readdress those same research questions, but to demonstrate that the metastrategy can be applied to a completely different problem class in a straightforward manner and yet still deliver high-quality solutions. To this end, we choose the context of the dynamic lot sizing problems described in Section 4.2.

First, note that a number of different decompositions are possible for this problem class. Three logical

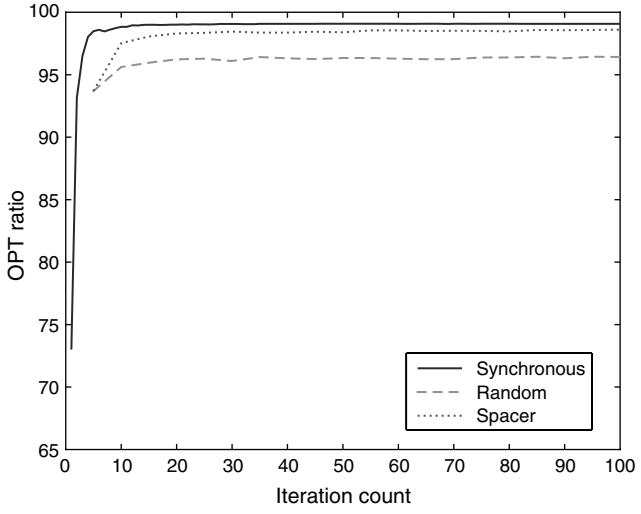

Figure 9 OPT Ratio of CAVE: Asynchronous Temporal Decomposition of P4

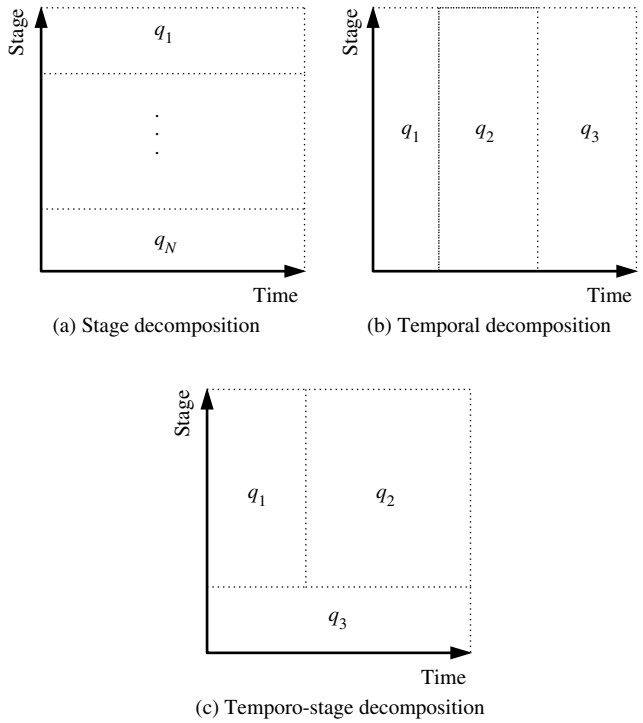

Figure 10 Potential Decompositions $\mathcal{Q}$ for Multistage Dynamic Lot Sizing

alternatives are presented in Figure 10. It is reasonable to assume that there is a decision maker at each stage who is forecasting future demands, and then creating a production plan over a planning horizon. Under this reasoning, it makes sense to choose a decomposition by stage. This defines $\mathcal{Q} = \{q_N, \ldots, q_i, \ldots, q_1\}$ where subproblem q_i corresponds to stage i *across all time periods.* Recall that we use the LINEAR value function approximation $\bar{V}_{q_i}$ to measure the impact of orders placed by subproblem q_{i-1} (stage $i-1$) with subproblem q_i. In contrast to the experiments in the previous sections, which measured the value of layer-one resource flows between subproblems, we now use $\bar{V}_{q_i}$ to measure the value of level-two resource flows (orders) between subproblems.

To test the metastrategy, we randomly generated a series of eight test problems. For each stage i in a test problem, the setup cost c_i^s was chosen from the equally likely outcomes 150, 300, 600, and 1500. Holding costs were assumed to increase as the items progress from raw material to finished product. Following this logic, we set the holding cost for the first stage (stage N) to $c_N^h = 1$; the holding cost for each subsequent stage $i \in [1, N)$ was generated according to the progression

$$c_i^h = e + \sum_{j>i} c_j^h, \tag{27}$$

where e was chosen from the equally likely outcomes 0.1, 0.5, 1.0, and 2.0. The assumption that holding costs are strictly increasing is realistic for most serial lot-sizing systems because holding costs are cost-of-capital-based, and a positive value is added to the product at each stage. Variable production costs c_{ti}^v were set to zero for all i and t. Initial first-stage demands z_{t1} were assumed to be identically distributed, for a given time t, across all problem instances. Specifically, the mean $E[z_{t1}]$ was chosen from the equally likely outcomes 0, 10, 20, 30, 40, 100, 200, and 400. For a particular problem realization $\omega = (\ldots, \omega_{t1}, \ldots)$, we set

$$z_{t1}(\omega_{t1}) = E[z_{t1}] + \varrho E[z_{t1}]\zeta(\omega_{t1}), \tag{28}$$

where $\zeta \sim N(0, 1)$ and $\varrho \geq 0$ was the multiplicative "error" term. Exogenous demands z_{ti} for $i > 1$ were set to zero for all t.

The dimensions of the test problems are given in Table 5, with each problem corresponding to a different realization ω. The "LP LB" (linear programming lower bound) figure is the optimal objective function obtained using CPLEX for the linear relaxation of each problem. We should emphasize that CPLEX is not a specialized algorithm for this class, so comparisons in terms of CPU times or other computational performance measures are not meaningful. But we would argue that specialized algorithms are generally not scalable to handle larger problems with multiple product types and uncertainties. By contrast, because we are solving sequences of relatively smaller subproblems, we can handle these more difficult problems, even using standard commercial solvers. The "IP OPT" column in Table 5 indicates the optimal objective function of the integer formulation.

Table 5 Results for Deterministic Multistage Dynamic Lot-Sizing Problems (Error Term Fixed at $\varrho =$ 0.05 for All Problems)

Problem	N	$\lvert\mathcal{T}\rvert$	LP LB	IP OPT	IP Gap	metastrategy	OPT ratio
P1	5	12	\$4,806.0	\$12,142.8	152.7	\$12,460.2	1.026
P2	5	12	\$4,791.5	\$12,025.2	151.0	\$12,426.6	1.033
P3	5	18	\$5,915.0	\$15,064.2	154.7	\$15,634.7	1.038
P4	5	18	\$5,805.3	\$14,956.6	157.6	\$15,527.4	1.038
P5	10	12	\$10,115.2	\$25,435.2	151.5	\$26,695.2	1.050
P6	10	12	\$10,130.2	\$25,359.6	150.3	\$26,611.2	1.049
P7	10	18	\$14,444.7	\$33,785.0	133.9	\$37,084.0	1.098
P8	10	27	\$10,978.6	—	—	\$36,391.8	—

Since this is a minimization problem, we redefine the "OPT ratio" as the ratio, in percent, of the optimal objective function of the metastrategy to the IP OPT. We show the results for problems up to the limit that we could solve using CPLEX, and the results indicate that for all but one problem, we are within a few percent of the optimal solution.

7. Conclusion

In this paper we present a metastrategy for solving DRTPs. We show through a series of examples how the metastrategy can be implemented to solve a broad variety of important practical problem instances, including fleet management, driver scheduling, and multistage dynamic lot-sizing. The method requires that the user fill in four "hot spots" to specialize the strategy to a specific application. We do not mean to minimize the challenge of modeling complex problems with our strategy but suggest that this list of hot spots helps organize the steps that are required.

The metastrategy introduces into the basic model the representation of the organization of decisions and information. We then solve subproblems that reflect actual problems that might be solved by humans. These subproblems can be solved using existing algorithms. The focus of the paper is on strategies for coordinating the subproblems. In the case of multistage lot-sizing problems, we show that linear approximations can produce very good results. In the case of resource allocation problems, we demonstrate the advantage of using nonlinear approximations, but also the technical challenges that this strategy introduces.

We believe that the explicit modeling of the organization of decisions and information represents a useful strategy when approaching very large-scale problems such as a supply chain or freight transportation company. It allows us to exploit the vast array of algorithms developed in the operations research community, while retaining the additional realism of modeling the actual decision-making structure within an organization (or market).

Acknowledgments

This research was supported in part by Grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research. The authors gratefully acknowledge the helpful comments of the editors and reviewers.

References

Bertsekas, D. P., J. N. Tsitsiklis. 1989. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ.

Bertsekas, D., J. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Cheung, R., W. B. Powell. 1996. An algorithm for multistage dynamic networks with random arc capacities, with an application to dynamic fleet management. *Oper. Res.* **44** 951–963.

Dejax, P., T. Crainic. 1987. A review of empty flows and fleet management models in freight transportation. *Transportation Sci.* **21** 227–247.

Eppen, G. D., F. J. Gould, B. P. Pashigian. 1969. Extensions of the planning horizon theorem in the dynamic lot size model. *Management Sci.* **15** 268–277.

Ermoliev, Y. 1988. Stochastic quasigradient methods. Y. Ermoliev, R. Wets, eds. *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, Berlin, Germany.

Frantzeskakis, L., W. B. Powell. 1990. A successive linear approximation procedure for stochastic dynamic vehicle allocation problems. *Transportation Sci.* **24** 40–57.

Godfrey, G., W. B. Powell. 2001. An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems. *Management Sci.* **47** 1101–1112.

Godfrey, G., W. B. Powell. 2002a. An adaptive, dynamic programming algorithm for stochastic resource allocation problems I: Single period travel times. *Transportation Sci.* **36** 21–39.

Godfrey, G., W. B. Powell. 2002b. An adaptive, dynamic programming algorithm for stochastic resource allocation problems II: Multi-period travel times. *Transportation Sci.* **36** 40–54.

Graves, S. C. 1981. Multistage lot-sizing: An iterative procedure. L. B. Schwarz, ed. *Multi-level Production/Inventory Control Systems: Theory and Practice. TIMS Studies in the Management Sciences*, Vol. 16. North-Holland, New York, 95–110.

Papadaki, K., W. B. Powell. 2002. A monotone adaptive dynamic programming algorithm for a stochastic batch service problem. *Eur. J. Oper. Res.* **142** 108–127.

Papadaki, K., W. B. Powell. 2003. An adaptive dynamic programming algorithm for a stochastic multiproduct batch dispatch problem. *Naval Res. Logist.* **50** 742–769.

Powell, W. B., T. A. Carvalho. 1998. Dynamic control of logistics queueing network for large-scale fleet management. *Transportation Sci.* **32** 90–109.

Powell, W. B., R. Cheung. 1994. Stochastic programs over trees with random arc capacities. *Networks* **24** 161–175.

Powell, W. B., J. A. Shapiro, H. P. Simão. 2001. A representational paradigm for dynamic resource transformation problems. R. F. C. Coullard, J. H. Owens, eds. *Annals of Operations Research*, Vol. 104 (1–4). Kluwer Academic Publishers, Dordrecht, The Netherlands, 231–279.

Puterman, M. L. 1994. *Markov Decision Processes*. Wiley, New York.

Shapiro, J. A. 1999. A framework for representing and solving dynamic resource transformation problems. Ph.D. thesis, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ.

Sutton, R., A. Barto. 1998. *Reinforcement Learning*. MIT Press, Cambridge, MA.

Wagner, H., T. Whitin. 1958. Dynamic version of the economic lot size model. *Management Sci.* **5** 89–96.

Zangwill, W. 1969. A backlogging model and a multi-echelon model of a dynamic economic lot size production system—a network approach. *Management Sci.* **15** 506–527.