A Distributed Decision Making Structure for Dynamic Resource Allocation Using Nonlinear Functional Approximations

Huseyin Topaloglu
School of Operations Research and Industrial Engineering
Cornell University, Ithaca, NY 14853
topaloglu@orie.cornell.edu

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

September 22, 2003

Subject classifications:
Dynamic Programming: Approximate Dynamic Programming
Computers: Artificial Intelligence
Technology Transportation: Dynamic Fleet Management

Area of Review:
Computing and Decision Technology

Abstract

This paper proposes a distributed solution approach to a certain class of dynamic resource allocation problems and develops a dynamic programming-based multi-agent decision making, learning and communication mechanism. In the class of dynamic resource allocation problems we consider, a set of reusable resources of different types have to be assigned to tasks that arrive randomly over time. The assignment of a resource to a task removes the task from the system, modifies the state of the resource and generates a contribution. We build a decision making scheme where the decisions regarding the resources in different sets of states are made by different agents. We explain how to coordinate the actions of different agents using nonlinear functional approximations, and show that in a distributed setting, nonlinear approximations produce sequences of min-cost network flow problems which naturally yield integer solutions. We also experimentally compare the performances of the centralized and distributed solution strategies.

Monolithic solution approaches to optimization problems arising in the context of complex systems are generally not feasible. Distributed solution techniques divide these problems into loosely-coupled subproblems and solve them independently by using an inter-subproblem communication mechanism that ensures that the independent solutions of the subproblems will yield a high quality solution. In this paper, we present a distributed solution approach to dynamic resource allocation problems. Our approach divides the problem into subproblems and relies on multiple agents to solve them. We develop a dynamic programming-based communication mechanism that helps each agent to estimate the impact of their solutions on the other agents. Central to our approach is the use of *nonlinear* functional approximations to handle the coordination.

Dynamic resource allocation problems considered in this paper involve the assignment of a set of *discrete* and *reusable* resources to tasks that occur randomly *over time*. Each resource in the system has a state and a type associated with it, which together determine the contribution of assigning a resource to a task. The arrival process of the tasks is known only through a probability distribution. The assignment of a resource to a task produces a reward, removes the task from the system and modifies the state of the resource. Such problems arise in many fields, such as dynamic crew scheduling (staff members are the resources, flights are the tasks, location and hours on duty of the staff member are the resource state), fleet management (vehicles are the resources, loads are the tasks, vehicle location is the resource state), machine assignment (machines are the resources, jobs are the tasks, machine wear is the resource state), emergency equipment/personnel management (ambulances/doctors are the resources, patients are the tasks, location/hours on duty is the resource state). Multiple resource states and the indivisibility of the resources differentiate our problem class from other resource allocation problems where a single divisible resource (for example computing capacity) has to be allocated among different tasks. The latter type of problems occur frequently in distributed computing systems (see Kurose & Simha (1989), Waldspurger, Hogg, Huberman, Kephart & Stornetta (1992)).

We were confronted with this problem in the context of managing fleets of vehicles to serve loads that occur over time at a set of locations. In real world operations, usually there is one decision agent for each location responsible for managing the vehicles at that location, and when making its decisions, the decision agent pays very little attention to what is happening in other locations. We devise an efficient way of assessing the impact of the decisions made at a certain location on the decisions made at the other locations and at future time periods. In this way, the decision makers in different locations can work collectively to achieve the maximum expected reward over all time periods, and we obtain a distributed solution method where the division of subproblems perfectly

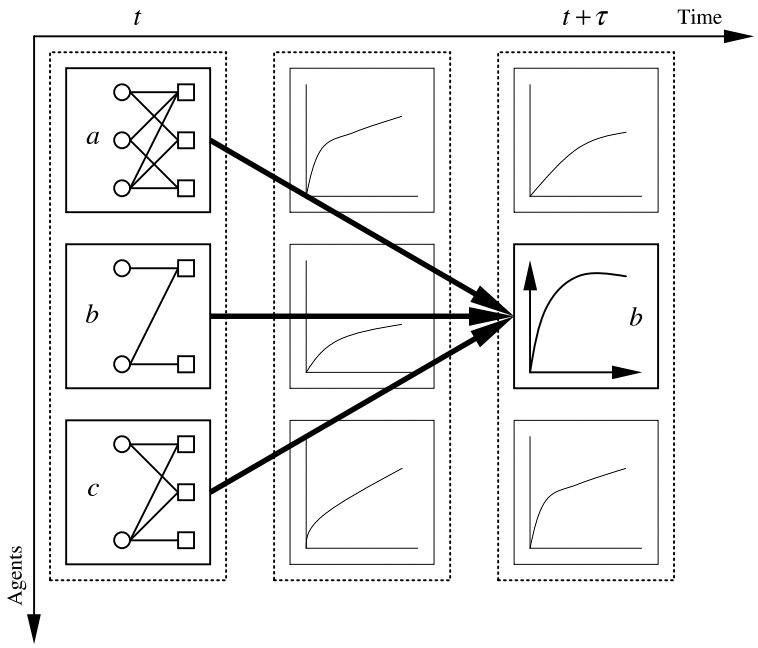

Figure 1: Interaction of different agents over time. In the figure, resources under the command of each agent is represented by circles and the tasks are represented by squares. Lines between the circles and the squares represent possible assignment decisions. Each agent may also have the option of doing nothing with a resource or sending it to another agent by incurring a cost.

matches the operational decision making structure.

In order to achieve this collaboration, the agents that solve different subproblems assess and learn the impact of their actions on the other parts of the system through nonlinear functional approximations. Roughly speaking, each agent makes its decisions in order to maximize the contribution of its actions and the impact of its actions on the rest of the system. However, a communication mechanism built on nonlinear functional approximations brings special challenges in coordinating the actions of different agents. Consider figure 1 where we illustrate the interaction of the agents over time. At the current time period t, agents a, b and c solve their subproblems to maximize the contribution of their own decisions and the impact of their decisions on the rest of the system. Assume that these agents can potentially impact agent b at time period $t+\tau$ by sending resources from the subproblems that they solve at time period t to the subproblem that agent b is going to solve at time period $t+\tau$, and this impact is approximated by a nonlinear function of the total number of resources in the future subproblem. Clearly, the impact of a particular agent on the future subproblem depends on the number of resources sent to the future subproblem by the other agents. Therefore, the agents cannot concentrate only on their own subproblems when assessing the impact of their decisions on the rest of the system. We note that if the approximations were

linear, such a problem would not occur.

In this paper, we introduce a communication mechanism built on nonlinear functional approximations that allows each agent to concentrate on its own subproblem. We facilitate the learning of the agents by updating the parameters of the functional approximations adaptively as each agent solves the subproblems under its control. Motivated by the fact that the expected marginal contribution (profit, revenue, reward, etc.) from an incremental resource decreases as the number of available resources increases, we use concave, piecewise-linear approximations in this paper. Hence, the parameters that characterize these approximations are their slopes over different parts of their domains.

In order to update the parameters of the functional approximations, we use the dual solution information from the subproblems, which roughly tells us the marginal worth of a resource. Thus, the primal solution is utilized to determine which actions to take and the dual solution is utilized to update the functional approximations and contrive an adaptive learning mechanism. An important question is how well the agents perform after having been trained for a certain period of time. To answer this question, we carry on the learning process (i.e. updating the approximations) for a certain amount of time. After this learning period, we fix the parameters of the approximations and test how well the system performs with the fixed set of approximations.

Our work falls into the area of *distributed artificial intelligence*, which has traditionally been divided into two subdisciplines (see Bond & Gasser (1988), Gasser & Huhns (1989), O'Hare & Jennings (1996), Moulin & Chaib-Draa (1996) for a general overview of the area). The first discipline, *distributed problem solving*, focuses on information division and management when a large problem is decomposed into smaller subproblems in order to obtain the optimal solution by solving independent subproblems (see Corkill (1982), Durfee, Lesser & Corkill (1987), Pope, Conry & Mayer (1992) for representative approaches). The second discipline, *multi-agent systems*, deals with the behavior management of independent entities called agents (see Stone & Veloso (2000) for a recent survey). Our approach is a distributed problem solving method. However, this paper has ties with both disciplines. Motivated both by the natural division of information among the decision makers in the real world and the need for computational tractability, we decompose a complex problem into smaller subproblems. Once we obtain the functional approximations by the distributed solution procedure, the evolution of the system has multi-agent characteristics, as each decision agent now makes its decisions independently and the behaviors of the agents are coordinated with the functional approximations. An important part of our approach is a reinforcement learning mechanism

where the agent acquires knowledge from its past experiences (i.e. the dual solution of *all* the subproblems solved by the agent). The learning of each agent is captured in a very compact way by a set of functional approximations (see Langley (1995), Mitchell (1997), Stone & Veloso (2000) for learning aspects of multi-agent systems). Learning and estimating the impact of an agent on the other agents through nonlinear functions is new and brings the aforementioned challenges in coordinating the agents.

This paper builds on Topaloglu & Powell (2002) and Godfrey & Powell (2002), who formulate our dynamic resource allocation problem as a dynamic program and propose using separable, nonlinear (specifically piecewise-linear, concave) approximations of the value function, decomposing the problem into *time stages*. Here we take the dynamic programming formulation of Topaloglu & Powell (2002), decompose each time stage of this formulation into even smaller subproblems and distribute them to multiple agents. In the fleet management example that was the motivating application for this work, the classical dynamic programming formulation solves one subproblem for each time period. Here, we decompose each time period so that we solve one subproblem for each time period and each location. For this reason, we call this decomposition scheme *spatial decomposition*.

Adaptive learning using dynamic programming approximations is a research area that has been very active within the past two decades. The previous research can be classified into two groups according to how the value function approximations are represented. The first group represents the approximations as simple table lookup functions. This amounts to maintaining a table containing the approximate value function values $\{\hat{V}(s) : s \in \mathcal{S}\}$ or the approximate Q-factor values $\{\hat{Q}(s,a) : s \in \mathcal{S}, a \in \mathcal{A}\}$, where $\mathcal{S}$ and $\mathcal{A}$ are respectively the sets of possible states and actions. There exists a number of techniques to generate the values in these tables using trajectory samples of the system, $\{s_1, a_1, s_2, a_2, \ldots\}$, where s_k and a_k are the k^{th} state and action visited. For example, temporal difference learning methods (Sutton (1984), Sutton (1988)) iteratively update $\hat{V}(s_k)$ as $\hat{V}(s^k) \leftarrow (1-\alpha)\hat{V}(s_k) + \alpha\left(c(s_k, s_{k+1}) + \hat{V}(s_{k+1})\right)$, where $c(s_k, s_{k+1})$ is the immediate reward obtained by the transition from state s_k to s_{k+1} and α is a proper step size. Note that the term in parenthesis is the new estimate of $\hat{V}(s_k)$, and we merge the old and the new estimates using a simple smoothing operation. Similarly, Q-learning (Watkins (1989), Watkins & Dayan (1992)) updates the approximate Q-values as $\hat{Q}(s_k, a_k) \leftarrow (1-\alpha)\hat{Q}(s_k, a_k) + \alpha\left(c(s_k, s_{k+1}) + \max_{a \in \mathcal{A}} \hat{Q}(s_{k+1}, a)\right)$. We note that the advantage of these methods is that they do not require the transition probabilities, which may be hard to obtain for practical problems. But maintaining the tables may become cumbersome if $\mathcal{S}$ or $\mathcal{A}$ include many elements.

The second body of work seeks more compact representations by using parameterized functional approximations. A family of functions $\{\hat{V}(\cdot, r) : r \in \mathbb{R}^n\}$, parameterized by r, is used as an approximation to the true value function and the challenge is to find a value of r such that $\hat{V}(\cdot, r)$ is a "good" approximation. This representation has to maintain the values of only n parameters rather than $|\mathcal{S}|$ or $|\mathcal{S}||\mathcal{A}|$ parameters as in table lookup methods. Off-line, least squares methods can be used to fit the approximations to the value functions estimated using sampled trajectories (Bertsekas & Tsitsiklis (1996)). Alternatively, using temporal differences we can estimate the parameters iteratively (Dayan (1992) and Tsitsiklis & Van Roy (1997) present convergence results in on-line settings). When $\hat{V}(\cdot, r)$ is linear in r, a good value of r can be obtained by solving a large linear program without using trajectory samples. These linear programs have n variables and $|\mathcal{S}||\mathcal{A}|$ constraints, and therefore, require some approximation procedure (Schweitzer & Seidmann (1985), de Farias & Van Roy (2003)).

Our approach is more akin to the parameterized approximation strategies. We utilize very compact representations of the value function approximations. For a problem with 200 resources and 100 state-type pairs, a table lookup method needs to store about 200^{100} values, whereas our approach needs to store about 100×200 slope values. Also, our approach is sampling-based and does not require the knowledge of transition probabilities. The final and the most novel aspect of our approach is revealed when we finally have a good approximation $\hat{V}(\cdot, r)$ and we want to use it to make a decision. This requires solving a problem that maximizes the sum of the immediate reward and the value function approximation. However, if the approximation does not have a special structure, solving this problem may require enumeration over all possible solutions. We are interested in large-scale applications, such as fleet management, where such enumeration is infeasible. The separable, piecewise-linear, concave approximations we choose help us to convert these problems into min-cost network flow problems, where integer solutions are obtained very fast.

In this paper we make the following research contributions: 1) We present a distributed solution approach to dynamic resource allocation problems that closely captures the multi-agent decision making structure of real world operations, and develop a dynamic programming-based learning mechanism. 2) We show how to coordinate different subproblems in order to obtain high quality solutions by using computationally tractable piecewise-linear, concave approximations of the value function. 3) We prove that the problem reduces to solving sequences of simple min-cost network flow problems. 4) We illustrate the extension of our methodology to handle multiperiod resource transformation times and derive a key result that significantly accelerates the performance of our solution scheme. 5) We numerically compare our distributed solution method with a centralized

method in Topaloglu & Powell (2002) and present what we gain or lose by decomposition.

In section 1, we define the problem, briefly describe the dynamic programming formulation due to Topaloglu & Powell (2002) and present our agent-based dynamic programming formulation. Section 2 describes the basic idea of approximating the value function and establishes that the approximate subproblems under spatial decomposition are min-cost network flow problems when we use separable, piecewise-linear, concave value function approximations. Section 3 summarizes how to update the value function approximations. Section 4 presents a series of numerical results on deterministic and stochastic experiments that test the solution quality of the new distributed solution methodology.

1 Problem Formulation

We first briefly introduce the notation necessary to formulate our dynamic resource allocation problem and present the dynamic programming formulation due to Topaloglu & Powell (2002). Then we develop the dynamic programming formulation that is amenable to distributed computation.

The problem we are interested in can be summarized as follows: We have a set of indivisible, reusable resources of different types in different states. At each decision epoch a certain number of demands enter the system. We have to decide which demands to serve and what type of resources to use to serve those demands. Serving a demand generates a contribution and changes the state of the resource, but the type of the resource remains the same. We also have the option of doing nothing with a resource or modifying the state of a resource at a cost without assigning it to a demand. The objective is to maximize the total expected contribution over all time periods.

The formulation in this section assumes that all decisions take a single time period to implement. We show how to relax this assumption in section 1.3. A rigid assumption of our model is that the resources are indivisible and each resource can serve only one demand at a time. We also employ other assumptions for the sake of presentation, such as the demands that are not served in one time period are lost. There are a number of railcar allocation and fleet management applications where these assumptions are valid. However, the strength of our approach comes from the fact that it is sampling-based. For example, if each uncovered demand is lost with a certain probability, we can use the result of a Bernoulli trial to decide whether a certain demand will be lost or available in the next time period.

We define the following:

$\mathcal{T}$ = Set of time periods over which the demands occur, $\{0, \ldots, T-1\}$.

$\mathcal{I}$ = Set of possible states for resources.

$\mathcal{K}$ = Set of resource types.

$\mathcal{L}$ = Set of demand types that can occur in the system.

$\mathcal{D}$ = Set of all decisions, $\mathcal{D} = \mathcal{D}^m \bigcup \mathcal{D}^s$, where:

$\mathcal{D}^m$ = Set of possible "modify" decisions that can be applied to resources.

$\mathcal{D}^s$ = Set of possible "service" decisions that can be applied to resources.

$\delta_{ijd} = \begin{cases} 1 & \text{if decision } d \in \mathcal{D} \text{ transforms a resource in state } i \in \mathcal{I} \text{ to state } j \in \mathcal{I} \\ 0 & \text{otherwise.} \end{cases}$

c^k_{idt} = Contribution obtained by modifying one resource of type $k \in \mathcal{K}$ in state $i \in \mathcal{I}$ by applying decision $d \in \mathcal{D}$ on it at time $t \in \mathcal{T}$. We set $c_t = \left[\, c^k_{idt} \, \right]_{k \in \mathcal{K},\ i \in \mathcal{I},\ d \in \mathcal{D}}$.

x^k_{idt} = Number of resources of type $k \in \mathcal{K}$ in state $i \in \mathcal{I}$ modified by decision $d \in \mathcal{D}$ at time $t \in \mathcal{T}$. We set $x_t = \left[\, x^k_{idt} \, \right]_{k \in \mathcal{K},\ i \in \mathcal{I},\ d \in \mathcal{D}}$.

For a simple dynamic fleet management example, $\mathcal{I}$ is the set of all locations in the network, $\mathcal{K}$ is the set of all vehicle types, $\mathcal{L}$ is the set of all types of demands that can occur in the system. The type of a demand determines the origin and the destination location, the revenue and the feasible vehicle types for that demand. An example of a modify decision would be to move empty to a certain location, whereas an example of a service decision would be to cover one demand of a certain demand type. As will be clear shortly, we consider objective functions that are linear in x_t. This excludes vehicle routing decisions and load consolidation possibilities from our formulation. Clearly, by setting $c^k_{idt} = 1$ for all $d \in \mathcal{D}^s$ and $c^k_{idt} = 0$ for all $d \in \mathcal{D}^m$, we can easily incorporate demand coverage maximization criterion as opposed to contribution maximization.

Our problem is characterized by an information process which we represent by the sequence of random variables $\{W_t : t \in \mathcal{T}\}$ defined on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. For our work, W_t describes the demands present in the system at time t. We let $\omega = (\omega_0, \omega_1, \ldots, \omega_{T-1})$ represent an elementary outcome, with $\omega_t = W_t(\omega)$ being the information arriving at time t. Throughout the paper, when referring to the realization of the random variable W_t corresponding to ω, we simply write ω_t rather than $W_t(\omega)$. The most important stochastic element of the problem is given by:

$u_{lt}(\omega_t)$ = Number of demands of type $l \in \mathcal{L}$ present in the system at time $t \in \mathcal{T}$ corresponding to outcome $\omega \in \Omega$.

ℓ_d = The demand type corresponding to service decision $d \in \mathcal{D}^s$.

$u_{lt}(\omega_t)$ simply represents our ability to execute decisions in the set $\mathcal{D}^s$. If $d \in \mathcal{D}^s$ is a service decision, then ℓ_d is the demand type that decision d is intended to serve. In our representation, $u_{\ell_d t}(\omega_t)$ is an upper bound on the number of times we can execute a particular decision $d \in \mathcal{D}^s$. To avoid notational clutter, we use $u_{dt}(\omega_t)$ for $u_{\ell_d t}(\omega_t)$.

1.1 Dynamic Programming Formulation

In this section we briefly show the formulation of the problem as a dynamic program. In order to represent the state of the system at any time period, we define:

$R_{it}^k =$ Number of resources of type $k \in \mathcal{K}$ in state $i \in \mathcal{I}$ at time $t \in \mathcal{T}$. We set $R_t = \left[R_{it}^k \right]_{k \in \mathcal{K},\ i \in \mathcal{I}}$.

Assuming all decisions take a single time period to implement (an assumption that we relax later), the dynamics of the system are given by the following equation:

$$R_{j,t+1}^k = \sum_{i \in \mathcal{I}} \sum_{d \in \mathcal{D}} \delta_{ijd}\ x_{idt}^k \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I}. \tag{1}$$

Given the state R_t and outcome ω_t, the feasible action space at time t consists of all the vectors x_t in the set $\mathcal{X}_t(R_t, \omega_t)$, where:

$$\mathcal{X}_t(R_t, \omega_t) = \{x_t : \sum_{d \in \mathcal{D}} x_{idt}^k = R_{it}^k \qquad \forall k \in \mathcal{K},\ \forall i \in \mathcal{I} \tag{2}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} x_{idt}^k \leq u_{dt}(\omega_t) \qquad \forall d \in \mathcal{D}^s \tag{3}$$

$$x_{idt}^k \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall i \in \mathcal{I},\ \forall d \in \mathcal{D}\}. \tag{4}$$

For a fixed realization of demands, the structure of the flow of resources among subproblems (i.e. the system dynamics and the constraints) can be represented using a network as in figure 2. In figure 2, we assume that $\mathcal{I} = \{a, b, c\}$ and $\mathcal{K} = \{1, 2\}$.

Assuming that given R_t, the random outcomes in time period t are conditionally independent of the random outcomes in the earlier time periods, the optimum action for each possible system state can be found by solving the following optimality equation:

$$V_t(R_t) = E\left\{ \max_{x_t \in \mathcal{X}_t(R_t, W_t)} c_t x_t + V_{t+1}(R_{t+1}) \mid R_t \right\}. \tag{5}$$

where $V_t(\cdot)$ is the value function giving the total expected contribution starting from time period t till the end of the planning horizon (see Bellman (1957), Puterman (1994)).

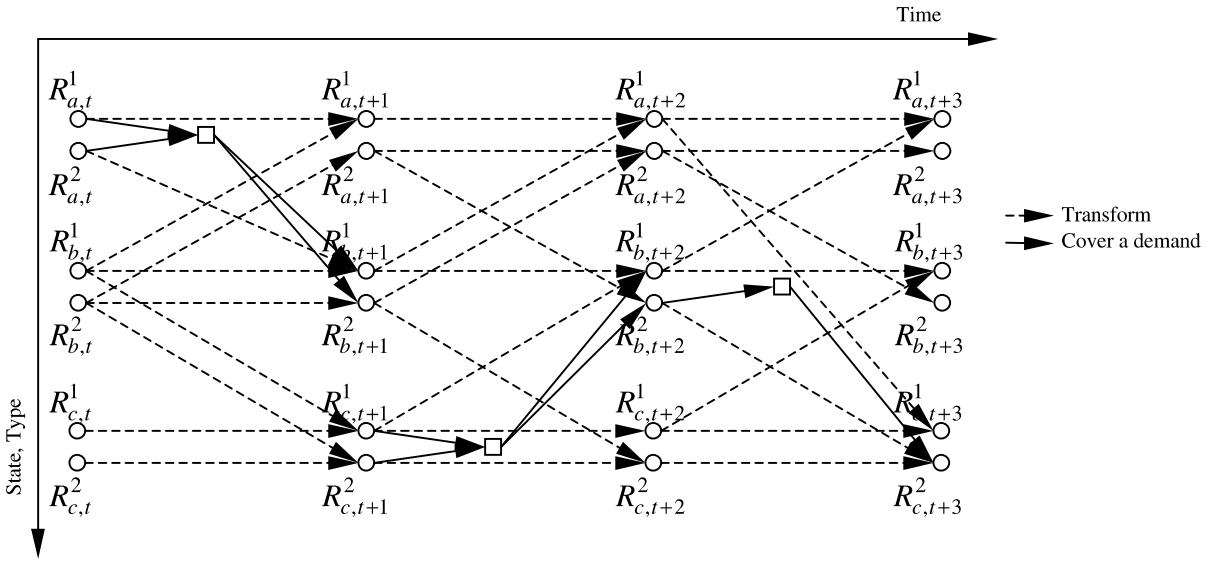

Figure 2: Pattern of flow of resources among subproblems.

We now replace the value function $V_{t+1}(\cdot)$ in (5) with an approximation $\hat{V}_{t+1}(\cdot)$, drop the expectation, solve the following problem for a particular sample realization of W_t:

$$\tilde{V}_t(R_t, \omega_t) = \max_{x_t \in \mathcal{X}_t(R_t, \omega_t)} c_t x_t + \hat{V}_{t+1}(R_{t+1}). \tag{6}$$

The value function approximation $\hat{V}_t(\cdot)$ is then iteratively updated using the information obtained from solving problem (6). In order to approximate the value function $V_t(\cdot)$, Topaloglu & Powell (2002) use a separable function $\hat{V}_t(\cdot)$ of the form:

$$\hat{V}_t(R_t) \quad = \quad \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \hat{V}_{it}^k(R_{it}^k). \tag{7}$$

Motivated by the fact that the value function in (5) is a concave function of R_t (see, for example, Van Slyke & Wets (1969)), they use linear or piecewise-linear, concave forms for each of $\hat{V}_{it}^k(\cdot)$. They show that if $\hat{V}_{it}^k(\cdot)$ is a linear function for all $k \in \mathcal{K}$, $i \in \mathcal{I}$, then subproblems of the form (6) are min-cost network flow problems and if $\hat{V}_{it}^k(\cdot)$ is a piecewise-linear, concave function for all $k \in \mathcal{K}$, $i \in \mathcal{I}$, then subproblems are min-cost integer multicommodity network flow problems. Their numerical work shows that piecewise-linear approximations provide better solution quality than linear approximations but have considerably slower run times since they require solving sequences of min-cost integer multicommodity network flow problems.

We emphasize that the approach of this section is a centralized decision making scheme where the decisions regarding all resources in a certain time period are made simultaneously. We develop this approach into a distributed decision making scheme in the next section.

1.2 Spatial Decomposition

Most of the work in the dynamic programming literature is based on setting up the subproblems according to a temporal decomposition and capturing the impact of current actions on the way the system evolves in the future. The approach in section 1.1 basically followed this framework by indexing the subproblems with respect to $t \in \mathcal{T}$. For this reason we call the approach of section 1.1 *temporal decomposition*. Under temporal decomposition, we end up with the optimality equation (5), which computes the optimal actions for all the resources simultaneously.

Our spatial decomposition approach decomposes the subproblem (5) by the elements of the resource state space $\mathcal{I}$ with the help of a carefully defined state variable and computes the actions regarding resources in different states by solving separate subproblems. Therefore the spatial decomposition scheme solves one subproblem for each pair $(it),\ i \in \mathcal{I},\ t \in \mathcal{T}$. We assume that one agent is responsible for making the decisions applied to the resources in one particular state. Therefore each element $i \in \mathcal{I}$ becomes an agent and the set of subproblems $\{(it) : t \in \mathcal{T}\}$ represents the subproblems that agent i is expected to solve.

It is also possible to collect the elements of the state space $\mathcal{I}$ into different groups $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_N\}$ (we need $\bigcup_{n=1}^{N} \mathcal{I}_n = \mathcal{I}$ and $\mathcal{I}_m \bigcap \mathcal{I}_n = \varnothing$) and assign each group $\mathcal{I}_n$ to a single agent. We can call this approach *regional decomposition* since each agent is now responsible for a collection of states (locations in the fleet management context). Below, we concentrate on spatial decomposition, but the development of regional decomposition would follow the same lines.

There are several motivations for using spatial decomposition. Most importantly, there are large scale problems arising in supply chain management and freight transportation where spatial decomposition more realistically captures the actual decision making structure by distributing the decisions regarding resources in different states to different agents. As an algorithmic appeal, it reduces the subproblem size considerably by increasing the number of subproblems solved. This in turn increases the ability of the algorithm to handle large problems. Finally the subproblems are a lot easier to solve. In particular, we show in section 2 that for a certain class of value function approximations (separable, piecewise-linear, concave) the approximate subproblems reduce to min-cost network flow problems under spatial decomposition, whereas this is not possible under temporal or regional decomposition.

To establish the sequence in which the subproblems are solved, we must define an ordering between the elements of $\mathcal{I}$. Any ordering would do. Without loss of generality, we may assume

that $\mathcal{I} = \{1, 2, \ldots, |\mathcal{I}|\}$. In this case, comparisons of the form $i < j$, for $i,\ j \in \mathcal{I}$ and operations of the form $i + 1$ for $i \in \mathcal{I}$ make sense. We also define an ordering among pairs (it) for $i \in \mathcal{I},\ t \in \mathcal{T}$, whereby $(jt') > (it)$ if $(j > i,\ t' = t)$ or $t' > t$.

We index our subproblems by the pairs $(it),\ i \in \mathcal{I},\ t \in \mathcal{T}$ as opposed to $t \in \mathcal{T}$ in temporal decomposition. We start the dynamic programming recursion starting from the subproblem $(1, 0)$ (the problem that the first agent solves in the first time period) and define the next subproblem to be solved as follows:

$$\begin{aligned}(\vec{it}) &= \text{ Subproblem to be solved after the subproblem } (it).\\ &= \begin{cases}(i+1, t) & \text{if } i < |\mathcal{I}|\\ (1, t+1) & \text{if } i = |\mathcal{I}|.\end{cases}\end{aligned}$$

Note that $(jt') > (it)$ if and only if subproblem (jt') is solved after (it).

The core idea behind spatial decomposition is as follows: As we are solving different subproblems in a certain time period corresponding to different agents in $\mathcal{I}$, we keep track of the decisions that were made by the previous agents by an augmented state variable. This is similar to keeping track of the history of a non-Markovian process in order to convert it into a Markovian process.

To formulate the problem according to spatial decomposition, we define the following:

$$\begin{aligned}\mathcal{Q} &= \text{ Set of subproblems, } \{(it) : i \in \mathcal{I},\ t \in \mathcal{T}\}.\\ R^k_{jt'(it)} &= \text{ Number of resources of type } k \in \mathcal{K} \text{ that we know will be in state } j \in \mathcal{I} \text{ at time } t' \text{ right before agent } i \text{ is to solve subproblem } (it) \in \mathcal{Q}. \text{ It is necessary that } (j \geq i,\ t' = t) \text{ or } (j \in \mathcal{I},\ t' = t + 1).\\ R_{(it)} &= \text{ Vector whose components are } \{R^k_{jt'(it)} : (k \in \mathcal{K},\ j \geq i,\ t' = t) \text{ or } (k \in \mathcal{K},\ j \in \mathcal{I},\ t' = t+1)\}.\ R_{(it)}, \text{ then, is our state variable when agent } i \text{ is solving subproblem } (it).\\ x_{(it)} &= \left[\ x^k_{idt}\ \right]_{k \in \mathcal{K},\ d \in \mathcal{D}}.\\ c_{(it)} &= \left[\ c^k_{idt}\ \right]_{k \in \mathcal{K},\ d \in \mathcal{D}}.\end{aligned}$$

Noting the elements of the vector $R_{(it)}$, we see that when agent i is solving subproblem (it), it still uses the number of resources in states other than i. But as we show shortly this information is incorporated in the subproblem minimally (such as setting some variables to constants).

In the definition of $R^k_{jt'(it)}$, the condition $(j \geq i,\ t' = t)$ or $(j \in \mathcal{I},\ t' = t+1)$ indicates that when agent i is solving the subproblem (it), there can be resources in state $j \geq i$ at time period t or there can be resources in state $j \in \mathcal{I}$ at time period $t+1$. The latter are the resources that have been sent to time $t+1$ by an agent that solved its subproblem in time period t before agent i. We note that the condition $(j \geq i,\ t' = t)$ or $(j \in \mathcal{I},\ t' = t+1)$ is equivalent to $(it) \leq (jt') \leq (|\mathcal{I}|, t+1)$.

Within the state variable $R_{(it)}$, the resources in states $\{R^k_{it(it)} : k \in \mathcal{K}\}$ are *actionable*, which is to say that they can be used to cover a demand or modified into a certain state by agent i by using a decision in subproblem (it). The resources in the states $\{R^k_{jt'(it)} : k \in \mathcal{K},\ (it) < (jt') \leq (|\mathcal{I}|, t+1)\}$ always move to state $\{R^k_{\underset{jt'(it)}{\rightarrow}} : k \in \mathcal{K},\ (it) < (jt') \leq (|\mathcal{I}|, t+1)\}$ at "time" $(\overrightarrow{it})$ and they are modified in the subsequent subproblems.

We can now write the dynamics of our system at subproblem (it) as:

$$R^k_{\underset{j,t+1,(\overrightarrow{it})}{}} = \sum_{d \in \mathcal{D}} \delta_{ijd}\ x^k_{idt} + R^k_{j,t+1,(it)} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I} \tag{8}$$

$$R^k_{\underset{jt(\overrightarrow{it})}{}} = R^k_{jt(it)} \qquad \forall k \in \mathcal{K},\ \forall j > i, \tag{9}$$

with the initial condition that $R^k_{j,t+1,(1t)} = 0$ for all $k \in \mathcal{K},\ j \in \mathcal{I}$. This initial condition is due to the fact that when we are solving the first subproblem in time period t, there cannot be any resources that have been sent to time period $t+1$ by a previous subproblem.

In order to write the feasible set of actions under spatial decomposition, we slightly change the definition of the information arrival process. We now assume that our problem is characterized by an information process which we represent by the sequence of random variables $\{W_{(it)} : (it) \in \mathcal{Q}\}$, where $W_{(it)}$ describes the demands present in the system when agent i is solving subproblem (it). We let $\omega = (\omega_{(10)}, \omega_{(20)}, \ldots, \omega_{(|\mathcal{I}|,T-1)})$ represent an elementary outcome, with $\omega_{(it)} = W_{(it)}(\omega)$ being the information arriving right before agent i solves subproblem (it). In this way, we now allow arrival of new information to the system as the agents are solving the subproblems within a given time period. Throughout the paper, when referring to the realization of the random variable $W_{(it)}$ corresponding to ω, we simply write $\omega_{(it)}$ rather than $W_{(it)}(\omega)$. Then the most important stochastic element of the problem is given by:

$u_{l(it)}(\omega_{(it)}) =$ Number of demands of type $l \in \mathcal{L}$ present in the system when agent i is solving subproblem $(it) \in \mathcal{Q}$ corresponding to outcome $\omega \in \Omega$.

For brevity, when we want to refer to the number of demands corresponding to a service decision $d \in \mathcal{D}^s$, we use $u_{d(it)}(\omega_{(it)})$ instead of $u_{\ell_d(it)}(\omega_{(it)})$. To formulate the problem under spatial decom-

position, we make the additional assumption that for each decision $d \in \mathcal{D}^s$ there is a single state $i_d \in \mathcal{I}$ that a resource has to be in to be able to serve the demand type $\ell_d \in \mathcal{L}$. This prevents a particular demand type from appearing in multiple subproblems corresponding to different elements of $\mathcal{I}$. Hence, for each demand that can occur in the system, there is exactly one agent that is able to cover that demand, and c^k_{idt} for $d \in \mathcal{D}^s$ and $i \neq i_d$ is a very high cost representing an infeasible decision (alternatively we could have the additional constraint that $x^k_{idt} = 0$ for $d \in \mathcal{D}^s$ and $i \neq i_d$). Then the constraints that limit our ability to execute a decision $d \in \mathcal{D}^s$ can be written as:

$$\sum_{k \in \mathcal{K}} x^k_{idt} \leq u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s,$$

with no need for summation over $\mathcal{I}$. Given the state $R_{(it)}$ and outcome $\omega_{(it)}$, the feasible action space at subproblem (it) consists of all the vectors $x_{(it)}$ in the set $\mathcal{X}_{(it)}(R_{(it)}, \omega_{(it)})$, where:

$$\mathcal{X}_{(it)}(R_{(it)}, \omega_{(it)}) = \{x_{(it)} : \sum_{d \in \mathcal{D}} x^k_{idt} = R^k_{it(it)} \qquad \forall k \in \mathcal{K} \tag{10}$$

$$\sum_{k \in \mathcal{K}} x^k_{idt} \leq u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s \tag{11}$$

$$x^k_{idt} \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall d \in \mathcal{D}\}. \tag{12}$$

By employing $R_{(it)}$ as our state vector for subproblem $(it) \in \mathcal{Q}$, we can write the dynamic programming formulation of the problem using spatial decomposition as follows:

$$V_{(it)}(R_{(it)}) = E\left\{ \max_{x_{(it)} \in \mathcal{X}_{(it)}(R_{(it)}, W_{(it)})} c_{(it)} x_{(it)} + V^{\rightarrow}_{(it)}(R^{\rightarrow}_{(it)}) \mid R_{(it)} \right\} \tag{13}$$

$$V_{(it)}(R_{(it)}, \omega_{(it)}) = \max_{x_{(it)} \in \mathcal{X}_{(it)}(R_{(it)}, \omega_{(it)})} c_{(it)} x_{(it)} + V^{\rightarrow}_{(it)}(R^{\rightarrow}_{(it)}). \tag{14}$$

Under spatial decomposition, the pattern of resource flows among subproblems can be represented using a network shown in figure 3. Figure 3 assumes that $\mathcal{I} = \{a, b, c\}$ with $a < b < c$ and $\mathcal{K} = \{1\}$. The grey nodes in figure 3 represent the supply of resources that the agents have to act on. White nodes represent the supply of resources over which the agent of the corresponding subproblem has no control when it is solving a particular subproblem. The reader is encouraged to compare figure 3 with figure 2.

Similar to (5), it can be shown that the value function in (13) is a concave function of $R_{(it)}$. Therefore, at this point we can apply the ideas of section 1.1 to replace the value function in (14) by a separable, concave, piecewise-linear approximation $\hat{V}_{(it)}(\cdot)$ for $(it) \in \mathcal{Q}$ to obtain an approximate subproblem for agent $i \in \mathcal{I}$ at time $t \in \mathcal{T}$:

$$\tilde{V}_{(it)}(R_{(it)}, \omega_{(it)}) = \max_{x_{(it)} \in \mathcal{X}_{(it)}(R_{(it)}, \omega_{(it)})} c_{(it)} x_{(it)} + \hat{V}^{\rightarrow}_{(it)}(R^{\rightarrow}_{(it)}). \tag{15}$$

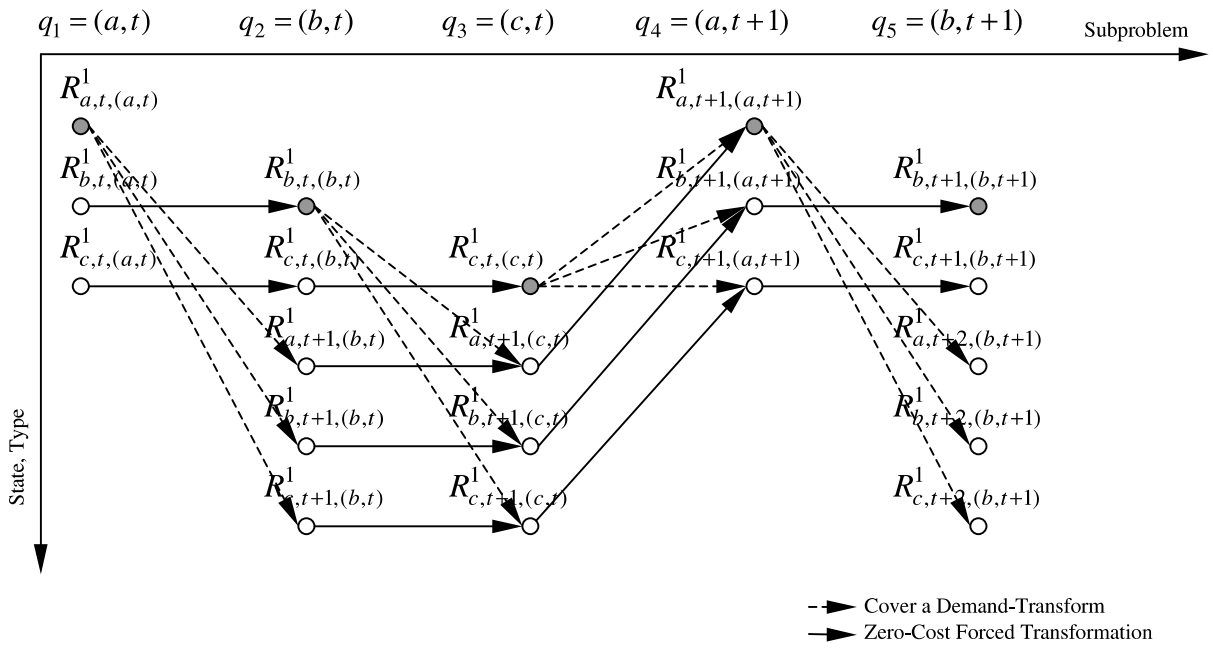

Figure 3: Pattern of flow of resources among subproblems under spatial decomposition.

The architecture of our agents can be described as follows: The set of value function approximations $\{\hat{V}^{\rightarrow}_{(it)}(\cdot) : t \in \mathcal{T}\}$ is the information available to agent i about its impact on the rest of the system. For each time period t, having perceived the state of the environment through the state vector $R_{(it)}$, agent i makes its decisions using the approximate dynamic programming recursion (15). The procedure we use to update the value function approximations $\{\hat{V}^{\rightarrow}_{(it)}(\cdot) : t \in \mathcal{T}\}$ constitutes the learning mechanism of agent i and this is the topic of section 3.

1.3 Extensions To Multiperiod Transfer Times

We now extend the formulation in section 1.2 to the case where the transfer times for the decisions are not all equal to one time period. When we discuss the state variable $R^k_{jt'(it)}$ in section 1.2, we note that we need to have $(j \geq i,\ t' = t)$ or $(j \in \mathcal{I},\ t = t+1)$, since the subproblems solved in time period t before subproblem (it) can send resources only to time period $t+1$. If all the transformation times are not equal to one time period, then a subproblem solved in time period t can send resources not only to time period $t+1$ but to time periods $t+1, t+2, \ldots, t+\tau_{\max}$, where we use $\tau_{\max}$ to denote the maximum possible transformation time. Then all we need to do is to change the definition of our state variable as follows:

$R^k_{jt'(it)} =$ Number of resources of type $k \in \mathcal{K}$ that we know will be in state $j \in \mathcal{I}$ at time

t' right before agent i is to solve subproblem $(it) \in \mathcal{Q}$. It is necessary that $(j \geq i,\ t' = t)$ or $(j \in \mathcal{I},\ t+1 \leq t' \leq t+\tau_{\max})$.

$R_{(it)} =$ Vector whose components are $\{R^k_{jt'(it)} : (k \in \mathcal{K},\ j \geq i,\ t' = t)$ or $(k \in \mathcal{K},\ j \in \mathcal{I},\ t+1 \leq t' \leq t+\tau_{\max})\}$.

To capture the richness of this problem class, we adopt a more general notation for describing the system dynamics by defining:

$$\delta_{jt'}(i,d,t) = \begin{cases} 1 & \text{if decision } d \in \mathcal{D} \text{ at time } t \in \mathcal{T} \text{ transforms a resource in state } i \in \mathcal{I} \\ & \text{to state } j \in \mathcal{I} \text{ and the transformation of decision } d \text{ ends at time } t' \in \mathcal{T} \\ 0 & \text{otherwise.} \end{cases}$$

The system dynamics are now described by:

$$\begin{aligned} R^k_{\underset{jt'(it)}{\rightarrow}} &= \sum_{d \in \mathcal{D}} \delta_{jt'}(i,d,t)\ x^k_{idt} + R^k_{jt'(it)} && \forall k \in \mathcal{K},\ \forall j \in \mathcal{I},\ t+1 \leq t' \leq t+\tau_{\max} \\ R^k_{\underset{jt(it)}{\rightarrow}} &= R^k_{jt(it)} && \forall k \in \mathcal{K},\ \forall j > i, \end{aligned}$$

with the initial condition that $R^k_{jt'(10)} = 0$ for all $k \in \mathcal{K}$, $j \in \mathcal{I}$, $1 \leq t' \leq \tau_{\max}$. The initial condition is due to the fact that when we are solving the first subproblem at time 0, there cannot be any resources sent to any time period in the future by a previous subproblem.

The feasible action space $\mathcal{X}_{(it)}(R_{(it)}, \omega_{(it)})$ is still given by the equations (10), (11) and (12) as $\{R^k_{it(it)} : k \in \mathcal{K}\}$ represents the actionable part of the state vector $R_{(it)}$. Using the new state variable, system dynamics and action space, we can write the dynamic programming formulation of the problem exactly as in (13) and (14).

2 Approximating The Value Function

Our state variable in subproblem (it) is $R_{(it)} = \{\ R^k_{jt'(it)} : (k \in \mathcal{K},\ j \geq i,\ t' = t)$ or $(k \in \mathcal{K},\ j \in \mathcal{I},\ t' = t+1)\ \}$. Hence, similar to (7), we can define our separable value function approximation as:

$$\hat{V}_{(it)}(R_{(it)}) = \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}: j \geq i} \hat{V}^k_{jt(it)}(R^k_{jt(it)}) + \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}} \hat{V}^k_{j,t+1,(it)}(R^k_{j,t+1,(it)}). \tag{16}$$

In addition to modeling the actual control structure of large-scale problems more accurately, spatial decomposition has a nice algorithmic feature, which is that the subproblems are min-cost

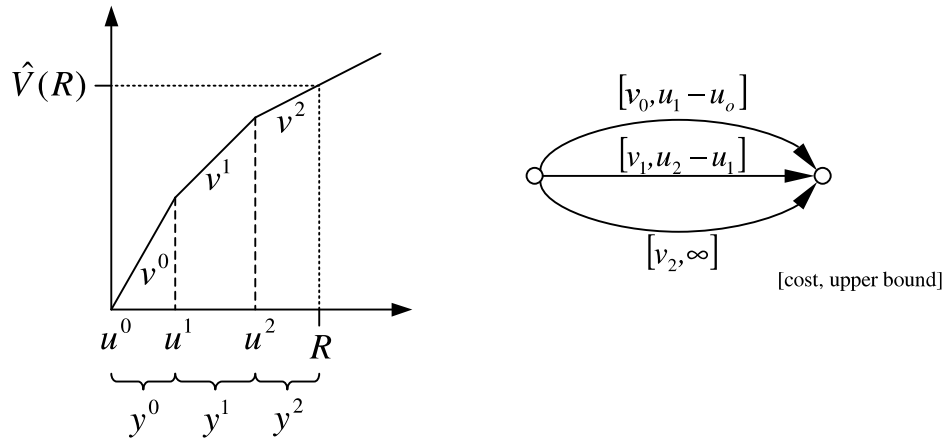

Figure 4: Representing piecewise-linear, concave functions in networks.

network flow problems when we use piecewise-linear, concave value function approximation components in (16). This is established in the following proposition. Topaloglu & Powell (2002) note that under temporal decomposition, the subproblems are min-cost integer multicommodity network flow problems if they use piecewise-linear, concave value function approximation components in (7).

Proposition 1 *Consider the approximate subproblem (15) with:*

$$\hat{V}_{\vec{(it)}}(R_{\vec{(it)}}) = \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}: j > i} \hat{V}^k_{\vec{jt(it)}}(R^k_{\vec{jt(it)}}) + \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}} \hat{V}^k_{\vec{j,t+1,(it)}}(R^k_{\vec{j,t+1,(it)}}),$$

where the value function approximation components $\{ \hat{V}^k_{\vec{jt(it)}}(\cdot) : k \in \mathcal{K},\ j > i \}$ *and* $\{ \hat{V}^k_{\vec{j,t+1,(it)}}(\cdot) : k \in \mathcal{K},\ j \in \mathcal{I} \}$ *are piecewise-linear, concave functions. Then the approximate subproblem (15) is a min-cost network flow problem.*

The proof of proposition 1 is given in the appendix. Here, we explain the ideas of the proof that are relevant to the rest of the paper. We assume that each value function approximation component $\hat{V}^k_{\vec{j,t+1,(it)}}(\cdot)$ is characterized by a series of points $\{ u^{km}_j : m \in \mathcal{N}^k_j \}$ and slopes $\{ v^{km}_j : m \in \mathcal{N}^k_j \}$ such that the slope of the function $\hat{V}^k_{\vec{j,t+1,(it)}}(\cdot)$ on the interval $(u^{km}_j, u^{k,m+1}_j)$ is v^{km}_j (see figure 4). For simplicity, we take $\mathcal{N}^k_j = \{0, 1, \ldots, |\mathcal{N}^k_j|\}$. We denote the flow over the interval $(u^{km}_j, u^{k,m+1}_j)$ in the domain of $\hat{V}^k_{\vec{j,t+1,(it)}}(\cdot)$ by y^{km}_j as shown in figure 4. Then the approximate subproblem (15) can be written explicitly as:

$$\tilde{V}_{(it)}(R_{(it)}, \omega_{(it)}) = \max \sum_{k \in \mathcal{K}} \sum_{d \in \mathcal{D}} c^k_{idt}\ x^k_{idt} + \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}: j > i} \hat{V}^k_{\vec{jt(it)}}(R^k_{\vec{jt(it)}}) + \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}} \sum_{m \in \mathcal{N}^k_j} v^{km}_j y^{km}_j \quad (17)$$

$$\text{s.t} \quad \sum_{d\in\mathcal{D}} x^k_{idt} = R^k_{it(it)} \qquad \forall k \in \mathcal{K} \tag{18}$$

$$R^k_{\underset{jt(it)}{\rightarrow}} = R^k_{jt(it)} \qquad \forall k \in \mathcal{K},\ \forall j > i \tag{19}$$

$$-\sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt} + R^k_{\underset{j,t+1,(it)}{\rightarrow}} = R^k_{j,t+1,(it)} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I} \tag{20}$$

$$\sum_{k\in\mathcal{K}} x^k_{idt} \le u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s \tag{21}$$

$$R^k_{\underset{j,t+1,(it)}{\rightarrow}} - \sum_{m\in\mathcal{N}^k_j} y^{km}_j = 0 \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I} \tag{22}$$

$$0 \le y^{km}_j \le u^{k,m+1}_j - u^{km}_j \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I},\ \forall m \in \mathcal{N}^k_j \tag{23}$$

$$x^k_{idt} \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall d \in \mathcal{D}. \tag{24}$$

Constraints (18), (21) and (24) define the set of feasible decisions as in (10), (11) and (12), whereas constraints (19) and (20) define the dynamics of the system as in (9) and (8). The other constraints are simply used to compute the value function approximations. We note that constraints (19) can be dropped, since they merely set the variable $R^k_{\underset{jt(it)}{\rightarrow}}$ to a constant, and we simply plug the value of $R^k_{\underset{jt(it)}{\rightarrow}}$ in the objective function. Furthermore (20) and (22) can be combined as

$$-\sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt} + \sum_{m\in\mathcal{N}^k_j} y^{km}_j = R^k_{j,t+1,(it)} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I}.$$

We show the structure of the problem (17) in figure 5. Constraints (18) are the flow balance constraints for the nodes on the left side of the figure. The combined form of constraints (20) and (22) are the flow balance constraints for the nodes on the right side of the figure. x^k_{idt} represents the flow on an arc leaving node $k \in \mathcal{K}$ on the left side and going into the node $k \in \mathcal{K},\ j \in \mathcal{I}$ with $\delta_{ijd} = 1$ on the right side. Note that constraints (21) are generalized upper bound constraints that limit the total flow on a certain set of arcs. (These constraints seemingly prevent problem (17) to be formulated and solved as a min-cost network flow problem, but proposition 1 shows that problem (17) can be converted into an equivalent min-cost network flow problem.) The parallel arcs leaving the nodes on the right side of the figure represent the value function approximations. If the problem in figure 5 were in fleet management context, the nodes on the left side would represent the supply of different types of vehicles in location $i \in \mathcal{I}$ at time $t \in \mathcal{T}$. The nodes on the right side would represent the locations that the vehicles in location i can be dispatched to. We would have bundle constraints to limit the total number of a certain service decision applied to all vehicles of all types.

The proof of proposition 1 transforms this problem to the following, which can be shown to be

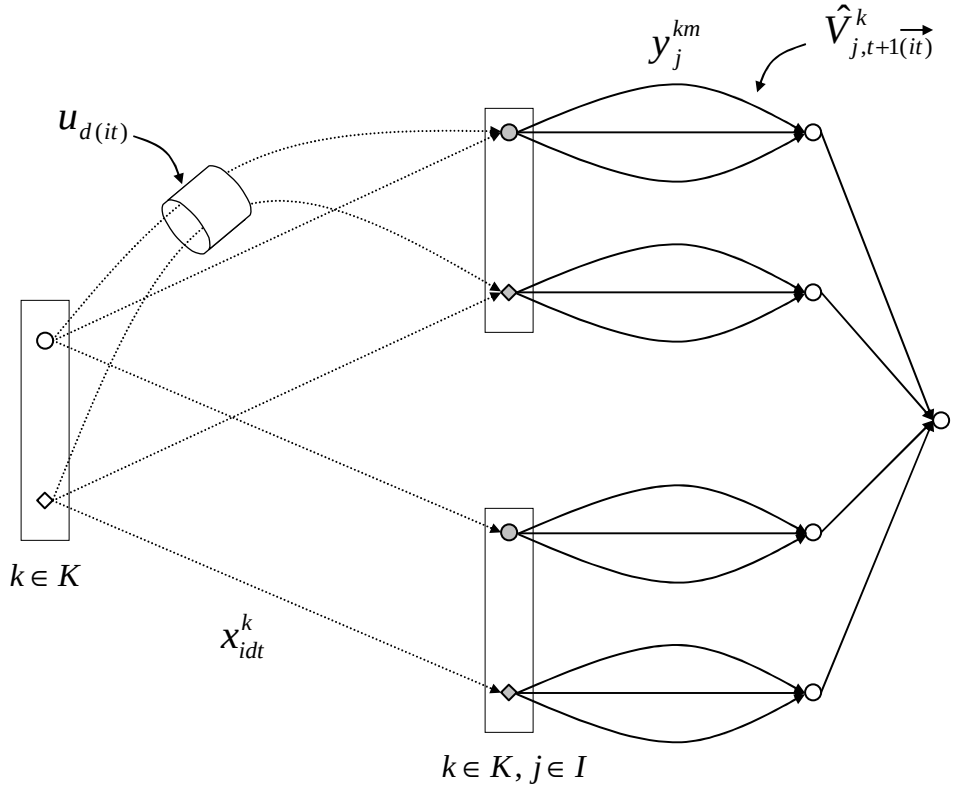

Figure 5: Representation of problem (17). Note that $u_{d(it)}(\omega_{(it)}$ act as generalized upper bounds constraining the total flow on several arcs.

a min-cost network flow problem:

$$\tilde{V}_{(it)}(R_{(it)}, \omega_{(it)}) = \max \sum_{k \in \mathcal{K}} \sum_{d \in \mathcal{D}} c_{idt}^{k} x_{idt}^{k} + \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{I}} \sum_{m \in \mathcal{N}_j^k} v_j^{km} y_j^{km} \tag{25}$$

s.t

$$\sum_{j \in \mathcal{I}} \sum_{m \in \mathcal{N}_j^k} y_j^{km} = \sum_{j \in \mathcal{I}} R_{j,t+1,(it)}^{k} + R_{it(it)}^{k} \qquad \forall k \in \mathcal{K} \tag{26}$$

$$\sum_{d \in \mathcal{D}} \delta_{ijd}\, x_{idt}^{k} - \sum_{m \in \mathcal{N}_j^k} y_j^{km} = -R_{j,t+1,(it)}^{k} \qquad \forall j \in \mathcal{I},\ \forall k \in \mathcal{K} \tag{27}$$

$$\sum_{k \in \mathcal{K}} x_{idt}^{k} - \eta_d = 0 \qquad \forall d \in \mathcal{D}^s \tag{28}$$

$$\eta_d \leq u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s \tag{29}$$

$$0 \leq y_j^{km} \leq u_j^{k,m+1} - u_j^{km} \qquad \forall j \in \mathcal{I},\ \forall k \in \mathcal{K},\ \forall m \in \mathcal{N}_j^k \tag{30}$$

$$x_{idt}^{k} \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall d \in \mathcal{D}. \tag{31}$$

3 Updating The Value Function Approximations

Godfrey & Powell (2001) describe updating piecewise-linear, concave value function approximations

in detail. In this section we summarize how to update the value function approximations and resolve the complications that arise because of the fact that we solve the min-cost network flow problem (25) instead of the original problem (17).

After we solve the approximate problem (17), we use the following to update the value function approximations:

1. The optimal dual variables $\{\pi^k_{it(it)} : k \in \mathcal{K}\}$ of constraints (18) to update the value function approximation components $\{\hat{V}^k_{it(it)}(\cdot) : k \in \mathcal{K}\}$.

2. The optimal dual variables $\{\pi^k_{jt(it)} : k \in \mathcal{K},\ j > i\}$ of constraints (19) to update the value function approximation components $\{\hat{V}^k_{jt(it)}(\cdot) : k \in \mathcal{K},\ j > i\}$.

3. The optimal dual variables $\{\pi^k_{j,t+1,(it)} : k \in \mathcal{K},\ j \in \mathcal{I}\}$ of constraints (20) to update the value function approximation components $\{\hat{V}^k_{j,t+1,(it)}(\cdot) : k \in \mathcal{K},\ j \in \mathcal{I}\}$.

Note that the vector of dual variables $\{\ \pi^k_{jt'(it)} : (k \in \mathcal{K}, j \geq i, t' = t)$ or $(k \in \mathcal{K}, j \in \mathcal{I}, t' = t{+}1)\ \}$ gives a subgradient of $\tilde{V}_{(it)}(\cdot, \omega_{(it)})$ at $R_{(it)}$. Hence, we use $\pi^k_{jt'(it)}$ as an estimate of the subgradient of $\hat{V}^k_{jt'(it)}(\cdot)$ at the point $R^k_{jt'(it)}$ and smooth the slope of the function $\hat{V}^k_{jt'(it)}(\cdot)$ around $R^k_{jt'(it)}$ by $\pi^k_{jt'(it)}$. We summarize the details below:

Let $\hat{V}(\cdot)$ be a one dimensional, piecewise-linear, concave function characterized by a series of points $\{u^0, \ldots, u^n\}$ and slopes $\{v^0, \ldots, v^n\}$ such that the slope the function $\hat{V}(\cdot)$ on the interval (u^m, u^{m+1}) is v^m. Let π be an estimate of the subgradient of $\hat{V}(\cdot)$ at point R and $0 \leq ß \leq 1$ be a step size parameter.

Step 1. Update the points: If $R = u^m$ for some m, then the set of points and slopes remain as $\{u^0, \ldots, u^n\}$ and $\{v^0, \ldots, v^n\}$. Otherwise let $u^m < R < u^{m+1}$. Then let the new set of points and slopes be $\{u^0, \ldots, u^m, R, u^{m+1}, \ldots, u^n\}$, $\{v^0, \ldots, v^m, v^m, v^{m+1}, \ldots, v^n\}$.

Step 2. Find the smoothing interval:

$$\mathcal{J}^+ = \min\{\ j : R \leq u^j,\ (1-ß)v^j + ß\pi \geq v^{j+1}\ \}$$
$$\mathcal{J}^- = \max\{\ j : u^j < R,\ v^{j-1} \geq (1-ß)v^j + ß\pi\ \}.$$

Step 3. Update the slopes by:

$$v^j \leftarrow (1-ß)v^j + ß\pi \text{ for all } j \in [\mathcal{J}^-, \mathcal{J}^+].$$

Step 2 ensures that the concavity of the approximation is preserved after the update. This step finds the smallest interval around R such that if we update the slopes over this interval by π, the concavity of the approximation is maintained.

In practice we solve problem (25) which is the min-cost network flow transformation of problem (17). Therefore we need to obtain an optimal dual solution to (17) using an optimal dual solution to (25). The optimal values of the dual variables of constraints (19) are obvious since they merely set the variables $\{R^k_{\underset{jt(it)}{\rightarrow}} : k \in \mathcal{K}, j > i\}$ to constants $\{R^k_{jt(it)} : k \in \mathcal{K}, j > i\}$. We then need to establish the relationship between the optimal dual variables of the constraints (18), (20) and (26), (27). The following proposition shown in the appendix does this:

Proposition 2 *Let $\{\pi^k_{it(it)} : k \in \mathcal{K}\}$ and $\{\pi^k_{j,t+1,(it)} : k \in \mathcal{K},\ j \in \mathcal{I}\}$ be the optimal dual variables corresponding to constraints (26) and (27) in problem (25). Then $\{\bar{\pi}^k_{it(it)} : k \in \mathcal{K}\} = \{\pi^k_{it(it)} : k \in \mathcal{K}\}$ and $\{\bar{\pi}^k_{j,t+1,(it)} : k \in \mathcal{K},\ j \in \mathcal{I}\} = \{-(\pi^k_{j,t+1,(it)} - \pi^k_{it(it)}) : k \in \mathcal{K},\ j \in \mathcal{I}\}$ are the optimal dual variables corresponding to constraints (18) and (20) in problem (17).*

The second complication that arises from using spatial decomposition is that there are a lot of "forced" transformations represented by the bold arcs going out of the white nodes in figure 3. At one subproblem there are many resources that simply flow to the next subproblem (see constraints (9)) rather than being modified by applying decisions to them by the relevant agent. This interferes with the efficient communication between the subproblems and the performance of the algorithm deteriorates. In the remainder of this section we present a result which alleviates the communication lag problem and increases the performance of the algorithm dramatically. We fix $\omega \in \Omega$ and for a given state $R_{(it)}$ in subproblem $(it) \in \mathcal{Q}$, we define the problem $P_{(it)}[R_{(it)}, \omega]$ and its optimal objective value $F_{(it)}(R_{(it)}, \omega)$ as:

$$P_{(it)}[R_{(it)}, \omega]: \qquad F_{(it)}(R_{(it)}, \omega) = \max \sum_{(lt') \geq (it)} c_{(lt')}\ x_{(lt')} \tag{32}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_{d \in \mathcal{D}} \delta_{ljd}\ x^k_{ldt'} + R^k_{j,t'+1,(lt')} - R^k_{\underset{j,t'+1,(lt')}{\rightarrow}} &= 0 && \forall k \in \mathcal{K},\ \forall (lt') \geq (it), \forall j \in \mathcal{I} \\
R^k_{jt'(lt')} - R^k_{\underset{jt'(lt')}{\rightarrow}} &= 0 && \forall k \in \mathcal{K},\ \forall (lt') \geq (it), \forall j > l \\
x_{(lt')} &\in \mathcal{X}_{(lt')}(R_{(lt')}, \omega_{t'}) && \forall (lt') \geq (it).
\end{aligned}$$

This problem may be considered as the *posterior* multistage problem corresponding to $\omega \in \Omega$. We let $\{x^*_{(it)} : (it) \in \mathcal{Q}\}$ and $\{R^*_{(it)} : (it) \in \mathcal{Q}\}$ be the series of actions and states in the optimal solution

for $P_{(1,0)}[R_{(1,0)}, \omega]$, which corresponds to the first subproblem $(1,0)$. We omit their dependence on ω for clarity of the notation. Then, we have:

$$F_{(it)}(R^*_{(it)}, \omega) = c_{(it)}x^*_{(it)} + F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)}, \omega) \qquad \forall (it) \in \mathcal{Q}.$$

We now fix subproblem (it) and state $j \in \mathcal{I}$ with $j > i$ and $k \in \mathcal{K}$. We define:

$R^*_{(it)} + e^k_{jt} =$ The state variable obtained by increasing the component $R^k_{jt(it)}$ in $R^*_{(it)}$ by 1.

We note that since $j > i$, the component $R^k_{jt(it)}$ in the state vector $R^{\rightarrow}_{(it)}$ is also defined.

Proposition 3 $F_{(it)}(R^*_{(it)} + e^k_{jt}, \omega) - F_{(it)}(R^*_{(it)}, \omega) \geq F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)} + e^k_{jt}, \omega) - F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)}, \omega)$ *for all* $\omega \in \Omega$.

Proof: We know that:

$$F_{(it)}(R^*_{(it)}, \omega) = c_{(it)}x^*_{(it)} + F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)}, \omega).$$

Also, given the state vector $R^*_{(it)} + e^k_{jt}$ and outcome ω, the actions $x^*_{(it)}$ are again feasible for problem (32) since the component $R^k_{it(it)}$ is the same in $R^*_{(it)}$ and $R^*_{(it)} + e^k_{jt}$, and $x^*_{(it)}$ must satisfy the constraints (10), (11), (12) which depend only on ω and $R^k_{it(it)}$. Finally we note that applying the decision vector $x^*_{(it)}$ to the state vector $R^*_{(it)} + e^k_{jt}$ generates the state vector $R^{*\rightarrow}_{(it)} + e^k_{jt}$. Then,

$$F_{(it)}(R^*_{(it)} + e^k_{jt}, \omega) \geq c_{(it)}x^*_{(it)} + F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)} + e^k_{jt}, \omega).$$

Thus,

$$\begin{aligned} F_{(it)}(R^*_{(it)} + e^k_{jt}, \omega) - F_{(it)}(R^*_{(it)}, \omega) &\geq c_{(it)}x^*_{(it)} + F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)} + e^k_{jt}, \omega) - c_{(it)}x^*_{(it)} - F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)}, \omega) \\ &= F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)} + e^k_{jt}, \omega) - F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)}, \omega). \quad \square \end{aligned}$$

We now fix subproblem (it), state $j \in \mathcal{I}$ and $k \in \mathcal{K}$. We define:

$R^*_{(it)} + e^k_{j,t+1} =$ The state variable obtained by increasing the component $R^k_{j,t+1,(it)}$ of $R^*_{(it)}$ by 1.

We note that the component $R^k_{j,t+1,(it)}$ in the state vector $R^{k\rightarrow}_{(it)}$ is also defined.

Proposition 4 $F_{(it)}(R^*_{(it)} + e^k_{j,t+1}, \omega) - F_{(it)}(R^*_{(it)}, \omega) \geq F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)} + e^k_{j,t+1}, \omega) - F^{\rightarrow}_{(it)}(R^{*\rightarrow}_{(it)}, \omega)$ *for all* $\omega \in \Omega$.

The proof of this proposition is very similar to the previous one.

We define, for the values of subproblem $(it) \in \mathcal{Q}$, $j \in \mathcal{I}$ and $k \in \mathcal{K}$ that we fixed before:

$$\begin{aligned} \Pi^k_{jt(it)} &= F_{(it)}(R^*_{(it)} + e^k_{jt}, \omega) - F_{(it)}(R^*_{(it)}, \omega) \\ \Pi^k_{j,t+1,(it)} &= F_{(it)}(R^*_{(it)} + e^k_{j,t+1}, \omega) - F_{(it)}(R^*_{(it)}, \omega). \end{aligned}$$

Then the following two lines of inequalities are implied by propositions 4 and 3 respectively which proves corollary 1:

$$\begin{aligned} \Pi^k_{j,t+1,(it)} \geq \Pi^k_{j,t+1,(i+1,t)} \geq \cdots &\geq \Pi^k_{j,t+1,(|\mathcal{I}|,t)} \geq \Pi^k_{j,t+1,(1,t+1)} = \\ &= \Pi^k_{j,t+1,(1,t+1)} \geq \Pi^k_{j,t+1(2,t+1)} \geq \ldots \geq \Pi^k_{j,t+1,(j,t+1)}. \end{aligned}$$

Corollary 1 *For all* $k \in \mathcal{K},\ i \in \mathcal{I},\ t \in \mathcal{T}$

$$\Pi^k_{jt'(it)} = \max_{(it) \leq q' \leq (jt')} \{\Pi^k_{jt',q'}\},$$

where $(t' = t,\ j \geq i)$ *or* $t' = t + 1$.

Therefore, if we denote the optimal dual variables of the constraints (18), (19), (20), at any one iteration by $\{\pi^k_{jt'(it)} : k \in \mathcal{K},\ (it) \leq (jt') \leq (|\mathcal{I}|, t+1)\}$, then in order to update the value function component $\hat{V}^k_{jt'(it)}(\cdot)$ we use $\max_{(it) \leq q' \leq (jt')}\{\pi^k_{jt',q'}\}$ rather than $\pi^k_{jt'(it)}$. This result proves to be instrumental in the performance of the algorithm. In figure 6, we compare the performance of spatial decomposition (using and not using corollary 1) with temporal decomposition of Topaloglu & Powell (2002) on a deterministic instance of the problem (see section 4 for the experimental setup). Corollary 1 helps spatial decomposition give good quality solutions quickly. Without using the corollary, our algorithm has to continue for about 100 iterations to reach the solution quality that can be obtained within 6 iterations by temporal decomposition.

4 Experimental Results

In this section, we explore the effectiveness of the spatial decomposition scheme on the specific fleet management problem described in the introduction and compare it with the centralized temporal decomposition. Our experimental setup closely follows that of Topaloglu & Powell (2002). We create a basic problem and modify different attributes of this problem to create new problems to test the robustness and effectiveness of the spatial decomposition to problems with different characteristics. Table 1 describes the attributes of this basic problem.

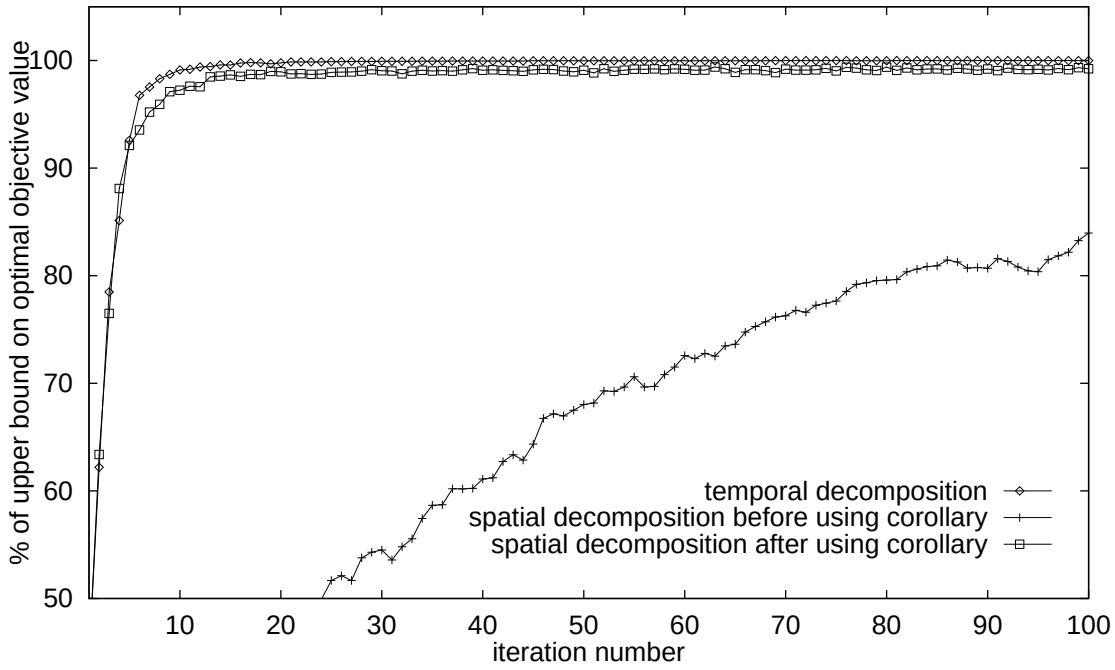

Figure 6: Increase in the performance of spatial decomposition using corollary 1.

No. of time periods, $\|\mathcal{T}\|$	60
No. of locations (agents), $\|\mathcal{I}\|$	20
No. of vehicle types, $\|\mathcal{K}\|$	5
No. of demand compatibility	5
No. of vehicles	200
No. of demands	4000

Table 1: Basic problem characteristics.

4.1 Deterministic Experiments

Problems with deterministic demand arrival processes can be formulated as large integer programs and we can obtain upper bounds on the optimal objective values by solving the linear programming relaxations of these problems. Therefore it is relatively easy to measure the solution quality of our methodology in a deterministic setting.

In order to evaluate the quality of the solutions provided by the spatial decomposition scheme, we normalize the objective value of each iteration to 100 by using the objective value of the linear programming relaxation of the problem. The criteria we use to evaluate the algorithm are the mean, standard deviation, maximum and minimum of the objective value in the last 30 of the 100 iterations, CPU time and the number of iterations to reach a certain percent of the upper bound on the objective value. As a comparison, we also show the performance of the temporal decomposition

Problem	Alg.	Max.	Mean	Time (sec.) to reach				No. iterations for				Time (s)	Time (s)
				85	90	95	97.5	85	90	95	97.5	per iter.	per subpr.
30	T	98.58	98.37	19	35	165	276	2	3	10	15	24.9	0.830
periods	S	98.63	98.15	35	105	296	1031	3	7	15	40	28.3	0.047
60	T	98.90	98.76	37	101	248	506	2	4	8	14	46.6	0.777
periods	S	99.04	98.67	98	169	536	1368	4	6	14	28	58.3	0.049
90	T	98.89	98.75	56	106	276	721	2	3	6	13	74.9	0.832
periods	S	98.76	98.55	97	251	745	1932	3	6	13	26	89.0	0.049

Table 2: Results of deterministic runs with varying horizon length and number of demands.

Problem	Alg.	Max.	Mean	Time (sec.) to reach				No. iterations for				Time (s)	Time(s)
				85	90	95	97.5	85	90	95	97.5	per iter.	per subpr.
10	T	99.75	99.65	11	11	30	63	2	2	4	7	14.5	0.242
locations	S	99.53	99.37	13	24	84	142	2	3	8	12	18.3	0.031
20	T	98.90	98.76	37	101	248	506	2	4	8	14	46.6	0.777
locations	S	99.04	98.67	98	169	536	1368	4	6	14	28	58.3	0.049
40	T	98.90	98.52	194	530	992	2276	3	6	9	17	154.2	2.570
locations	S	98.97	98.64	695	1552	2633	5248	7	13	19	33	176.9	0.074

Table 3: Results of deterministic runs with varying number of locations.

scheme described in section 1.1.

Table 2 shows the results for different numbers of time periods, table 3 shows the results for different numbers of locations, table 4 shows the results for different compatibility patterns between resources and demands, and finally table 5 shows the results with different numbers of resources. "T" denotes the temporal decomposition of section 1.1, "S" denotes the spatial decomposition of section 1.2 in these tables. Note that one problem in each table corresponds to the basic problem we describe in table 1, and the rest of the problems are variations of this basic problem.

Both temporal and spatial decomposition yields results that are very close to the objective value of the linear programming relaxation of the problem. However temporal decomposition takes fewer iterations than spatial decomposition to reach a certain percent of the objective upper bound. Although we do not present here, we have also run the problems in tables 2, 3, 4 and 5 with demand coverage maximization criterion. (This is achieved by setting $c^k_{idt} = 1$ for all $d \in \mathcal{D}^s$ and $c^k_{idt} = 0$ for all $d \in \mathcal{D}^m$.) Spatial decomposition performed very close to temporal decomposition and provided near optimal results also under this performance measure.

4.2 Stochastic experiments

In this section we show the performance of spatial decomposition on problems with stochastic demand arrival processes. Testing the algorithm on a stochastic problem involves two stages:

Problem	Alg.	Max.	Mean	Time (sec.) to reach				No. iterations for				Time (s)	Time (s)
				85	90	95	97.5	85	90	95	97.5	per iter.	per subpr.
Compatibility	T	98.90	98.76	37	101	248	506	2	4	8	14	46.6	0.777
pattern I	S	99.04	98.67	98	169	536	1368	4	6	14	28	58.3	0.049
Compatibility	T	99.51	99.34	59	59	433	991	2	2	7	13	75.5	1.258
pattern II	S	98.94	98.67	67	129	420	714	3	5	12	22	60.1	0.050
Compatibility	T	98.61	98.41	33	88	374	505	2	4	12	15	44.8	0.747
pattern III	S	98.83	98.64	94	231	499	1329	4	8	14	29	54.1	0.045
Compatibility	T	99.86	99.75	235	287	479	938	4	5	9	14	82.4	1.373
pattern IV	S	99.24	98.89	68	101	372	748	3	4	10	16	71.1	0.059

Table 4: Results of deterministic runs with varying compatibility patterns.

Problem	Alg.	Max.	Mean	Time (sec.) to reach				No. iterations for				Time (s)	Time (s)
				85	90	95	97.5	85	90	95	97.5	per iter.	per subpr.
100	T	96.87	96.48	66	384	475		3	12	14		50.2	0.837
resources	S	97.09	96.53	119	343	677		5	12	20		41.6	0.035
200	T	98.90	98.76	37	101	248	506	2	4	8	14	46.6	0.777
resources	S	99.04	98.67	98	169	536	1368	4	6	14	28	58.3	0.049
400	T	99.52	99.43	40	40	140	419	2	2	5	12	48.8	0.813
resources	S	98.66	98.50	71	136	431	1124	3	5	12	24	61.7	0.051

Table 5: Results of deterministic runs with varying number of resources.

training and evaluation. In the training stage, we sample the demands that occur in the system, solve approximate subproblems of form (15) using this sample realization and update the value function approximations. In the evaluation stage we still continue sampling demands and solving the approximate subproblems, however we do not update the value function approximations anymore. In essence, the evaluation stage measures the quality of the value function approximations obtained in the training stage. Our setup runs showed that the value function approximations stabilize after around 70 iterations. Hence in our experiments the training stage is composed of 70 sample realizations whereas the evaluation stage is composed of 50.

The main difficulty in evaluating the performance of the algorithm under stochastic demand arrivals is obtaining a tight bound on the objective value of the problem. For this purpose, we first solve the linear programming relaxation of the deterministic posterior problem corresponding to each sample realization of the demands in the evaluation stage. This gives an upper bound on the objective value of the problem if we were to solve it as a dynamic program and the realization of the demands turned out to be the same as the one used in the deterministic linear program. This bound is likely to be quite loose. However, we utilize it only to normalize our results.

As a competing strategy to our algorithm, we employ 20-period rolling horizon procedure (denoted by "RH" in the tables that follow). Topaloglu & Powell (2002) show that using 20 periods for the rolling horizon provides the best results for the basic problem in table 1.

Problem	Alg.	Mean	Std. dev.	Percentiles 50^{th}	5^{th}	95^{th}	Time (s) per iter.	Time (s) per subpr.
10	T	96.96	0.220	96.99	96.61	97.31	14.8	0.246
locations	S	94.06	0.491	93.98	93.26	94.84	18.6	0.031
	RH	93.17	0.431	93.26	92.47	93.82		
20	T	93.28	0.422	93.35	92.51	93.90	46.9	0.781
locations	S	89.63	0.535	89.66	88.64	90.34	58.6	0.049
	RH	86.84	0.545	86.88	86.11	87.56		
40	T	92.21	0.465	92.89	91.55	93.22	154.3	2.572
locations	S	88.17	0.662	88.22	87.14	89.35	177.4	0.074
	RH	86.89	0.772	86.77	85.08	87.11		

Table 6: Results of stochastic runs with varying number of locations.

The criteria we use to evaluate the performance of the algorithm are the mean, standard deviation, 50^{th}, 5^{th} and 95^{th} percentile of the objective value in the evaluation iterations. We note that the objective value of each iteration is normalized to 100 using the deterministic linear program corresponding to the demand realization of the iteration in question. The experimental setup is similar to the one used in deterministic experiments. We take the basic problem defined in table 1 and vary the number of locations, compatibility structures and number of resources. The only difference is that the number of demands in table 1 now becomes the expected number of demands. We assume that the numbers of demands that need to be carried between each origin-destination pair at each time period are independent random variables with Poisson distribution.

The 20-period rolling horizon procedure solves a problem over 20 time periods for every time period, making it much slower than our methodology. Furthermore, the solution quality of our approach is better than that of rolling horizon. Hence, our approach is obviously much faster than the rolling horizon without sacrificing the solution quality and we do not report the run times for the rolling horizon procedure.

Tables 6, 7 and 8 summarize the results of stochastic experiments with different numbers of locations, compatibility patterns and numbers of resources. Both temporal and spatial decomposition yield better results than the rolling horizon procedure, but in a stochastic setting, temporal decomposition seems to be more effective than spatial decomposition. However, considering the fact that the performance gap between temporal and spatial decomposition is about 3% and subproblem solution times under spatial decomposition are 10-30 times faster than those under temporal decomposition, spatial decomposition may be preferable for large applications if fast decision making is critical.

Problem	Alg.	Mean	Std. dev.	Percentiles 50^{th}	5^{th}	95^{th}	Time (s) per iter.	Time (s) per subpr.
Compatibility pattern I	T	93.28	0.422	93.35	92.51	93.90	46.9	0.781
	S	89.63	0.535	89.66	88.64	90.34	58.6	0.049
	RH	86.84	0.545	86.88	86.11	87.56		
Compatibility pattern II	T	95.40	0.379	95.37	94.78	95.83	75.1	1.252
	S	92.42	0.583	92.38	91.34	93.21	60.5	0.050
	RH	90.87	0.641	90.83	89.89	91.91		
Compatibility pattern III	T	91.51	0.423	91.50	90.87	92.19	45.4	0.757
	S	87.08	0.502	87.17	86.27	87.76	55.2	0.046
	RH	82.66	0.695	82.65	81.54	83.74		
Compatibility pattern IV	T	97.12	0.409	97.20	96.38	97.70	82.6	1.377
	S	95.66	0.565	95.85	94.02	96.62	71.0	0.059
	RH	93.74	0.544	93.75	92.90	94.59		

Table 7: Results of stochastic runs with varying compatibility patterns.

Problem	Alg.	Mean	Std. dev.	Percentiles 50^{th}	5^{th}	95^{th}	Time (s) per iter.	Time (s) per subpr.
100 resources	T	84.87	0.591	84.83	84.00	85.92	50.6	0.843
	S	81.29	0.585	81.37	80.37	82.17	41.2	0.034
	RH	76.81	0.726	76.82	75.57	78.02		
200 resources	T	93.28	0.422	93.35	92.51	93.90	46.9	0.781
	S	89.63	0.535	89.66	88.64	90.34	58.6	0.049
	RH	86.84	0.545	86.88	86.11	87.56		
400 resources	T	96.51	0.314	96.54	95.95	96.94	49.1	0.818
	S	93.16	0.413	93.20	92.59	93.68	62.6	0.052
	RH	91.67	0.503	91.64	90.88	92.51		

Table 8: Results of stochastic runs with varying number of resources.

5 Conclusions

In this paper we proposed a new dynamic programming-based distributed solution approach to the dynamic resource allocation problem which gives the control of resources in different states to different agents. We showed how to construct a learning and communication mechanism among the agents by using piecewise-linear, concave approximations of the value function. We showed that this approach yields integer solutions naturally and provides near optimal solutions on deterministic problems. In a stochastic setting, centralized decision making proved to be more effective than distributed decision making.

Clearly, the optimal centralized policy will perform at least as well as any distributed decision making scheme. The results we present are encouraging in the sense that under deterministic demand arrivals we do not lose the solution quality by using spatial decomposition, and under stochastic demand arrivals the loss in the solution quality is limited to 3%. Considering its fast CPU times, its ability to yield integer solutions naturally and the fact that it accurately captures

the decision making structure in many real world systems, spatial decomposition might be preferred to temporal decomposition for large, practical applications.

References

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press, Princeton.

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

Bond, A. H. & Gasser, L., eds (1988), *Readings in Distributed Artificial Intelligence*, Morgan Kauffman, San Mateo, CA.

Corkill, D. D. (1982), A Framework for Organizational Self-Design in Distributed Problem Solving Networks, PhD thesis, University of Massachusetts, Amherst, MA.

Dayan, P. D. (1992), 'The convergence of TD(λ) for general λ', *Machine Learning* **8**, 341–362.

de Farias, D. P. & Van Roy, B. (2003), 'The linear programming approach to approximate dynamic programming', *Operations Research* **51**(6), 850–865.

Durfee, E. H., Lesser, V. R. & Corkill, D. D. (1987), Distributed artificial intelligence, *in* M. N. Huhns, ed., 'Cooperation Through Communication in a Distributed Problem Solving Network', Margan Kauffman, Los Altos, CA, chapter 2, pp. 29–58.

Gasser, L. & Huhns, M. N., eds (1989), *Distributed Artificial Intelligence*, Vol. 2, Pitman, London.

Godfrey, G. A. & Powell, W. B. (2001), 'An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems', *Management Science* **47**(8), 1101–1112.

Godfrey, G. A. & Powell, W. B. (2002), 'An adaptive, dynamic programming algorithm for stochastic resource allocation problems I: Single period travel times', *Transportation Science* **36**(1), 21–39.

Kurose, J. F. & Simha, R. (1989), 'A microeconomic approach to optimal resource allocation in distributed computer systems', *IEEE Transactions on Computers* **38**(5), 705–717.

Langley, P. (1995), *Elements of Machine Learning*, Morgan Kauffman, San Mateo, CA.

Mitchell, T. (1997), *Machine Learning*, McGraw-Hill, Auckland.

Moulin, B. & Chaib-Draa, B. (1996), An overview of distributed artificial intelligence, *in* G. M. P. O'Hare & N. R. Jennings, eds, 'Foundations of Distributed Artificial Intelligence', John Wiley and Sons, New York, pp. 3–56.

O'Hare, G. M. P. & Jennings, N. R., eds (1996), *Foundations of Distributed Artificial Intelligence*, John Wiley and Sons, New York.

Pope, R. P., Conry, S. E. & Mayer, R. A. (1992), Distributing the planning process in a dynamic environment, *in* 'Proceedings of the Eleventh International Workshop on Distributed Artificial Intelligence', Glen Arbor, Michigan, pp. 317–31.

Puterman, M. L. (1994), *Markov Decision Processes*, John Wiley and Sons, Inc., New York.

Schweitzer, P. & Seidmann, A. (1985), 'Generalized polynomial approximations in Markovian decision processes', *Journal of Mathematical Analysis and Applications* **110**, 568–582.

Stone, P. & Veloso, M. (2000), 'Multiagent systems: A survey from a machine learning perspective', *Autonomous Robots* **8**, 345–383.

Sutton, R. S. (1984), Temporal Credit Assignment in Reinforcement Learning, PhD thesis, University of Massachusetts, Amherts, MA.

Sutton, R. S. (1988), 'Learning to predict by the methods of temporal differences', *Machine Learning* **3**, 9–44.

Topaloglu, H. & Powell, W. B. (2002), Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems, Technical Report CL-00-02, Department of Operations Research and Financial Engineering, Princeton University.

Tsitsiklis, J. & Van Roy, B. (1997), 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control* **42**, 674–690.

Van Slyke, R. & Wets, R. (1969), 'L-shaped linear programs with applications to optimal control and stochastic programming', *SIAM Journal of Applied Mathematics* **17**(4), 638–663.

Waldspurger, C. A., Hogg, T., Huberman, B. A., Kephart, J. O. & Stornetta, W. S. (1992), 'Spawn: A distributed computational economy', *IEEE Transactions on Software Engineering* **18**(2), 103–117.

Watkins, C. J. C. H. (1989), Learning from Delayed Rewards, PhD thesis, Cambridge University, Cambridge, England.

Watkins, C. J. C. H. & Dayan, P. (1992), 'Q-Learning', *Machine Learning* **8**, 279–292.

A Proofs of Propositions

Proposition 1 Assume each $\hat{V}^k_{\underset{j,t+1,(it)}{\rightarrow}}(\cdot)$ is characterized by a series of points $\{\ u_j^{km} : m \in \mathcal{N}_j^k\ \}$ and slopes $\{\ v_j^{km} : m \in \mathcal{N}_j^k\ \}$ as in figure 4. We denote the flow over the interval $(u_j^{km}, u_j^{k,m+1})$ in the domain of $\hat{V}^k_{\underset{j,t+1,(it)}{\rightarrow}}(\cdot)$ by y_j^{km}. Then the approximate subproblem can be written as:

$$\tilde{V}_{(it)}(R_{(it)}, \omega_{(it)}) = \max \sum_{k\in\mathcal{K}} \sum_{d\in\mathcal{D}} c^k_{idt}\ x^k_{idt} + \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{I}:j>i} \hat{V}^k_{\underset{jt(it)}{\rightarrow}}(R^k_{\underset{jt(it)}{\rightarrow}}) + \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{I}} \sum_{m\in\mathcal{N}_j^k} v_j^{km}\ y_j^{km} \quad (33)$$

$$\text{s.t.} \qquad \sum_{d\in\mathcal{D}} x^k_{idt} = R^k_{it(it)} \qquad \forall k \in \mathcal{K} \quad (34)$$

$$R^k_{\underset{jt(it)}{\rightarrow}} = R^k_{jt(it)} \qquad \forall k \in \mathcal{K},\ \forall j > i \quad (35)$$

$$-\sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt} + R^k_{\underset{j,t+1,(it)}{\rightarrow}} = R^k_{j,t+1,(it)} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I} \quad (36)$$

$$\sum_{k\in\mathcal{K}} x^k_{idt} \le u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s \quad (37)$$

$$R^k_{\underset{j,t+1,(it)}{\rightarrow}} - \sum_{m\in\mathcal{N}_j^k} y_j^{km} = 0 \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I} \quad (38)$$

$$0 \le y_j^{km} \le u_j^{k,m+1} - u_j^{km} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I},\ \forall m \in \mathcal{N}_j^k \quad (39)$$

$$x^k_{idt} \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall d \in \mathcal{D}. \quad (40)$$

We first note that the variables $R^k_{\underset{jt(it)}{\rightarrow}}$ are set to constants by constraints (35), hence can be dropped from the problem by substituting their values in the objective function. We define new variables

η_d by $\eta_d = \sum_{k\in\mathcal{K}} x^k_{idt}$ for all $d \in \mathcal{D}^s$. Using constraints (38) we write $R^k_{\underset{\rightarrow}{j,t+1,(it)}} = \sum_{m\in\mathcal{N}^k_j} y^{km}$ and use this in (36) which eliminates the need for constraints (38). Finally we multiply constraints (36) by -1. Then we obtain the following equivalent optimization problem:

$$\tilde{V}_{(it)}(R_{(it)}, \omega_{(it)}) = \max \sum_{k\in\mathcal{K}} \sum_{d\in\mathcal{D}} c^k_{idt}\ x^k_{idt} + \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{I}} \sum_{m\in\mathcal{N}^k_j} v^{km}_j\ y^{km}_j \tag{41}$$

$$\text{s.t.} \quad \sum_{d\in\mathcal{D}} x^k_{idt} = R^k_{it(it)} \qquad \forall k \in \mathcal{K} \tag{42}$$

$$\sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt} - \sum_{m\in\mathcal{N}^k_j} y^{km}_j = -R^k_{j,t+1,(it)} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I} \tag{43}$$

$$\sum_{k\in\mathcal{K}} x^k_{idt} - \eta_d = 0 \qquad \forall d \in \mathcal{D}^s \tag{44}$$

$$\eta_d \le u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s \tag{45}$$

$$0 \le y^{km}_j \le u^{k,m+1}_j - u^{km}_j \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{I},\ \forall m \in \mathcal{N}^k_j \tag{46}$$

$$x^k_{idt} \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall d \in \mathcal{D}. \tag{47}$$

Next we use the identity:

$$\sum_{d\in\mathcal{D}} x^k_{idt} = \sum_{d\in\mathcal{D}} x^k_{idt} \sum_{j\in\mathcal{I}} \delta_{ijd} = \sum_{j\in\mathcal{I}} \sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt}$$

that must be satisfied for any feasible solution. This is true because if it is infeasible to apply decision d to a resource of type k in state i then $x^k_{idt} = 0$ will hold in any feasible solution, otherwise $\sum_{j\in\mathcal{I}} \delta_{ijd} = 1$. Using this equality and summing constraints (43) over $j \in \mathcal{I}$ we obtain:

$$\sum_{d\in\mathcal{D}} x^k_{idt} = \sum_{j\in\mathcal{I}} \sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt} = \sum_{j\in\mathcal{I}} \sum_{m\in\mathcal{N}^k_j} y^{km}_j - \sum_{j\in\mathcal{I}} R^k_{j,t+1,(it)}$$

We put this result in constraints (42) to obtain the final optimization problem:

$$\tilde{V}_{(it)}(R_{(it)}, \omega_{(it)}) = \max \sum_{k\in\mathcal{K}} \sum_{d\in\mathcal{D}} c^k_{idt}\ x^k_{idt} + \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{I}} \sum_{m\in\mathcal{N}^k_j} v^{km}_j\ y^{km}_j \tag{48}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{I}} \sum_{m\in\mathcal{N}^k_j} y^{km}_j = \sum_{j\in\mathcal{I}} R^k_{j,t+1,(it)} + R^k_{it(it)} \qquad \forall k \in \mathcal{K} \tag{49}$$

$$\sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt} - \sum_{m\in\mathcal{N}^k_j} y^{km}_j = -R^k_{j,t+1,(it)} \qquad \forall j \in \mathcal{I},\ \forall k \in \mathcal{K} \tag{50}$$

$$\sum_{k\in\mathcal{K}} x^k_{idt} - \eta_d = 0 \qquad \forall d \in \mathcal{D}^s \tag{51}$$

$$\eta_d \le u_{d(it)}(\omega_{(it)}) \qquad \forall d \in \mathcal{D}^s \tag{52}$$

$$0 \le y^{km}_j \le u^{k,m+1}_j - u^{km}_j \qquad \forall j \in \mathcal{I},\ \forall k \in \mathcal{K},\ \forall m \in \mathcal{N}^k_j \tag{53}$$

$$x^k_{idt} \in \mathbb{Z}_+ \qquad \forall k \in \mathcal{K},\ \forall d \in \mathcal{D}. \tag{54}$$

We consider a network composed of three sets of nodes $\mathcal{H} = \{k : k \in \mathcal{K}\}$, $\mathcal{M} = \{(j,k) : j \in \mathcal{I},\ k \in \mathcal{K}\}$, $\mathcal{J} = \{d : d \in \mathcal{D}^s\}$, and a root node. The supply of node $k \in \mathcal{H}$ is $\sum_{j\in\mathcal{I}} R^k_{j,t+1,(it)} + R^k_{it(it)}$, the supply of node $(j,k) \in \mathcal{M}$ is $-R^k_{j,t+1,(it)}$ and the supply of node $d \in \mathcal{J}$ is 0. The variables in the problem (48) represent the flows on arcs given by the following table:

Variable	From Node	To Node
$x^k_{idt},\ k \in \mathcal{K},\ d \in \mathcal{D}^s$	$(j,k) \in \mathcal{M} : \delta_{ijd} = 1$	$d \in \mathcal{J}$
$x^k_{idt},\ k \in \mathcal{K},\ d \in \mathcal{D}^m$	$(j,k) \in \mathcal{M} : \delta_{ijd} = 1$	root node
$\eta_d,\ d \in \mathcal{D}^s$	$d \in \mathcal{J}$	root node
$y^{km}_j,\ j \in \mathcal{I},\ k \in \mathcal{K},\ m \in \mathcal{N}^k_j$	$k \in \mathcal{H}$	$(j,k) \in \mathcal{M}$

Then constraints (49) are the flow balance constraints for node $k \in \mathcal{H}$, constraints (50) are the flow balance constraints of node $(j,k) \in \mathcal{M}$ and constraints (51) are the flow balance constraints for node $d \in \mathcal{J}$. □

Proposition 2 In order to show this proposition we need the following two lemmas:

Lemma 1 *Consider the following two equivalent linear programming problems with dual variables associated with the constraints indicated next to them:*

$$\begin{array}{llccccccc}
LP1 & \max & & & e^t y & + & f^t z & & \\
 & s.t. & Ax & + & By & & & = \ c & \sigma \\
 & & x & & & - & Dz & = \ 0 & \mu \\
 & & & & y & , & z & \geq \ 0, & \\
 & & & & & & & & \\
LP2 & \max & & & e^t y & + & f^t z & & \\
 & s.t. & & & By & + & ADz & = \ c & \sigma \\
 & & & & y & , & z & \geq \ 0. &
\end{array}$$

If (σ^*) is an optimal dual solution for LP2 then $(\sigma^*, -A^t\sigma^*)$ is an optimal dual solution for LP1.

Proof: The equivalence of the problems is obvious. We only need to show that the solution $(\sigma^*, -A^t\sigma^*)$ is dual feasible for LP1 and yields the same objective value. Since σ^* is dual feasible for the LP2, we have $B^t\sigma^* \geq e$, $D^tA^t\sigma^* \geq f$. Then $A^t\sigma^* - (A^t\sigma^*) = 0$, $B^t\sigma^* \geq e$ and $-D^t(-A^t\sigma^*) \geq f$ which establishes the dual feasibility of the solution $(\sigma^*, -A^t\sigma^*)$ for LP2. To complete the proof, we note that the dual objective values for both problems are equal. □

Lemma 2 *Consider the following linear programming problem with dual variables associated with the constraints indicated next to them:*

$$\begin{array}{llccccccc}
LP1 & \max & f^t x & + & g^t y & & & & \\
 & s.t. & Ax & & & = & b & \sigma \\
 & & Cx & + & Dy & = & e & \mu \\
 & & x & , & y & \geq & 0. &
\end{array}$$

Assume that the relationship $QC = A$ holds between matrices A and C for some Q. Then using the second set of constraints we can write $QCx = Ax = Qe - QDy$ and put this in the first constraint to obtain an equivalent linear programming problem:

$$\begin{array}{lccccccc} LP2 & \max & f^t x & + & g^t y & & & \\ & s.t. & & - & QDy & = & b - Qe & \sigma \\ & & Cx & + & Dy & = & e & \mu \\ & & x & , & y & \geq & 0. & \end{array}$$

If (σ^*, μ^*) is an optimal dual solution for the second problem then $(\sigma^*, \mu^* - Q^t\sigma^*)$ is an optimal dual solution for the first problem.

Proof: We show that the solution $(\sigma^*, \mu^* - Q^t\sigma^*)$ is dual feasible for the first problem and yields the same objective value. We first establish the dual feasibility of the solution $(\sigma^*, \mu^* - Q^t\sigma^*)$ for the first problem. Since (σ^*, μ^*) is dual feasible for the second problem, we have $C^t\mu^* \geq f$ and $-D^tQ^t\sigma^* + D^t\mu^* \geq g$. Then $C^t\mu^* = C^tQ^t\sigma^* + C^t(\mu^* - Q^t\sigma^*) = A^t\sigma^* + C^t(\mu^* - Q^t\sigma^*) \geq f$ and $D^t(-Q^t\sigma^* + \mu^*) \geq g$ which establishes the dual feasibility of the solution $(\sigma^*, \mu^* - Q^t\sigma^*)$ for the first problem. Finally we show the equality of the dual objective values by $b^t\sigma^* + e^t(\mu^* - Q^t\sigma^*) = (b^t - e^tQ^t)\sigma^* + e^t\mu^*$ which completes the proof. □

Proposition 2 The proof rests heavily on the proof of proposition 1. The transformation from problem (33) to (41), namely eliminating $R^k_{\underset{j,t+1,(it)}{\rightarrow}}$ in the constraints by writing it as $\sum_{m\in\mathcal{N}_j^k} y_j^{km}$, is of the form described in lemma 1, which does not change the optimal values for the dual variables of the remaining constraints. We then multiply constraints (36) by -1 to get constraints (43). Hence we need to show that the optimal dual variables of (42), (43) are $\{\pi^k_{it(it)} : k \in \mathcal{K}\}$ and $\{\pi^k_{j,t+1,(it)} - \pi^k_{it(it)} : k \in \mathcal{K},\ j \in \mathcal{I}\}$. By lemma 2, the optimal dual variables of the constraints (42) remain unchanged after the transition from problem (41) to (48) since they correspond to the first set of constraints in that lemma. Now we fix $k \in \mathcal{K}$; the transformation from (41) to (48) was made by noting:

$$\sum_{d\in\mathcal{D}} x^k_{idt} = \sum_{j\in\mathcal{I}}\sum_{d\in\mathcal{D}} \delta_{ijd}\ x^k_{idt}$$

and to make the transformation we added constraints (43) over all $j \in \mathcal{I}$. This is equivalent to multiplying the constraint set (43) for a fixed k by a $1 \times |\mathcal{I}|$ matrix of all 1's. Thus the matrix Q in lemma 2 is a $1 \times |\mathcal{I}|$ matrix of all 1's in this transformation. Then the optimal dual variables of constraints (43) for the fixed k are given by $\pi^k_{\cdot,t+1,(it)} - [1, 1, \ldots, 1]^t\pi^k_{it(it)}$ if we denote the vector $\left[\pi^k_{j,t+1,(it)}\right]_{j\in\mathcal{I}}$ by $\pi^k_{\cdot,t+1,(it)}$. □