Decomposing Inventory Routing Problems with Approximate Value Functions

Alejandro Toriello[1], George Nemhauser[2], Martin Savelsbergh[3]

[1]Daniel J. Epstein Department of Industrial and Systems Engineering

University of Southern California

3715 McClintock Avenue GER 240, Los Angeles, California 90089

toriello *at* usc *dot* edu

[2]H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology

765 Ferst Drive NW, Atlanta, Georgia 30332

george *dot* nemhauser *at* isye *dot* gatech *dot* edu

[3]CSIRO Mathematics, Informatics and Statistics

North Ryde, NSW 1670, Australia

martin *dot* savelsbergh *at* csiro *dot* au

September 3, 2010

Abstract

We present a time decomposition for inventory routing problems. The methodology is based on valuing inventory with a concave piecewise linear function and then combining solutions to single-period subproblems using dynamic programming techniques. Computational experiments show that the resulting value function accurately captures the inventory's value, and solving the multi-period problem as a sequence of single-period subproblems drastically decreases computational time without sacrificing solution quality.

1 Introduction

Fixed-charge network flow models are among the most difficult and well-studied in discrete optimization [25]. From a supply chain management perspective, fixed-charge networks isolate and abstract the interplay of fixed and variable costs inherent in many transportation models. *Lot-sizing* models are another fixture in optimization and operations research [12, 26]. By studying the relation between production and inventory holding costs, these models capture the long-term consequences of short-term decisions. Traditionally, the decisions in fixed-charge network flow and lot-sizing models are studied and implemented separately. More recently, the *inventory routing problem* (IRP) [5, 10, 23] has combined the two decision classes into a single, more holistic model.

This paper formulates and studies a generic IRP, and investigates a solution approach that incorporates an approximate value function into a *mixed-integer programming* (MIP) framework. By combining the techniques from MIP and *approximate dynamic programming* (ADP), we hope to circumvent difficulties typically encountered when solving multi-period models: The inventory value function allows us to shorten planning horizons, thus yielding a model size tractable to MIP methodology.

Within this context, the choice of a value function must balance two competing interests. The function must be simple enough to fit within a MIP framework, yet complex enough to capture the influence inventory has on the model. We have chosen to use a separable, piecewise linear concave value function. From a modeling perspective, these functions can be implemented in a MIP with only a small number of auxiliary continuous variables, and therefore do not alter the model's difficulty. From a practical perspective, the concavity captures the diminishing marginal value one expects in a model of this type, where, for instance, inventory at a consumer is more valuable the closer the consumer is to a stock-out. Finally, from a theoretical perspective, piecewise linear concave functions are related to pure IP value functions, which were shown in [4] to be formed from linear functions by repeated applications of three operations: non-negative linear combinations of two numbers, taking the minimum of two numbers, and taking the floor of a number. In this light, our approximate value functions mimic true value functions, but avoid the floor operation to maintain continuity and approximate the minimum operation with separable, concave slopes.

Approximate piecewise linear concave value functions have already been studied for several

resource allocation problems [27, Chapter 12]. In these applications, the single-period, or *transition* model, can usually be formulated as a linear program (LP). When a transition model is solved, the optimal dual solution is used as a proxy for a resource's value. After solving a transition, the dual solution is used to refine the approximate value function via an algorithm that preserves the function's concavity [32].

In our case, the single-period transition model is still a MIP, and the previous technique is no longer applicable, as dual multipliers are unavailable. Our approach instead updates the entire approximate value function via a combination of sampling and data fitting. A similar approach with classical least-squares linear fitting was used in [18] to value American options. Within each value function update, we randomly choose several starting inventory vectors, and solve several single-period transitions beginning from each sampled starting point. We record each inventory vector together with the resulting approximate value as measured by our approximate value function. After enough observations, we solve a constrained data fitting problem that yields the best-fit separable piecewise linear concave value function for our observations. We update our current value function by merging it with the best-fit function using a step-size rule. This process repeats until the resulting value function converges. To our knowledge, no other research has yet employed data fitting to compute value functions for MIP models in this manner.

Our main contribution is an algorithmic framework to compute a separable, piecewise linear approximate inventory value function for an IRP model. The value function allows for a decomposition approach in which solutions are generated one or a few periods at a time without sacrificing solution quality.

The rest of the paper is organized as follows. Section 2 reviews some relevant literature. Section 3 introduces our model. Section 4 details our value function and the algorithm used to compute it. Section 5 presents the results of a computational study and Section 6 gives conclusions and discusses future avenues of research.

2 Literature Review

One method used to solve many multi-period supply chain management models is to formulate them as mixed-integer programs (MIP). In this approach, a certain planning horizon is chosen, and

the model's parameters during this horizon are assumed to be known and deterministic. Typically, this approach is implemented within a rolling horizon framework: During each decision epoch, a model is formulated with the most current information, the model is solved to determine a decision, and then the parameters are updated to reflect the change in the system caused by the decision.

The length of the planning horizon implies a key tradeoff in this approach. A shorter horizon yields a model that is easier to solve, at the expense of a myopic solution, while a longer horizon yields more realistic solutions at the expense of more difficult, or even intractable models.

Because high-quality solutions typically require a longer horizon, many researchers have focused on techniques for solving these larger, more difficult models directly. One approach common in the integer programming community is column generation; see, for example, [7, 8, 9, 13, 21], and [11] for a detailed discussion. Many authors have in addition studied heuristic approaches to complex multi-period MIP models, e.g. [6, 14, 24, 28, 29].

Another approach, stemming from the Markov decision process and dynamic programming communities, involves decomposing the multi-period model into single-period subproblems linked by ending inventory levels. To counteract the myopic nature of solutions obtained from single-period solves, these models include value functions for the ending inventories. However, the complexity of large multi-period MIP's precludes the calculation of exact value functions, and approximations are necessary, e.g. [1, 2, 5, 16, 17, 22, 30, 31]. This approach has also been investigated for general resource allocation problems of various types; see [3, 27] and references therein for a thorough discussion.

3 An Inventory Routing Model

We consider an inventory routing model with multiple suppliers and consumers in which vehicles perform trips starting at a supplier and ending at a consumer, i.e., vehicles do not start and end at a depot. This situation occurs, for example, in maritime logistics (cf. [7, 28].)

A finite set of suppliers S and a finite set of consumers C are located far from each other, so that the distances between two elements of the same set are much smaller than distances between two elements of different sets. Homogeneous capacitated vehicles are available to begin a trip from any supplier, can visit as many suppliers as necessary, and then travel to visit as many consumers

as necessary, after which the trip ends (with no return.) At most one vehicle may begin a trip in any period, and a vehicle's trip begins and ends in the same period. This last assumption is not as restrictive as it sounds, because if the travel time between S and C is assumed constant (say τ periods), then any trip that begins on the supplier side in period t ends on the consumer side in period $t+\tau$, and therefore the inventories on the supplier side for period t may be identified with consumer inventories in period $t+\tau$.

Each supplier $i \in S$ has constant inventory bounds $[0, d_i]$ and a constant per period production rate $w_i > 0$. Similarly, each consumer $j \in C$ has constant inventory bounds $[0, d_j]$ and a constant per period consumption rate $w_j > 0$. Each supplier or consumer $i \in S \cup C$ has a starting inventory $s_i^0 \in [0, d_i]$. The fixed transportation costs between any two points $i, j \in S \cup C$ are $c_{ij} \geq 0$, and the per unit revenue obtained from delivering product from supplier $i \in S$ to consumer $j \in C$ is r_{ij}. Since vehicles are homogeneous, we normalize product measurements so that vehicles have unit capacity. There are no per-unit transportation and no inventory holding costs.

We model each period's transportation decision with a network $G = (N, A)$, where $N = \{\sigma, \sigma'\} \cup S \cup C$ and

$$A = \big(\{\sigma\} \times S\big) \cup S^2 \cup (S \times C) \cup C^2 \cup (C \times \{\sigma'\}).$$

Here, σ and σ' are artificial source and sink nodes. Define $A' = S^2 \cup (S \times C) \cup C^2 \subseteq A$ to be the set of *real* arcs. For period t and $a \in A$, let $x_a^t \in \{0, 1\}$ denote whether a vehicle travels on arc a. Let $y_t \in \{0, 1\}$ denote the aggregate indicator variable that is equal to one if and only if a trip occurs in period t. Let q_{ij}^t denote the amount of product from supplier $i \in S$ delivered to consumer $j \in C$ in period t, and let z_a^t denote the amount of product in the vehicle when it travels over arc $a \in A'$ in period t. (Note that the definition of z does not differentiate by product.) Let $s_i^t \in [0, d_i]$ denote the inventory at $i \in S \cup C$ at the end of period t. We use the notation $\delta^+(i) = \{a \in A : i \text{ is the tail of } a\}$ and $\delta^-(i) = \{a \in A : i \text{ is the head of } a\}$.

Let $\mathcal{T} = \{1, \ldots, T\}$ be the set of periods in the planning horizon. The IRP is modeled as:

$$\max \sum_{t \in \mathcal{T}} \Bigg(\sum_{i \in S, j \in C} r_{ij} q_{ij}^t - \sum_{a \in A'} c_a x_a^t \Bigg) \tag{1a}$$

$$\text{s.t. } y^t = \sum_{i \in S} x_{\sigma i}^t, \forall\ t \in \mathcal{T} \tag{1b}$$

$$\sum_{a\in\delta^+(i)} x_a^t - \sum_{a\in\delta^-(i)} x_a^t = 0, \forall\ i \in S \cup C, t \in \mathcal{T} \tag{1c}$$

$$s_i^t = s_i^{t-1} + w_i - \sum_{j\in C} q_{ij}^t, \forall\ i \in S, t \in \mathcal{T} \tag{1d}$$

$$s_j^t = s_j^{t-1} - w_j + \sum_{i\in S} q_{ij}^t, \forall\ j \in C, t \in \mathcal{T} \tag{1e}$$

$$z_a^t \leq x_a^t, \forall\ a \in A', t \in \mathcal{T} \tag{1f}$$

$$\sum_{a\in\delta^+(i)} z_a^t - \sum_{a\in\delta^-(i)} z_a^t = \sum_{j\in C} q_{ij}^t, \forall\ i \in S, t \in \mathcal{T} \tag{1g}$$

$$\sum_{a\in\delta^-(j)} z_a^t - \sum_{a\in\delta^+(j)} z_a^t = \sum_{i\in S} q_{ij}^t, \forall\ j \in C, t \in \mathcal{T} \tag{1h}$$

$$y^t \in \{0,1\}, \forall\ t \in \mathcal{T} \tag{1i}$$

$$x_a^t \in \{0,1\}, \forall\ a \in A, t \in \mathcal{T} \tag{1j}$$

$$q_{ij}^t \geq 0, \forall\ i \in S, j \in C, t \in \mathcal{T} \tag{1k}$$

$$z_a^t \geq 0, \forall\ a \in A', t \in \mathcal{T} \tag{1l}$$

$$s_i^t \in [0, d_i], \forall\ i \in S \cup C, t \in \mathcal{T}. \tag{1m}$$

The objective (1a) maximizes net profit over the planning horizon. The trip limit constraint (1b) ensures than no more than one vehicle starts a trip every period. The vehicle flow balance constraint (1c) requires that a vehicle exits a supplier or a consumer if it visits it. The supplier and consumer inventory balance constraints (1d) and (1e) track inventory levels from one period to the next. The variable upper bound constraint (1f) ensures that product flows on an arc only if a vehicle travels on it. The continuous product balance constraints (1g) and (1h) track the amount of product on a vehicle and relate it to the delivery variables. (Because all pickups occur before any delivery, we can use the q_{ij}^t variables in both constraint classes and do not need to differentiate flow by product.) Note that the variable domain constraints (1i)–(1m) along with the variable upper bounds (1f) and product balances (1g) and (1h) ensure that the total amount delivered and the total on the vehicle over any arc do not exceed the vehicle's unit capacity.

The formulation detailed above does not explicitly include subtour elimination constraints of

the form

$$\sum_{a \subseteq R^2} x_a^t \leq |R| - 1, \forall\ R \subseteq S \text{ or } R \subseteq C.$$

However, since $c \geq 0$, the product balance constraints (1g) and (1h) prevent any subtour from appearing in an optimal solution.

For T large enough, the model is infeasible if $\sum_{i \in S} w_i \neq \sum_{j \in C} w_j$. However, this requirement can be relaxed by adding a third-party supplier i_0 and a third-party consumer j_0, and modifying the model accordingly. One approach is to set $w_{i_0} = w_{j_0} = 0$, $d_{i_0} = d_{j_0} = \infty$, $h_{j_0}^0 = 0$ and $h_{i_0}^0 = M$, for some large enough $M > 0$. These parameters effectively mean that supply and consumption at the third-party points is unlimited, and delivery to and from these points can be treated as a slack in the model. Even with a third-party supplier and consumer, however, the model still requires $\sum_{i \in S} w_i \leq 1$ and $\sum_{j \in C} w_j \leq 1$; that is, the total supply per period and the total consumption per period must not exceed the vehicle's capacity.

As stated, the model includes no value for inventory at the end of the planning horizon. In practical terms, this implies that optimal solutions to (1) exhibit an *end effect*; inventories will be replenished periodically up until a given period, after which no more deliveries occur. This behavior is timed so that no inventory exceeds its bounds within the planning horizon, but creates a completely impractical solution with ending inventories close to their bounds. To work around this effect, models are usually solved on a rolling horizon basis, where the length T of the planning horizon far exceeds the number of periods for which a solution is needed. Once a solution to the model is obtained, only the decisions for the first few periods are actually implemented. Afterwards, the model parameters are updated to reflect the passage of time and the implemented decisions, and a new instance is formulated and solved.

The end effect can be eliminated by including a value for the ending inventory in the objective (1a), say $V(s^T)$, where $s^T = (s_i^T)_{i \in S \cup C}$. Moreover, if this value function accurately captures the different inventories' effect on the model, the planning horizon $\mathcal{T}$ can potentially be shortened to include only the periods for which decisions are immediately necessary. Because the difficulty and solve times of these MIP models tend to increase exponentially with T, shortening the planning horizon can have drastic effects on the time required to solve the model and on the quality of the resulting solutions. In the next section, we discuss how to incorporate a practical, accurate value

function into the model.

4 An Approximate Inventory Value Function

The inherent value of inventory is usually measured as the inventory's effect on future decisions, either in long-term average cost, or as a discounted future value defined recursively. In the discount model, the inventory value function $V : [0, d] \to \mathbb{R}$ satisfies a Bellman equation

$$V(s_S, s_C) = \max_{(x,q,z)\in X} \left\{ \sum_{i\in S, j\in C} r_{ij} q_{ij} - \sum_{a\in A'} c_a x_a + \gamma V(s_S + w_S - q_S, s_C - w_C + q_C) \right\}, \tag{2}$$

where $0 < \gamma < 1$ is a discount factor, X is the projection of the feasible set of (1) with $T = 1$ to the (x, q, z) variables, $s_S = (s_i)_{i\in S}$, and other aggregate vectors are defined analogously. (Time indices have been suppressed for simplicity.) Unfortunately, this value function cannot be computed exactly.

Given these difficulties, research in approximate dynamic programming (ADP) [27] has focused on approximating value functions in order to obtain practical value estimates. In general terms, an ADP algorithm seeks to estimate the true value V with an approximation $\tilde{V}$ that satisfies given structural requirements. To find a "good" estimate $\tilde{V}$, the state space $[0, d]$ of possible starting inventories is sampled repeatedly and the observed value is used to refine $\tilde{V}$.

For (1), a practical approximation $\tilde{V}$ of the value function must fit well within the MIP framework. One type of function that incorporates into a MIP model without difficulty is a separable piecewise linear concave function of the form

$$\tilde{V}(s) = \eta + \sum_{i\in S\cup C} v_i(s_i) \tag{3}$$

$$v_i(s_i) = \sum_{k=1}^{m-1} v_i^k (b_i^k - b_i^{k-1}) + v_i^m (s_i - b_i^{m-1}), \text{ for } s_i \in [b_i^{m-1}, b_i^m), \tag{4}$$

where $0 = b_i^0 < b_i^1 < \cdots < b_i^p = d_i$ is an a priori partition of the domain $[0, d_i]$, $\eta \in \mathbb{R}$ and $v_i^1 \geq v_i^2 \geq \cdots \geq v_i^p$. For simplicity, we assume that all inventory domains are partitioned into $p \in \mathbb{N}$ buckets, although this is not necessary.

It is well known in linear optimization that a separable piecewise linear concave function (3)

can be incorporated into the objective of a maximization LP or MIP. For (1), we define auxiliary continuous variables $s^T_{i,k}$ for $k = 1, \ldots, p$ and $i \in S \cup C$, and modify the model as follows:

$$\gamma\eta + \max \sum_{t \in \mathcal{T}} \Bigg(\sum_{i \in S, j \in C} r_{ij} q^t_{ij} - \sum_{a \in A'} c_a x^t_a \Bigg) + \gamma \sum_{i \in S \cup C} \sum_{k=1}^{p} v^k_i s^T_{i,k} \tag{5a}$$

$$\text{s.t. (1b)–(1m)} \tag{5b}$$

$$s^T_i = \sum_{k=1}^{p} s^T_{i,k}, \forall\ i \in S \cup C \tag{5c}$$

$$s^T_{i,k} \in [0, b^k_i - b^{k-1}_i], \forall\ k = 1, \ldots, p. \tag{5d}$$

Since $v^1_i \geq \cdots \geq v^p_i$, any optimal solution to the model will increase the auxiliary variables in order from $s^T_{i,1}$ to $s^T_{i,p}$, to achieve the largest possible contribution to the objective. For the remainder of the paper, we use $P_T(s, v, \eta)$ to denote an instance of (5), where T denotes the length of the planning horizon, $s^0 = s$, and (v, η) is used in (5a). We assume all other model parameters are fixed and v satisfies $v^1_i \geq \cdots \geq v^p_i$, for each $i \in S \cup C$. As an example, (1) with starting inventory $s^0 = s$ corresponds to $P_T(s, 0, 0)$.

4.1 An Approximate Dynamic Programming Algorithm

Given an instance of (1), our goal is to calculate a separable piecewise linear concave value function $\tilde{V}$ that approximately captures the value of inventory as defined in (2). Suppose our current "best guess" is $\tilde{V} = (v, \eta)$. We improve upon this incumbent by repeated applications of the following sequence of tasks: Sample a random inventory vector s and record its approximate value as measured by the optimal value of the one-period problem $P_1(s, v, \eta)$. Update s to the optimal ending inventory and repeat. Do this several times and from several different random starting inventory vectors s to obtain a collection of inventory vectors and associated approximate values. Find the separable, concave piecewise linear function that best fits this collection, and use it to update the incumbent $\tilde{V}$ via a step-size rule. Repeat the entire procedure to successively refine $\tilde{V}$.

Before formally stating the algorithm, we introduce the following terminology and notation:

- $\nu^*(P_T(s, v, \eta))$: Optimal objective value of $P_T(s, v, \eta)$.
- $s^*(P_T(s, v, \eta))$: Optimal ending inventory of $P_T(s, v, \eta)$.

- (v_0, η_0): Initial inventory value function. If none is available, set $(v_0, \eta_0) = (0, 0)$ or to another appropriate default satisfying $v_{i,0}^1 \geq \cdots \geq v_{i,0}^p, \forall\ i$.

- $f_{\max}$: Number of times we fit a value function to a set of inventory vectors and approximate values.

- $y_{\max}$: Number of random inventory samples (i.e. sample paths) per data set.

- $u_{\max}$: Number of solves starting from each sample (i.e. length of each sample path).

- $\alpha_f, f = 1, \ldots, f_{\max}$: Step size rule used to merge best-fit value function with incumbent value function. The step size we have used in our algorithm is given by McClain's rule [20, 27]:

$$\alpha_f = \begin{cases} 1, & f = 1 \\ \frac{\alpha_{f-1}}{1+\alpha_{f-1}-\beta}, & f \geq 2 \end{cases} \tag{6}$$

  where $0 < \beta \ll 1$ is a parameter. Initially, α_f behaves like a harmonic series, giving the first few iterations large weight. However, as f grows α_f converges to $\beta > 0$, and thus in the long term later iterations outweigh early ones.

The details are provided in Algorithm 1.

Algorithm 1 Approximating the Inventory Value Function

```
set (v, η) ← (v_0, η_0)
for f = 1, ..., f_max do
  for y = 1, ..., y_max do
    sample random inventory vector s from [0, d]
    set s_{f,y,1} ← s
    for u = 1, ..., u_max do
      solve P_1(s_{f,y,u}, v, η)
      set ν_{f,y,u} ← ν*(P_1(s_{f,y,u}, v, η)), s_{f,y,u+1} ← s*(P_1(s_{f,y,u}, v, η))
    end for
  end for
  solve fitting problem (8) with
```

$$(s_{f,y,u}, \nu_{f,y,u}), \quad u = 1, \ldots, u_{\max}, \quad y = 1, \ldots, y_{\max}$$

```
  to get best-fit value function (v̂, η̂)
  set v ← α_f v̂ + (1 − α_f)v, η ← α_f η̂ + (1 − α_f)η
end for
return (v, η)
```

The set of piecewise linear concave functions defined in (3) is convex. Thus, when we update the incumbent value function using the formula

$$\begin{aligned} v &\leftarrow \alpha_f \hat{v} + (1-\alpha_f) v \\ \eta &\leftarrow \alpha_f \hat{\eta} + (1-\alpha_f)\eta, \end{aligned} \tag{7}$$

where the first equation is evaluated component-wise, we preserve concavity as long as the initial value function (v_0, η_0) belongs to the set.

4.2 Separable, Piecewise Linear Concave Data Fitting

Assume we have a collection of inventory vectors $\{s^\ell\}_{\ell=1,\dots,n} \subseteq [0,d]$, and associated values $\nu_1, \dots, \nu_n \in \mathbb{R}$, where the index ℓ and number n indicate that these inventories and values may be any arbitrary collection. Given a partition $(b_i^k)_{i\in S\cup C}^{k=0,\dots,p}$ of the domain $[0,d]$, we solve the fitting problem

$$\begin{aligned} \min \ & \sum_{\ell=1}^{n} |\tilde{V}(s^\ell) - \nu_\ell| \\ \text{s.t. } & \tilde{V} \text{ satisfies (3)} \end{aligned} \tag{8}$$

to obtain the value function that best fits the collection. Here, $|\cdot|$ is any measure of error, such as the square or the absolute value.

To solve (8), for every s^ℓ and every $i \in S \cup C$ we determine the unique bucket $m \in \{1,\dots,p\}$ that satisfies $s_i^\ell \in [b_i^{m-1}, b_i^m)$, and define an auxiliary vector

$$\tilde{s}_{i,k}^\ell = \begin{cases} b_i^k - b_i^{k-1}, & k = 1,\dots,m-1 \\ s_i^\ell - b_i^{k-1}, & k = m \\ 0, & k = m+1,\dots,p \end{cases} \tag{9}$$

that decomposes s_i^ℓ into its constituent buckets from the partition. Then (8) becomes

$$\min \sum_{\ell=1}^{n} \left| \eta + \sum_{i \in S \cup C} \sum_{k=1}^{p} v_i^k \tilde{s}_{i,k}^\ell - \nu_\ell \right| \tag{10a}$$

$$\text{s.t. } v_i^1 \geq v_i^2 \geq \cdots \geq v_i^p, \forall\ i \in S \cup C \tag{10b}$$

$$\eta \in \mathbb{R}. \tag{10c}$$

Without (10b), this problem is a classical unconstrained linear data fitting problem, and has a closed-form solution, for instance, when $|\cdot|$ is the squared error. However, even with the additional constraints, the problem is a simple convex optimization problem that can be solved with readily available optimization software.

5 Computational Results

In this section, we present the results of a series of experiments designed to test the accuracy of Algorithm 1's approximate inventory value function and its effectiveness in generating good solutions to (1).

The instances used in the computational study were randomly generated based on problem characteristics found in real-world situations. For example, to generate the transportation costs $c_a, a \in A'$, the location of each supplier and each consumer was randomly generated from the uniform distribution over a unit square, and consumers' horizontal coordinate was displaced by eight units. The transportation costs were then proportional to the Euclidean distances between the locations. Similarly, production and consumption rates $w_i, i \in S \cup C$ were randomly generated so that the expected total production and the expected total consumption per period were both approximately 85% of vehicle capacity. However, a third-party supplier and a third-party consumer were added to the model in case the two total rates were not equal. The number of suppliers and consumers in each instance was six and five respectively (plus third parties), a number that is consistent with some real situations. Instances 1 through 4 have positive revenue values r_{ij}. Experiment 3 requires instances including only transportation costs c_a in the objective, so instances 5 through 7 have $r_{ij} = 0, \forall\ i \in S, j \in C$.

We generated value functions (3) for each of our instances using a straightforward implemen-

tation of Algorithm 1, as outlined in Section 4.1. Since the number of problems $P_1(s, v, \eta)$ to solve can be large, in the early iterations we solved relaxations where some integer variables were allowed to be fractional. However, for every instance at least the last $\frac{f_{\max}}{2}$ outer loops of Algorithm 1 solved the $P_1(s, v, \eta)$ problems without any variable relaxation. The total computation time to obtain an instance's value function was usually about 12 to 16 hours. This time consuming calculation occurs only once, but the resulting value function can be used to solve instances with different starting inventory vectors. In the remainder of the section, we use (v^*, η^*) to denote the value function we computed using Algorithm 1. All computations were performed on a Xeon 2.66 GHz workstation with 8 Gb of RAM. CPLEX 11 was used as the optimization solver. The following parameter values were used in Algorithm 1 to generate value functions for the instances:

- $\gamma = 0.9999$.
- $p = 3$.
- $b_i^k = \frac{kd_i}{p}, k = 0, \ldots, p$.
- $|\cdot| = (\cdot)^2$.
- $f_{\max} = 500$.
- $y_{\max} = 120$.
- $u_{\max} = 10$.
- $\beta = 0.15$.

The experiments evaluate (v^*, η^*) in several ways. The first experiment uses the value function to construct a solution for a 60-period model by solving a sequence of 60 single-period models linked by inventory levels, and compares this solution to a solution obtained by solving the entire 60-period model at once.

The next experiment performs a similar test but concentrates only on the first period's decision. Given a ten-period instance, we construct a solution by solving the first period as a stand-alone single-period model and then the remaining nine periods together, and compare this solution to a solution of the whole ten-period model, to determine whether the first-period decision we get by using the value function is myopic or sub-optimal.

In many applications, practitioners solve models with planning horizons that are much longer than needed, implementing only the first few periods of the resulting solution. In our final experiment we test the value function (v^*, η^*) against this practical approach. Suppose that the model must determine the next five periods' decisions; we compare the solution of a five-period model with value function to the solution obtained by solving a longer 60-period model (without value function) and then truncating to the first five periods.

5.1 Experiment 1: Direct Comparison

In the first experiment, we constructed five instances of $P_{60}(s, v^*, \eta^*)$ and randomly generated five starting inventories s, for a total of 25 problem instances. We used two methods to generate solutions:

- 1P: Construct a solution one period at a time by solving the subproblem $P_1(s, v^*, \eta^*)$ to optimality and using $s^*(P_1(s, v^*, \eta^*))$ as the next starting inventory. Refer to Section 4.1 for a precise definition of these terms.
- 60P: Give the entire $P_{60}(s, v^*, \eta^*)$ instance to CPLEX, emphasizing feasibility, with a one-hour time limit.

Table 5.1 presents the results of this experiment, with all times reported in seconds. Column Inst indexes the instance-starting inventory pair. Columns 1P Obj and 60P Obj respectively give the objective value of each solution. The Bound column has the best dual bound found by CPLEX within the time limit. Columns 1P Time and 60P Time show the computation time used to obtain the best solution via both methods.

We observe that the single-period heuristic generates competitive solutions, beating the 60P solution in all but three instances: 1-3, 3-3 and 5-1. However, the more notable result is the drastic time difference. The single-period heuristic is able to generate a solution in a few seconds (the average is 27.8,) while CPLEX does not find its best 60P solution until much later, after an average of 2,529 seconds. This drastic time difference suggests that one can generate a good solution quickly using the single-period heuristic.

The results of the first experiment suggested an additional experiment. Instead of constructing a solution by solving single-period instances, the same method could be applied with two-period

	Obj				Time (sec.)		
Inst	1P	2P	60P	Bound	1P	2P	60P
1-1	-810.19	-804.50	-827.59	-753.51	11	226	1963
1-2	-817.10	-823.54	-845.76	-757.60	14	345	1040
1-3	-892.84	-874.21	-847.39	-755.11	66	21362	3284
1-4	-813.53	-803.40	-828.75	-748.21	17	308	3246
1-5	-809.03	-807.93	-838.18	-754.80	13	261	2902
2-1	-745.49	-729.36	-750.48	-676.77	10	105	804
2-2	-715.36	-715.33	-745.97	-677.38	7	73	3253
2-3	-711.62	-708.11	-737.63	-663.68	9	141	1568
2-4	-708.93	-710.70	-733.13	-668.69	7	83	2953
2-5	-726.17	-721.23	-753.60	-676.24	8	58	1610
3-1	2794.40	2795.62	2771.41	2873.24	42	3060	235
3-2	2812.60	2818.29	2786.35	2902.84	52	2517	3309
3-3	2578.34	2667.05	2669.78	2841.47	193	48455	3147
3-4	2795.49	2798.76	2759.93	2875.42	43	1671	3313
3-5	2805.63	2808.43	2771.71	2880.39	38	1830	531
4-1	3342.36	3348.26	3300.65	3421.72	21	405	3424
4-2	3353.05	3363.90	3318.68	3424.61	19	245	3380
4-3	3287.23	3296.66	3266.03	3371.86	19	459	3266
4-4	3290.45	3296.85	3271.44	3363.51	22	324	3260
4-5	3335.52	3342.43	3305.05	3410.97	20	393	2092
5-1	-820.62	-812.62	-820.42	-741.29	13	188	3406
5-2	-803.15	-796.33	-834.64	-739.56	12	141	3347
5-3	-786.86	-782.59	-799.52	-721.43	16	216	2827
5-4	-797.96	-787.52	-819.95	-730.08	12	164	1895
5-5	-812.43	-812.41	-836.13	-738.71	11	173	3178

Table 5.1: Direct Comparison.

instances linked by inventory levels:

- 2P: Construct a solution by solving the subproblem $P_2(s, v^*, \eta^*)$ to optimality and using $s^*(P_2(s, v^*, \eta^*))$ as the next starting inventory.

Table 5.1 shows the results of the two-period heuristic for the same instances, using the same naming convention outlined earlier. Except for instances 1-2 and 2-4, the two-period heuristic solution has a better objective than the single-period heuristic solution and is only worse than the 60P solution in instances 1-3 and 3-3. The objective improvement, however, comes at the expense of a drastic increase in computation time that seems to depend on the particular instance. For example, the average time for all models derived from instances 2, 4 and 5 increases only by a factor of 15, from 14 seconds to 211 seconds. On the other hand, the two-period heuristic solution for instances 1-3 and 3-3 took many hours to complete, and indicates that the time required to solve even small problems of only a few periods depends heavily on the starting inventory vector s.

5.2 Experiment 2: Consequence of Using the Value Function

In our second experiment, we wanted to test if the solutions generated by the value function are myopic. Using the same parameters from Experiment 1, we constructed five $P_{10}(s, v^*, \eta^*)$ instances and used the five random starting inventories for a total of 25 problem instances. We generated solutions with two methods:

- 1P + 9P: Solve $P_1(s, v^*, \eta^*)$ to optimality with CPLEX and record the optimal solution. Solve $P_9(s^*(P_1(s, v^*, \eta^*), v^*, \eta^*)$ with CPLEX, emphasizing optimality with a 24-hour limit; concatenate the two solutions.
- 10P: Solve the entire $P_{10}(s, v^*, \eta^*)$ instance with CPLEX, emphasizing optimality with a 24-hour limit.

In words, method (1P + 9P) uses the value function to generate a one-period solution and then solves the remaining nine-period model (starting from the one-period model's optimal ending inventory). We compare this to solving the ten-period model with method 10P.

Table 5.2 gives the results of Experiment 2. As before, the naming convention for column INST pairs instances with starting inventories. The next two columns detail each solution's objective; 10P denotes the objective for the solution obtained by solving the ten-period model, while (1P + 9P) denotes the objective of the solution obtained by solving the first period with a single-period model and then solving the remaining nine-period model. The BOUND column has the best bound for the ten-period model (with value function) obtained by CPLEX as part of its optimization to generate the 10P solution.

The results in Table 5.2 indicate that the solutions generated by the two methods are very similar. The (1P + 9P) solution's objective is on average only 0.55% less than the 10P objective, and is greater in four instances: 1-1, 3-2, 4-3 and 5-5. Moreover, the largest relative gap between objectives is only 3.74%, occurring on instance 3-4, and every other (1P + 9P) objective is within 1.5% of the 10P objective. Results are similar when we compare objectives to the CPLEX dual bound; the average relative gap for the (1P + 9P) solution is 3.01%, while the same average for the 10P solution is 2.47%. Both measures indicate that implementing the 1P solutions results in an objective only about half a percentage point below what could be obtained by solving the entire ten-period model.

Inst	Obj (w/ val) 1P + 9P	10P	Bound
1-1	-133.44	-134.55	-130.6
1-2	-138.17	-136.89	-134.29
1-3	-136.32	-135.3	-125.02
1-4	-128.16	-127.89	-124.75
1-5	-135.53	-135.29	-132.04
2-1	-118.18	-117.77	-113.97
2-2	-118.82	-118.61	-116.19
2-3	-108.16	-107.62	-104.04
2-4	-109.49	-109.08	-107.23
2-5	-117.85	-116.16	-114.84
3-1	462.63	462.68	467.07
3-2	480.46	480.2	487.15
3-3	380.83	385.82	393.72
3-4	446.18	463.5	467.14
3-5	465.69	467.41	472.05
4-1	590.29	590.55	598.82
4-2	595.63	597.39	601.12
4-3	532.89	531.75	546.46
4-4	532.22	534.67	540.14
4-5	577.41	580.57	587.33
5-1	-126.35	-125.89	-120.84
5-2	-125.24	-124.06	-120.74
5-3	-113.82	-112.83	-105.39
5-4	-116.22	-115.07	-111.44
5-5	-123.4	-123.5	-119.37

Table 5.2: Value function influence on future decisions.

5.3 Experiment 3: Solution Comparison

The preceding experiments do not compare an actual solution as it would be generated in practice. Recall that the typical approach to solving multi-period models involves solving an instance with a long planning horizon, e.g. $T = 60$, but then *implementing* only the decisions of the first few periods of the resulting solution. This process is then repeated inside of a rolling horizon framework. A more practical comparison therefore involves comparing the first few periods of solutions generated by our method against the traditional approach.

In our third experiment, we created three instances of $P_5(s, v^*, \eta^*)$ and five starting inventories s, for a total of 15 problem instances. We generated solutions via two methods:

- 5P: Solve instance $P_5(s, v^*, \eta^*)$ with CPLEX, emphasizing feasibility and with an eight-hour time limit.
- Trad: Solve instance $P_{60}(s, 0, 0)$ (i.e. no ending value function) with CPLEX, emphasizing feasibility with a 24-hour time limit. Truncate to the first five periods of the problem's

solution.

However, comparing the two solutions is not a simple matter. One can consider the objective function value of each (without value function), but this does not take ending inventories into account, and is thus an incomplete comparison. On the other hand, if we use our value function (v^*, η^*) to measure the ending inventories' value, we are weighing the results in our favor (an optimal solution to $P_5(s, v^*, \eta^*)$ would always be the best solution according to this second measure.) We therefore decided on a compromise measure that attempts to capture the inherent value of inventory without using (v^*, η^*): If an instance has costs but not revenue in the objective (1a), then it is always better to have *less* inventory at the suppliers and *more* inventory at the consumers, because if a supplier has more inventory, its product must be picked up sooner (which entails a cost), and similarly, if a consumer has less inventory, its product must be dropped off sooner (which again entails a cost).

Table 5.3 details the results of the third experiment, using base instances with no revenue. The naming convention for column Inst is the same as for Table 5.1. Columns 5P Obj and Trad Obj respectively show the objective value of each method's solution (without value function). Columns 5P Time and Trad Time show the time needed to find each solution, in minutes. The four columns under S Inv and C Inv show each solution's total ending inventory for suppliers and consumers, respectively.

	Obj		Time (min.)		S Inv		C Inv	
Inst	5P	Trad	5P	Trad	5P	Trad	5P	Trad
5-1	-75.6	-77.07	425	1394	1.72	2.91	2.34	2.09
5-2	-75.49	-77.39	9	1344	2.59	2.78	2.78	2.67
5-3	-63.29	-64.57	434	1417	3.68	3.68	3.56	2.57
5-4	-60.73	-78.66	0.5	1407	2.85	2.09	2.30	2.89
5-5	-76.15	-80.28	100	1242	1.93	2.66	2.54	2.54
6-1	-81.47	-84.02	95	1317	1.60	2.93	2.34	2.29
6-2	-82.55	-81.54	17	1213	2.59	3.47	2.78	2.50
6-3	-67.05	-69.38	4	1437	3.68	3.84	3.56	3.4
6-4	-66.25	-67.34	29	1365	2.68	3.67	2.3	2.18
6-5	-82.94	-83.5	0.1	1375	1.74	2.74	2.54	2.54
7-1	-81.49	-83.06	10	1432	1.71	3.19	2.39	2.38
7-2	-83.74	-83.78	14	1124	2.55	2.84	2.83	2.68
7-3	-67.96	-54.69	435	1315	3.65	4.63	3.6	2.48
7-4	-83.87	-68.25	3	1405	1.63	2.81	3.35	2.34
7-5	-83.31	-83.4	123	1428	1.82	2.43	2.59	2.58

Table 5.3: Solution comparison.

We observe that the 5P solution method clearly outperforms the traditional approach. The

average objective values are very similar (-75.46 for the 5P solution and -75.8 for the traditional solution,) with the 5P solution beating the traditional solution in all but three instances, 6-2, 7-3, and 7-4. However, the average ending inventories show a marked difference. The average total ending supplier inventory is 2.43 for 5P versus 3.11 for the traditional solution, and the analogous statistic for consumer inventory is 2.79 versus 2.54. So on average, the 5P solutions end with 22% less inventory at suppliers and 9.6% more inventory at the consumers, even though they spend less to achieve this. The last statement can be underlined by one additional fact: For all instances except 5-4, 6-2, 7-3 and 7-4, the 5P solution dominates the traditional solution, implying that the 5P solutions do more with less in most instances, and not just on the average.

As in Experiment 1, the drastic difference in computation time is noteworthy. The 5P solutions are generated in an average of 113 minutes, as opposed to the traditional solutions, which take an average of 22.5 hours, close to the 24-hour limit. In fact, only for instances 5-1, 5-3 and 7-3 do the 5P times approach the eight-hour limit; in the rest of the cases the solution is found relatively quickly, at an average of only 31 minutes. As before, the shorter computation times suggest that the value function can be used to generate a good solution quickly.

6 Conclusions

We have introduced an innovative time decomposition of IRP models that combines techniques from mixed-integer programming and approximate dynamic programming. Our results indicate that the value function obtained from our algorithm closely captures the true value of inventory for the model (1). Our experiments also show that the value function can be used to generate high-quality solutions far more efficiently than traditional methods.

Our research gives rise to many interesting new questions. On the integer programming side, the ability to effectively solve instances with a large number of periods by solving a sequence of instances with a small number of periods suggests focusing on techniques for solving the instances with a small number of periods to optimality, for example using problem-specific cutting planes or extended formulations.

On the dynamic programming side, we can ask whether more complex value functions can better capture inventory's true value. Specifically, a separable function does not address situations

in which two or more inventories are close to bounds and costlier decisions must be made. A non-separable value function implies a fitting problem of the kind introduced in [19]; see also [15]. Another interesting question is whether our approach can be adapted to include the dynamic generation and modification of function breakpoints, in a manner similar to the techniques outlined in [27].

Finally, from a practical perspective the next natural step is to adapt the value function approach to more realistic models. This additional realism could come in the form of stochastic and/or non-constant parameters, time-expanded networks, and many other complications that real-world models possess. Each of these additions implies integer programming or dynamic programming challenges.

Acknowledgements

A. Toriello gratefully acknowledges support from the National Science Foundation, the Roberto C. Goizueta Foundation, and the ARCS Foundation. The work of G. Nemhauser and M. Savelsbergh was supported by AFOSR grant FA9550-07-1-0177 and NSF grant CMMI-0758234. The authors would like to thank the Associate Editor and two anonymous referees for helpful suggestions on earlier versions of the article.

References

[1] D. Adelman. Price-Directed Replenishment of Subsets: Methodology and its Application to Inventory Routing. *Manufacturing and Service Operations Management*, 5:348–371, 2003.

[2] D. Adelman. A Price-Directed Approach to Stochastic Inventory/Routing. *Operations Research*, 52:499–514, 2004.

[3] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[4] C.E. Blair and R.G. Jeroslow. The value function of an integer program. *Mathematical Programming*, 23:237–273, 1982.

[5] A. Campbell, L. Clarke, A. Kleywegt, and M. Savelsbergh. The Inventory Routing Problem. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 4, pages 95–114. Springer, 1998.

[6] A.M. Campbell and M.W.P. Savelsbergh. A Decomposition Approach for the Inventory-Routing Problem. *Transportation Science*, 38:488–502, 2004.

[7] M. Christiansen. Decomposition of a Combined Inventory and Time Constrained Ship Routing Problem. *Transportation Science*, 33:3–16, 1999.

[8] M. Christiansen and B. Nygreen. A method for solving ship routing problems with inventory constraints. *Annals of Operations Research*, 81:357–378, 1998.

[9] M. Christiansen and B. Nygreen. Modelling path flows for a combined ship routing and inventory management problem. *Annals of Operations Research*, 82:391–412, 1998.

[10] J.-F. Cordeau, G. Laporte, M.W.P. Savelsbergh, and D. Vigo. Vehicle Routing. In C. Barnhart and G. Laporte, editors, *Handbook in Operations Research and Management Science: Transportation, Volume 14*, chapter 6, pages 367–428. Elsevier, 2007.

[11] G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors. *Column Generation*. Springer, 2005.

[12] A. Drexl and A. Kimms. Lot sizing and scheduling – Survey and extensions. *European Journal of Operational Research*, 99:221–235, 1997.

[13] F.G. Engineer. *Shortest path based column generation for integer programming*. PhD thesis, Georgia Institute of Technology, 2009.

[14] M. Hewitt, G.L. Nemhauser, and M.W.P. Savelsbergh. Combining Exact and Heuristic Approaches for the Capacitated Fixed Charge Network Flow Problem. To appear in *INFORMS Journal on Computing*, 2009.

[15] D. Klabjan and D. Adelman. An Infinite-Dimensional Linear Programming Algorithm for Deterministic Semi-Markov Decision Processes on Borel Spaces. *Mathematics of Operations Research*, 32:528–550, 2007.

[16] A. Kleywegt, V. Nori, and M.W.P. Savelsbergh. The Stochastic Inventory Routing Problem with Direct Deliveries. *Transportation Science*, 36:94–118, 2002.

[17] A. Kleywegt, V. Nori, and M.W.P. Savelsbergh. Dynamic Programming Approximations for a Stochastic Inventory Routing Problem. *Transportation Science*, 38:42–70, 2004.

[18] F.A. Longstaff and E.S. Schwartz. Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies*, 14:113–147, 2001.

[19] A. Magnani and S. Boyd. Convex Piecewise-Linear Fitting. *Optimization and Engineering*, 10:1–17, 2009.

[20] J.O. McClain. Dynamics of exponential smoothing with trend and seasonal terms. *Management Science*, 20:1300–1304, 1974.

[21] S. Michel and F. Vanderbeck. A Column Generation based Tactical Planning Method for Inventory Routing. Technical Report 00169311, INRIA Bordeaux Sud-Ouest, team RealOpt, 2008. Available on-line at http://hal.inria.fr/docs/00/33/90/37/PDF/techRepR2.pdf.

[22] A. Minkoff. A Markov decision model and decomposition heuristic for dynamic vehicle dispatching. *Operations Research*, 41:77–90, 1993.

[23] N.H. Moin and S. Salhi. Inventory routing problems: a logistical overview. *Journal of the Operational Research Society*, 58:1185–1194, 2007.

[24] N. Nananukul. *Lot-Sizing and Inventory Routing for a Production-Distribution Supply Chain.* PhD thesis, The University of Texas at Austin, 2008.

[25] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc., 1999.

[26] Y. Pochet and L. Wolsey. *Production Planning by Mixed Integer Programming.* Springer, 2006.

[27] W.B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality.* John Wiley & Sons, Inc., 2007.

[28] M.W.P. Savelsbergh and J.-H. Song. Inventory Routing with Continuous Moves. *Computers and Operations Research*, 34:1744–1763, 2007.

[29] M.W.P. Savelsbergh and J.-H. Song. An Optimization Algorithm for the Inventory Routing Problem with Continuous Moves. *Computers and Operations Research*, 35:2266–2282, 2008.

[30] L. Schenk and D. Klabjan. Intramarket optimization for express package carriers. *Transportation Science*, 42:530–545, 2008.

[31] L. Schenk and D. Klabjan. Intra Market Optimization for Express Package Carriers with Station to Station Travel and Proportional Sorting. To appear in *Computers and Operations Research*, 2010.

[32] H. Topaloglu and W. Powell. An Algorithm for Approximating Piecewise Linear Concave Functions from Sample Gradients. *Operations Research Letters*, 31:66–76, 2003.