Inventory allocation and transportation scheduling for logistics of network-centric military operations

F. Barahona
P. Chowdhary
M. Ettl
P. Huang
T. Kimbrel
L. Ladanyi
Y. M. Lee
B. Schieber
K. Sourirajan
M. I. Sviridenko
G. M. Swirszcz

This paper describes a prototype inventory-placement and transportation-scheduling solution developed in support of the emerging military doctrine of Network-Centric Operations (NCO). NCO refers to an unprecedented ability to share information among cooperating forces, enabled by modern communications and computing technology. The objective of the Network-Centric concept is to collect, disseminate, and react to real-time information in order to improve the performance of the U.S. Army as a fighting force. One problem that arises in the logistics domain involves the maintenance of combat vehicles. We seek to determine the improvement, if any, made possible by exploiting accurate information on the status of available repair parts inventory, the current locations of mobile supply points, and the demand for parts. We describe logistics algorithms for maximizing the operational availability of combat vehicles by producing flexible, optimized inventory and delivery plans that decrease replenishment times and prioritize parts allocations and repairs. Our algorithms are designed to leverage real-time information available from modern communications and inventory tracking technology by employing state-of-the-art mathematical optimization models. Our simulations indicate that Network-Centric Logistics (NCL) can significantly improve combat vehicle availability in comparison with current practice.

1. Introduction

Military logistics systems face a dynamic and uncertain environment. The United States and its allies are confronted by increasing numbers of opportunistic adversaries and insurgencies that use unconventional fighting tactics to nullify an overwhelming force advantage. The response must be agile, adaptive, and flexible in both military operations and logistics. The current logistics system works well in an environment of relatively predictable demand, such as peacetime garrison operations or traditional, highly planned force-on-force operations. However, conventional logistics systems often break down in modern military operations that involve rapid force-structure change, extremely mobile forces, and greatly varying demands.

In order to achieve greater flexibility, modern logistics models require new analytical tools and execution models with greater adaptability and agility. In 2005, the Defense Advanced Research Projects Agency (DARPA) sponsored a Network-Centric Logistics (NCL) experiment to demonstrate the effectiveness of dynamic configuration algorithms for tactical ground logistics control. The objectives were to increase the flexibility of tactical supply chains and to improve delivery speed by treating tactical logistics as a dynamic configuration problem and by controlling physical inventory and distribution with proven techniques from adaptive inventory-management systems.

In this paper, we demonstrate how a dynamic multi-point supply approach can increase operational availability in a volatile combat environment compared with a traditional hierarchical logistics structure. (We

U.S. Department of Defense Distribution Statement: Approved for Public Release, Distribution Unlimited.

0018-8646/07/$5.00 © 2007 IBM

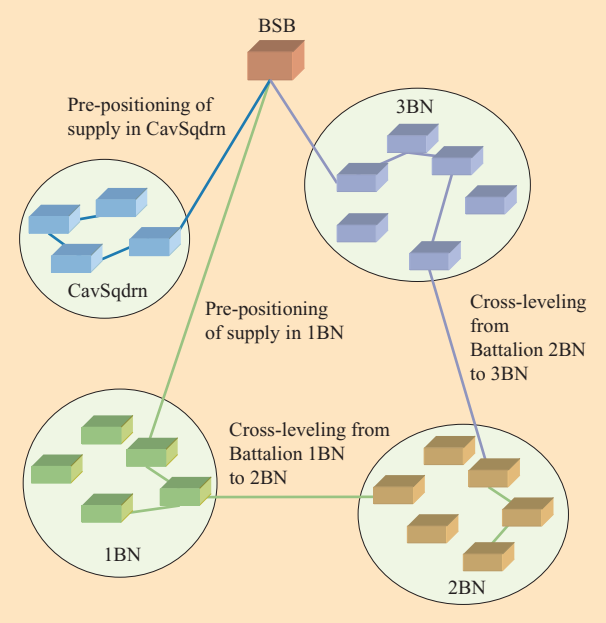

Figure 1

Testing environment based on a U.S. Army logistics scenario. Lines denote examples of possible movements of parts between units. (BSB: brigade support battalion; BN: battalion; CavSqdrn: cavalry squadron.)

define the term *operational availability* more rigorously in the section on measures of performance and in other sections that follow.) The logistics algorithms designed to achieve our goal are based on state-of-the-art mathematical optimization models. To evaluate the concept, we have conducted a series of experiments in which the logistics models are driven by a high-speed logistics simulation platform, a topic that is beyond the scope of the current paper. The simulator takes an operations plan (OPLAN) and generates detailed battlefield scenario data. The demonstration is driven by a fictitious scenario lasting 30 days. The optimization models produce a plan for the storage and delivery of repair parts for maintenance support in a single combat brigade.

The remainder of the paper is organized as follows. Section 2 describes logistics challenges encountered in military operations of today and introduces NCL concepts. Section 3 describes the inventory allocation problem and the mathematical algorithms we use to solve it. Section 4 introduces the transportation scheduling problem, and the next four sections describe our solution. Section 9 contains numerical results, and Section 10 concludes the paper.

2. The logistics challenge

The setting for our problem is based on a projected Army logistics scenario. We focus on the distribution of parts inventories needed to repair combat vehicles in a combat brigade. For the purpose of this study, only critical repair parts (i.e., parts necessary to restore vehicles to operational status) are considered. As shown in **Figure 1**, the brigade consists of several combat battalions denoted 1BN, 2BN, and 3BN. [For our purposes, the cavalry squadron (CavSqdrn) is also treated like a battalion.] These are supplied by a central logistics depot called the Brigade Support Battalion, or BSB. A battalion is composed of a group of smaller operational units, most of which are called companies; we simply refer to all of these units as companies. These units require logistics support to meet their demand for spare parts. This support is provided by combined delivery and repair-team trucks; however, some parts are crew-replaceable and do not require the delivering truck to stay while its mechanics carry out the repair. As indicated in the figure, parts may be "cross-leveled" between companies or battalions as defined in the next paragraph.

The brigade is network-enabled in the sense that it can share information through modern communication technologies. The network provides continuous visibility of all repair parts inventory at the BSB, the companies, and even in individual combat vehicles. Each company can be considered to have a local stocking point (in actuality representing on-board spare parts storage on the combat vehicles themselves). The locations of the BSB and the operational units change over time. NCL responds to supply needs by continually fulfilling demand as requested, allowing any truck to serve any battalion. Widely distributed parts require intelligence to identify where a needed supply part should come from and who should supply it. It is possible to transfer materiel between companies and across battalions. This is called *cross-leveling*, and it may entail obtaining parts from different, rapidly changing locations. In this system, repair parts can be supplied from multiple sources, including the BSB, pre-loaded stores on delivery and repair-team trucks, and even other combat vehicles if they carry on-board spare parts.

Although network-enabled logistics structures and operating procedures do not currently exist, they are under development as part of the future combat system of the Army. Traditional brigade combat teams of today use hierarchical distribution techniques in which repair parts are located in the BSB, parts requests are consolidated by the subordinate units, and a single daily replenishment operation, called a *logistics package*, delivers parts to the subordinate units. Each battalion is serviced only by trucks dedicated to it. **Table 1** lists the main features of logistics operations, contrasting the traditional

Table 1 Logistics operations for the baseline case and NCL.

	Baseline	*NCL*
Locations of supplies	Centralized at BSB	Partially distributed—some part types are carried on board
Supply network	Top-down hierarchy	Flexible: any truck can serve any battalion
Method of distribution	Daily batches using only per-battalion dedicated trucks	Flexible-size batches using any truck in brigade
Decision cycle	24 hours	As often as one hour
Decision criteria	First-come first-served	Mathematical optimization used to maximize vehicle availability

hierarchical logistics structure (i.e., the baseline) and NCL.

Measures of performance

The efficacy of the NCL approach was evaluated using standard military utility testing and evaluation methodologies. For this project, DARPA was interested in increased operational availability of combat vehicles and reduced customer wait time.

Operational availability (A_o) is a measure of the time during which the capabilities of a system are available for operational use. It takes into account failure and repair information. This dependent variable is measured at the vehicle level and summed over vehicles. Simple, unweighted values of A_o equal the time a vehicle is working divided by total time. The precise definition we use is described later. It includes time-varying relative priorities of combat units corresponding to their operations; for instance, a unit engaged in battle is considered to have high priority.

Customer wait time (*CWT*) is a measure of time from request to delivery. In our case, the request time is defined to be the time at which a part breaks. CWT comprises the time for administrative processing, possibly the time waiting for a part to arrive from outside the brigade, the time waiting for transportation to begin, and the travel time or total elapsed time spent in order to get the part from the supplier's location to the consumer's location.

Solution overview

Our solution consists of two main components, an inventory allocation module and a transportation scheduler, as illustrated in **Figure 2**. The inventory allocation module takes as input the available repair parts inventory, forecasted breakages, and the future positions of the operational units. It produces an allocation plan for each repair part and operational unit in the brigade over a future planning horizon of up to 72 hours. It accounts for storage capacity constraints as well as relative priorities of the combat units.

The transportation scheduling module uses as input the current locations of the (immobilized) broken vehicles, the projected future movements of other entities, and the allocation plan generated by the inventory allocation module. It produces a plan for delivering parts and carrying out repairs. This schedule specifies the spare parts to carry on each vehicle and the locations at which to load and unload the parts. The primary objective is to deliver the spare parts needed to repair currently broken vehicles. The secondary objective is to replenish the parts inventory at the logistics points in order to approach the levels specified by the inventory allocation module. This allows repairs to begin immediately upon breakage (if the part is available in the same unit) or very soon thereafter (if the part is available in the same battalion) in the case of crew-replaceable parts that are carried on board.

The framework allows real-time adjustment of schedules and resource allocation. The analytical modules continually update the solutions in response to changing conditions. At each time increment, which we call an *iteration*, of a scenario simulation, the optimizers receive the current state from the simulator and compute an optimized delivery plan covering the next 72 hours. We call an individual run with its own settings of input parameters an *experiment*.

The simulator commits and executes the initial portion of the plan. As the simulated scenario evolves and operational plans change, vehicles break down, and new repair parts become available, the simulator communicates these changes to the optimizers, which produce a new delivery plan optimized for the new state.

3. Inventory allocation

The goal of the inventory allocation module is to determine pre-positioned levels of supplies to best respond to future breakages. We develop a practical heuristic allocation model that uses up-to-date information on available inventory, supply points, and demand forecast. The model recognizes cross-leveling opportunities, accounts for movements of operational

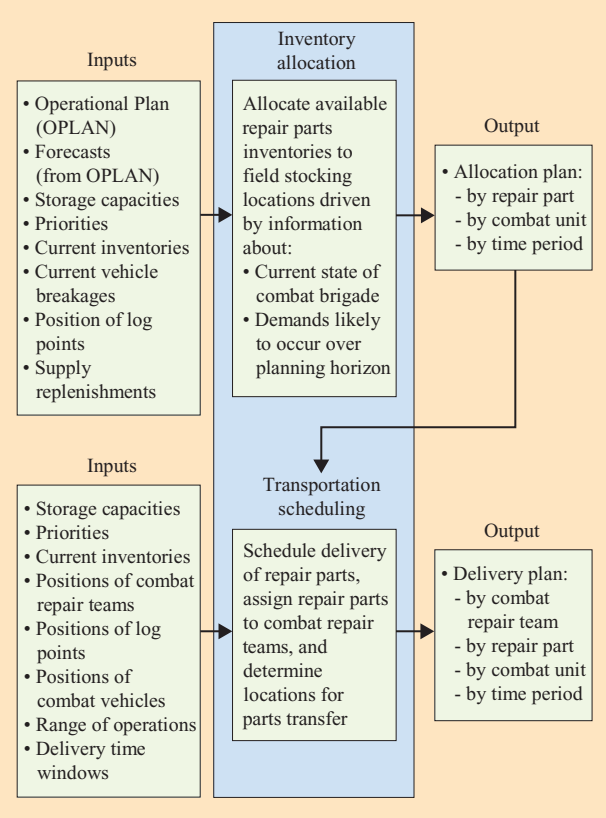

Figure 2

Functional design of the NCL logistics optimization prototype.

units within the brigade, and deals with short-term supply shortfalls such as delayed deliveries or supply shortages that could lead to longer-term degradation of capability. The inventory allocation is performed in two stages; the first stage strives to achieve high operational availability, and the second stage attempts to further reduce customer wait times through strategic placement of inventory. The allocation procedure is based on assigning a preference score computed from weighted fulfillment ratios and transportation lead times to each company to identify stocking points that can efficiently cross-fill demand at neighboring companies. Allocating inventory on the basis of the preference score minimizes customer wait time, which in turn maximizes fleet availability. Although the proposed inventory allocation solution is a practical heuristic that is not in general optimal, we show in Section 9 that a key performance utility commonly used by the U.S. Army, operational availability, can be improved by roughly ten percent when the solution is applied under a realistic military force scenario.

The academic literature presents a large body of research on inventory-service tradeoff models. However, none of the traditional approaches deal with dynamic supply chains or multiple sourcing in the context of optimization. We therefore propose a heuristic algorithm that leverages earlier work on service-level optimization in commercial supply chains, implosion techniques for manufacturing, and inventory allocation in service-after-sales networks [1–6]. In contrast to conventional inventory systems and methods, the proposed algorithm can deal with mobile entities that may change locations during a planning horizon. Furthermore, it does not rely upon static sourcing relationships to allocate inventory, but instead manages the allocation of inventory to each entity dynamically by exploiting opportunities for multiple sourcing, cross-leveling, and selecting suppliers on the fly. A recent paper [7] deals with a related inventory-allocation problem in a hierarchical repairable service parts system with two levels. The system in question consists of a central repair facility, a central warehouse, and a number of field stocking locations that service customers. The authors describe a repair and inventory allocation model that determines the number of parts to ship from the central warehouse to the field stocking locations in order to minimize the total expected inventory holding and back-order costs over a planning horizon. Although their approach could be employed to address the inventory allocation problem in a traditional brigade with hierarchical distribution operations, it does not capture the dynamic sourcing relationships found in an NCL environment with cross-leveling of materiel between combat units.

Before presenting our modeling and solution approach for making inventory allocation decisions in a network-enabled brigade, we summarize several key modeling assumptions: a) theater-level logistics are not modeled except for repair parts during operations; this excludes other categories of materiel such as armament, fuels, ammunitions, and communications subsystems; b) at most one part failure can occur on any given combat vehicle, so that each repair involves a single part type needed to return a broken vehicle to an operational condition; c) installation times are deterministic and depend on the type of repair part delivered to a combat vehicle; and d) the gathering of installed parts from broken vehicles in order to repair other broken vehicles (i.e., cannibalization) is not currently considered.

We now define the notation required to describe our model. Here and elsewhere, t always refers to the time t units from the current (simulated) time; that is, each time we run our algorithms, we "reset the clock" so that the current time is 0.

Logistics network

P : Set of repair part types indexed by p.

B : Set of battalions indexed by b.

C : Set of companies indexed by c.
w_{ct} : Relative importance of company c at time t.
w_{bt} : Relative importance of battalion b at time t.
L_{ijt} : Expected transit time between companies i and j for transport initiated in time period t.
Q_{pc} : Storage capacity of part p at company c.
Q_{pb} : Overall storage capacity of part p at battalion b where $Q_{pb} \leq \sum_{c \in b} Q_{pc}$; $c \in b$ denotes the set of all companies c assigned to battalion b.

Note that the maximum storage capacity of a battalion, Q_{pb}, is at most the sum of the on-board spare capacities of all of its combat vehicles. Further restrictions, such as stocking limits, may also apply.

Demand and supply

D_{pct} : Expected demand for part p at company c at time t (D_{pc0} equals the demand backlog at time 0).
U_{pt} : Quantity of part p expected to arrive from outside the brigade at time t.
W_{p0} : In-transit inventory of part p at time 0 (repair parts stored on trucks).
I_{pc0} : On-hand inventory of part p at company c at time 0 (current on-hand inventory).
A_{pt} : Quantity of part p available in the brigade at time t (supply available for allocation).

The demand for repair parts is assumed to be deterministic. However, the demand intensity varies on the basis of information about the mission type and location provided in the OPLAN, thus accountng for different scenario factors that have an impact on the vehicle damage and repair parts requirements (e.g., low-demand intensity exists during humanitarian assistance missions; high-demand intensity exists during combat operations).

Decision variables

X_{pct} : Number of parts of type p allocated to company c at time t.
X_{pbt} : Number of parts of type p allocated to battalion b at time t.
R^*_{pct} : Optimized cumulative receipts of part type p at company c at time t.
R^*_{pbt} : Optimized cumulative receipts of part type p at battalion b at time t.

The relative importance w_{ct} or w_{bt} is a non-negative weight representing the priority of a unit relative to other units. A larger weight means higher priority. The transit times are computed by a shortest-path algorithm that takes into account the state of the road network and the current movement plan. Expected demands are computed from scenario data, including per-part breakage rates as functions of combat vehicle activity (e.g., idle or engaged in combat). Initial quantities and parts arriving from outside the brigade are part of our scenario data, along with brigade structure, storage capacities, etc. The proposed algorithm proceeds in three steps, as described in the following sections.

Step I: Allocate repair parts on the basis of relative priorities

In this step, we attempt to maximize the fraction of breakages that can be serviced immediately from pre-positioned repair parts inventory, taking into account the priorities of the various military units. We allocate parts hierarchically, first to battalions and then to companies, using a prorating scheme.

The first goal when allocating spare parts to a battalion is to cover the breakages forecasted for all of its companies. We do this in a way that takes into account the relative priority of each company: The fraction of spare parts allocated to a battalion is the priority-weighted sum of the breakage forecasts of its companies divided by the priority-weighted sum of the breakage forecasts across all companies in all battalions. However, the number of parts allocated will not exceed the storage limit Q_{pb} at the battalion. The following formula expresses this rule:

$$X_{pbt} := \min \left\{ Q_{pb}, \frac{\sum_{c \in B} w_{ct} D_{pct}}{\sum_{c \in B} w_{ct} D_{pct}} A_{pt} \right\}, \quad (1)$$

where

$$A_{pt} = \left[A_{p0} + W_{p0} + \sum_{c \in B} I_{pc0} + \sum_{u=0}^{t} U_{pu} - \sum_{u=0}^{t-1} \sum_{c \in B} D_{pcu} \right]$$

is the expected amount of available (unallocated) supply of a part type p in the brigade at time t. A_{pt} includes all of the available supply in the brigade of part p in time period t, not just the inventory at the BSB, and is computed as the difference between the cumulative number of parts that entered the brigade until time t and the cumulative number of expected breakages until time $(t-1)$. Having thus allocated parts to a battalion b, we further allocate them to each company c in the battalion using a similar prorating scheme,

$$X_{pct} := \min \left\{ Q_{pc}, \frac{w_{ct} D_{pct}}{\sum_{c \in b} w_{ct} D_{pct}} X_{pbt} \right\}. \quad (2)$$

Denote $q_{pct} = (X_{pct} / w_{ct} D_{pct})$ as the weighted *fulfillment ratio* for an operational unit c. To improve operational availability (as defined in Section 2), we try to stock

parts as much as possible at the demand point to satisfy demand immediately. This can be achieved by having a higher fulfillment ratio. The allocation scheme (1) and (2) strives to balance the fulfillment ratios of all battalions and companies while satisfying the stocking limits Q_{pb} and Q_{pc}. Notice that the allocation in this step may entail fractional quantities of inventory being assigned to a company or battalion. Finally, we set $A_{pt} := A_{pt} - \sum_{c \in B} X_{pct}$.

Step II: Iteratively improve allocation to further reduce CWT

If the storage capacity limit at some companies or battalions is smaller than the desired allocation target based on their weighted share as given in Equations (1) and (2), leftover supply exists at the end of Step I. This remaining supply is allocated among companies or battalions that have not exceeded their capacity limit in order to minimize the expected time to respond to breakages that are not fixed immediately. The algorithm gives preference to operational units that are centrally located in order to maximize the benefit from cross-leveling. This helps to minimize the CWT as defined in Section 2.

For each time period t, the algorithm computes a preference score M for each company c in the brigade, ranks all companies by their preference scores, and allocates one part from the remaining unallocated available supply to the company with the highest score. This step is repeated until all remaining stock is allocated, or all companies reach their capacity limits. In cases in which companies reach their capacity limits, all remaining parts are allocated to the BSB.

Several scoring metrics (M) exist that we can use in the first step. The simplest one is based on the expected transit time L_{cjt} between any two companies c and j in the brigade. We define M_{ct} as the expected transit time to other companies,

$$M_{ct} = \sum_{j \in C} L_{cjt} . \tag{3}$$

A low M_{ct} indicates that a company is centrally located and can therefore respond quickly to requests from other companies in the brigades. The simple preference score M_{ct} considers transit times, but it does not consider the inventory state. Thus, we enhance the score by defining another preference score $\tilde{M}_{pct}$ that simultaneously considers transit times and inventory state:

$$\tilde{M}_{pct} := \sum_{j \in C} \max \{0, (q_{pct} - q_{pjt}) L_{cjt}\} . \tag{4}$$

$\tilde{M}_{pct}$ combines the expected transit time and the difference in weighted fulfillment ratios, helping to balance the fulfillment ratios across the brigade. We use $\tilde{M}_{pct}$ to allocate the remaining available supply using the algorithm below. The company with a smallest $\tilde{M}_{pct}$ score is given the highest priority for allocation. Notice that the following algorithm can assign fractional values in Step A2 as the remaining (expected) supply, A_{pt}, need not be an integer.

Algorithm A: Allocate remaining supply

- *Step A1:* Find $c^* := \text{argmin}_{c \in C: X_{pct} < Q_{pct}} \{\tilde{M}_{pct}\}$.
- *Step A2:* Set $X_{pc^*t} := X_{pc^*t} + \min \{1, A_{pt}\}$ and $A_{pt} = A_{pt} - \min \{1, A_{pt}\}$.
- *Step A3:* If $A_{pt} = 0$, stop. Otherwise, update $\tilde{M}_{pct}$ on the basis of the allocation in Step A2 and return to Step A1.

Given that the values of expected demand are usually fractional and the amount of expected supply available is small, allocating one unit at a time incrementally can be justified.

Step III: Determine allocation plan

Using the allocations obtained in Steps I and II, we generate a parts allocation plan for the BSB and all operational units in the brigade. The plan is expressed in the form of cumulative net receipts targets R^*_{pct} (cumulative net flow of parts into a company) for each repair part p and operational unit c and time period t. We have

$$R^*_{pct} := X_{pct} + \sum_{u=0}^{t-1} D_{pcu} - I_{pc0} . \tag{5}$$

The allocation plan is always feasible in the sense that the total number of parts delivered to an operational unit is always equal to the number of parts that are taken from other units (including the BSB). For example, if company c is scheduled to receive ten parts at time t, there are other companies that, in sum, provide these ten parts. These are represented as negative receipts in the receipts plan. Again, it is possible that the values of R^*_{pct} are fractional. To make R^*_{pct} an integer, we set $X_{pct} := X_{pct} - (R^*_{pct} - \lfloor R^*_{pct} \rfloor)$ and $A_{pt} := A_{pt} + (R^*_{pct} - \lfloor R^*_{pct} \rfloor)$. (Here the bracket symbols represent the floor function and supply the nearest integer less than or equal to the value within the brackets.) We then use Algorithm A to allocate A_{pt} to the companies and recalculate R^*_{pct} using Equation (5). Note that at most one of the R^*_{pct} values can be fractional after we apply Algorithm B[1] in any period t for every part p; this is unavoidable because the expected demand can be fractional.

The operational decision as to where to source the supply is provided by the transportation scheduler, which is described in the following section.

[1]We refer to the aforementioned steps of setting values for X_{pct} and A_{pt}, along with allocating A_{pt}, as Algorithm B.

4. Transportation scheduler

We now describe our solution for generating an optimized plan for the trucks that load and deliver spare parts. The primary goal is to determine a schedule for the trucks to load parts at the supply locations (BSB and companies) and deliver them to the combat vehicles that must be repaired. The schedules of the trucks are also given load and unload operations for parts that are not required immediately, so that the inventory level at each location approaches the target specified by the inventory allocation module.

Inputs

One of the inputs to the transportation scheduler is the output of the inventory optimization phase: namely, a list of parts that must be delivered to each location and the times at which to deliver them to achieve optimized preplacement of inventories.

Other inputs describe the number of available trucks and their operational constraints, such as their ranges of operation (e.g., the distances that they can travel before they must return to the BSB for refueling) and their capacities for storing parts. Our model for delivery can incorporate a variety of other operational constraints, such as load and delivery time windows, preferred or prohibited assignments of trucks to routes, and constraints on the total length and composition of a route (such as limits on the number and type of deliveries in each route). However, of these constraints, only the truck capacity and delivery time window constraints were applied in this prototype effort.

Outputs

For each available truck we either produce a route or label the truck as idle. A route consists of a list of locations to visit (BSB, company, or broken combat vehicle) in sequential time order, and the sequence of operations to perform at each location (load part, unload part, or repair broken vehicle). A route may include companies and broken vehicles that belong to different battalions, in contrast to the traditional hierarchical assignment of trucks. A route starts at the current location of the truck. It ends at the BSB, a company, or a broken vehicle. The duration of a route does not exceed the planning horizon of 72 hours.

Objective function

We now formally describe the objective function used by the optimizer as specified by DARPA. The function contains two terms. The first measures the availability of combat vehicles throughout the planning horizon. The second term penalizes the schedule (reduces its score) if the inventories at the various locations do not meet or exceed the levels recommended by the inventory allocator.

A good schedule prioritizes the repairs in order to maximize the number of operational combat vehicles across battalions and time, taking into account the relative weights w_{bt} of each battalion b at each time period t, and striving to balance the availabilities across battalions. To achieve this goal, we include in the objective function a term $f_{bt}(r_{bt})$ that measures the operational availability of vehicles at battalion b and time t as a result of the repairs, where r_{bt} denotes the number of vehicles in battalion b that have been repaired before or during period t. We wish to maximize the sum of these availabilities, weighted over all battalions and time periods; that is,

$$\sum_{b,t} w_{bt} f_{bt}(r_{bt}). \tag{6}$$

This is the first term in our objective function, and it is divided by a normalizing factor that is described later.

Three ranges of operational availability are defined for the combat vehicles: up to 80%, from 80% to 90%, and from 90% to 100%. Each one has a distinct priority; for instance, if we ignore weighting of battalions due to different activities such as "in combat" versus "idle," a balanced solution with two battalions each at 80% availability is considered by our customer, DARPA, to be better than one battalion at 70% and one at 90%. (The weights specified in the OPLAN for different battalions are included in the overall objective function as defined below.) The terms $f_{bt}(r_{bt})$ in the objective function capture these three levels of priority as follows. Let V_b be the total number of combat vehicles in battalion b, and let N_b be the number of operational vehicles at time period 0. Thus, the fraction of operational vehicles at period t in the battalion (ignoring future breakages) is given by $(r_{bt} + N_b)/V_b$. The primary priority in repairing is to increase this fraction to 80%. The secondary priority is to increase this fraction to 90%, and the tertiary priority is to be fully operational, i.e., to bring the fraction up to 100%. In the objective function we give these levels of priorities weights that reflect their relative importance: 4, 2, and 1, respectively. With these weights, each term $f_{bt}(r_{bt})$ is a piecewise-linear concave function written as

$$f_{bt}(r_{bt}) = f\left(\frac{r_{bt} + N_b}{V_b}\right),$$

where $f(x) = 2\min(x, 0.8) + \min(x, 0.9) + x$. Note that the first 80% of operational vehicles are counted four times in this expression, the next 10% twice, and the last 10% once, as desired. Because $(r_{bt} + N_b)/V_b$ is always between 0 and 1, the term $f_{bt}(r_{bt})$ yields a value between 0 and 3.5, as can be seen by substituting $x = 0$ and $x = 1$, respectively.

Recall that R^*_{pct} denotes the optimized cumulative receipts for part p at company c as generated by the inventory allocator, and that the initial inventory is I_{pc0}. Denote by R_{pct} the actual cumulative receipts by time t for part p at company c. It is defined as the number of parts of type p unloaded from trucks up to and including time t at company c, minus the number loaded at company c and taken elsewhere. To penalize deviations from the target receipts value R^*_{pct}, we apply the following penalty term Z_{pct} to the objective function:

$$Z_{pct} = \begin{cases} \max\left\{0, \dfrac{R^*_{pct} - R_{pct}}{R^*_{pct} + I_{pc0}}\right\} & \text{if } R^*_{pct} + I_{pc0} > 0; \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Z_{pct} measures the relative deviation of the actual receipts plan from the target receipts plan established by the inventory allocator. Rewriting

$$\frac{R^*_{pct} - R_{pct}}{R^*_{pct} + I_{pc0}}$$

as

$$1 - \frac{R_{pct} + I_{pc0}}{R^*_{pct} + I_{pc0}},$$

we can interpret it as the fraction of inventory missing at time t compared with the value suggested by the inventory allocator. Note that R_{pct} and R^*_{pct} may be negative. However, $R_{pct} + I_{pc0} \geq 0$ and $R^*_{pct} + I_{pc0} \geq 0$, and thus the above penalty is less than 1 for all p, c, and t. Taking the maximum of the fraction and zero ensures that there is no benefit (negative penalty) from exceeding the inventory requirements.

The composite function we seek to maximize, including the weighted measure of availability and the terms corresponding to pre-placement, is

$$\frac{1}{3.5\sum_{b,t} w_{bt}} \sum_{b,t} w_{bt} f_{bt}(r_{bt}) - \frac{\alpha}{|P| \cdot |C| \cdot T} \sum_{c,p,t} Z_{pct}, \quad (8)$$

where the normalizing factors 3.5 $\Sigma_{b,t}$ w_{bt} and $|P| \cdot |C| \cdot T$ convert each of the two sums in Equation (8) to numbers between 0 and 1. Recall that P is the set of part types and C is the set of companies. T denotes the planning horizon, and α is a weight between 0 and 1 that denotes the relative importance of meeting the inventory pre-placement targets. In this prototype effort, we used a value of 0.05. This value could be further tuned in a larger-scale development effort. (We note that this particular combination of including per-battalion weights for availability but per-company weights for inventory placement is not arbitrary; it was specified by our customer, DARPA.)

Outline of solution approach

We have developed and experimented with three approaches for producing optimized schedules. The first approach uses *integer programming* (IP, defined below), and is described in Section 5. The second approach uses local search and is described in Section 6. The third approach is a somewhat sophisticated greedy heuristic, described in Section 7.

In the integer programming approach, we first produce routes for the trucks in order to deliver parts to combat vehicles to be repaired. Then we add load and unload operations to these routes to deliver the spare parts specified by the inventory allocator. The local search and greedy approaches work to satisfy both goals concurrently. In Section 8 we describe how the three algorithms are combined.

5. Integer programming algorithm

Our first algorithm directly optimizes the first term of the objective function, using integer linear programming techniques. That is, it first seeks only to fix broken combat vehicles and ignores inventory placement. It then heuristically adds load and unload operations to the schedule thus generated so far, attempting to satisfy the recommended inventory placement produced by the inventory optimizer.

To meet the primary objective (repairing vehicles), we solve an integer programming mathematical model. This model requires *column generation*, a technique to generate possible routes for each truck. We use a specialized technique we call *fix and resolve* to solve the integer program and determine optimized routes for the trucks. We then use these routes to transport the rest of the inventory to the companies in addition to carrying out as many repairs as possible.

Figure 3 is an overview illustrating the relationships among the components of our algorithm, which are described in detail in the following sections. The figure shows the dataflows into and out of each component, including the input from the inventory allocation module described previously, and the final output, a set of routes and associated operations on those routes.

Integer programming mathematical model

An integer programming mathematical problem is similar to a linear programming problem, with the additional restriction that some of the variables must take only integer values. A *linear programming* problem is a mathematical problem in which we seek to assign values to numerical variables so that they satisfy a collection of linear equalities and inequalities (the constraints) and also maximize a linear function (the objective function). Linear and integer programming models have been in wide use since World War II in both military and

industrial operations research, to solve a variety of problems in vehicle and personnel scheduling, facility location, investment portfolio selection, distribution network planning, and other practical applications.

Variables

For each route j to which a truck can be assigned, we define a variable x_j. This is a binary variable: A value of 1 indicates that the route j is to be used in the schedule (a truck travels the route); 0 means that the route is not used. Thus, we define

$$x_j \in \{0, 1\} \text{ for all routes } j. \tag{9}$$

The number of possible routes may be very large. If we were to include all routes, we would have created an unmanageable model with far too many variables x_j. (Later, we provide a detailed discussion of our heuristic column-generation techniques used to generate just the routes and variables we need.)

We define the variable r_{bt} as the number of vehicles in battalion b that will be fixed before or during period t. This must be a non-negative number, i.e.,

$$r_{bt} \geq 0 \text{ for each battalion } b \text{ and each period } t. \tag{10}$$

This variable must also take only integer values. However, we do not need to enforce this constraint directly, and we reap a computational benefit as a consequence: By virtue of constraint (14) in the next section, if each x_j is an integer, each r_{bt} will be an integer as well.

Constraints

Each broken combat vehicle should be visited at most once during each schedule. This ensures that the objective value does not "take credit" for fixing any vehicle more than once. To express this, let $S(i)$ be the set of routes that include a stop at a broken vehicle i. Then,

$$\sum_{j \in S(i)} x_j \leq 1 \text{ for each broken combat vehicle } i. \tag{11}$$

Each available truck is either assigned to exactly one route or sits idle. To express this, let $T(v)$ be the set of routes to which truck v can be assigned. Then,

$$\sum_{j \in T(v)} x_j \leq 1 \text{ for each repair truck } v. \tag{12}$$

Two (or more) trucks can be candidates for the same route or for a portion of the same route. To keep track of such possible assignments, we label each candidate pair of route and truck, and thus the pair is unique.

The quantity of a part type transported away from a location on trucks visiting the location should not exceed the quantity that has arrived at the location. To model this, let $P(p, t, l)$ be the set of routes on which the assigned trucks transport away part p from location l up to and including time period t. Let d_{pjl} be the quantity of part p that the truck on route j transports away from location l. Let n_{ptl} be the quantity of part p that has arrived at location l up to and including period t. Then the constraint is expressed as

$$\sum_{j \in P(p,t,l)} d_{pjl} x_j \leq n_{ptl}$$
$$\text{for each part } p, \text{ each period } t, \text{ and each location } l. \tag{13}$$

The number of combat vehicles serviced by the trucks on the routes equals the number of combat vehicles repaired. Let $U(b, t)$ be the set of routes for trucks that visit a broken vehicle in battalion b before or during period t. Let a_{bjt} be the number of vehicles in battalion b that the truck on route j services up to and including period t. This is a non-negative integer constant. Thus,

$$\sum_{j \in U(b,t)} a_{bjt} x_j = r_{bt} \text{ for each battalion } b \text{ and each period } t. \tag{14}$$

In order to capture the function $f_{bt}(r_{bt})$ in the objective function, we introduce three variables y_1, y_2, and y_3. Variable y_1 measures the extent to which 80% of the vehicles are operational; y_2 measures the next 10%, and y_3 measures the final 10%. Their sum must be the actual operating fraction, and we include a constraint to capture that. Thus, each term $f_{b,t}(r_{b,t})$ in the objective function is

$$f_{bt}(r_{bt}) = 4y_1 + 2y_2 + y_3, \tag{15}$$

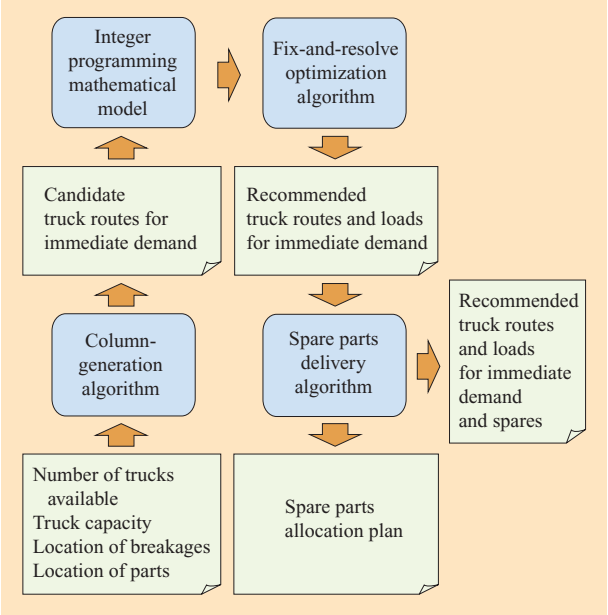

Figure 3

Components of the integer programming algorithm.

and the additional variables and constraints needed to represent it are defined by

$$y_1 + y_2 + y_3 = (r_{bt} + N_b)/V_b\,, \tag{16}$$

where

$$0 \le y_1 \le 0.8,$$

$$0 \le y_2 \le 0.1,$$

$$0 \le y_3 \le 0.1.$$

Column generation for truck routes

In integer programming models for scheduling vehicles, too many routes usually exist to consider explicitly. However, any feasible schedule contains relatively few routes, that is, at most one per available vehicle. We use a family of heuristics to produce good candidate routes, i.e., routes that drive the optimization forward and have the potential of being part of an optimal schedule. This can be considered as a *heuristic column-generation procedure*.

Recall that each possible route corresponds to a variable in the mathematical model; thus, it defines a column of the matrix that describes the model. The ability to generate good columns quickly is the critical test of a column-generation-based integer programming approach. We have several methods in our tool set for generating good routes. They are described in the sections that follow.

Column-generating techniques are discussed in [8, 9]. For background on vehicle-routing problems and solution approaches, see [10–14]. More recently, researchers have considered a problem similar to ours, including dynamic scheduling of both pickups and deliveries, but their objective function was different from ours [15].

Closest-neighbor algorithm

For a given truck, we pick at random a vehicle among the k broken ones that are closest to the current location of the truck and that require some part that is currently carried by the truck. We add this vehicle as the next stop to the route of the truck. Then, from this location of the vehicle, we look for another broken vehicle to visit next among the k closest ones, and so on. We repeat until no more broken vehicles can be fixed with the spare parts that the truck carries. The next stop on the route is then the BSB, where the truck goes to load new spare parts. Then we look again for a broken vehicle among the k closest ones, as before. We add stops to the route in this fashion until the truck on the route has visited all broken vehicles or the route lasts longer than the planning horizon. The truck capacity is taken into account when choosing each stop, so that a broken vehicle requiring a part that does not fit into the truck is ignored. Our experiments indicate that one should give the value 3 or 4 to the parameter k. In this heuristic, only one truck is considered at a time.

Clarke–Wright algorithm

The Clarke–Wright algorithm is a well-established procedure used for vehicle routing [16]. We start with elementary routes and then combine them to create longer ones that are time-efficient, as follows.

We first create elementary routes, in which a truck starts at the BSB (denoted β), visits a broken combat vehicle i, and returns to the BSB. So, for each broken vehicle i, we create the route $\{\beta, i, \beta\}$. The total number of routes equals the number of broken vehicles, which typically exceeds the number of trucks available. Thus, we need to combine these routes to produce one route per truck.

We then combine routes recursively, as follows. We examine together a route that includes broken combat vehicle i as a last stop before returning to the BSB, $\{\beta, \cdots, i, \beta\}$, and a route that includes broken vehicle j as a first stop after the BSB, $\{\beta, j, \cdots, \beta\}$. Let c_i and c_j be the time length of each route, respectively. Now we consider the combined route $\{\beta, \cdots, i, j, \cdots, \beta\}$ where the truck takes on the second route directly after completing the first and skips the intermediate visit to the BSB. Of course, we must reject combinations that are infeasible, for instance because the truck cannot obtain all necessary parts at the BSB. Let c_{ij} be the length in time of this combined route. The time savings of the combined route compared with the two single ones is $c_i + c_j - c_{ij}$. For every pair i and j of routes, we calculate the time savings of their possible combinations, and we combine the two routes with the largest savings, breaking ties randomly. We repeat this until the number of routes is equal to the number of trucks.

Cluster-and-route algorithm

Another procedure used to generate routes involves the creation of a cluster of broken combat vehicles for each truck and then the creation of a route that includes all vehicles in the cluster.

The clustering algorithm is as follows. First mark as "available" all broken vehicles. Then move each truck to a closest available vehicle and mark it as unavailable. Continue until all broken vehicles have been assigned, producing a cluster for each truck.

When defining the clusters, all trucks are treated in parallel, each of them making one move at a time, in contrast to the closest-neighbor procedure, in which the trucks are treated sequentially. Once the clusters have been defined, the routing of the truck is done in each cluster as in the closest-neighbor algorithm.

After we schedule visits to all vehicles in a cluster using parts available on the truck, we add additional stops to the route, such as the BSB or a company where the truck can pick up spare parts, and then further repairs may be added to the route.

For the cases in our test set, we observed that generating 700 to 1,000 possible routes with the three methods provided good results. The input of the test case specified a 72-hour horizon, 36 types of parts, seven repair trucks, eight battalions, 19 companies, and approximately 150 broken combat vehicles. From these routes, seven were chosen by the solution method we describe in the next section.

Solving the integer program with the fix-and-resolve method

To solve the integer program, we first solve its *linear relaxation*; that is, we allow the binary integer variables to assume fractional values, and we solve the resulting linear program using the simplex method [17] and the open-source solver CLP (available from COIN-OR [18]). Next, we examine the values of the relaxed binary variables x_j in the solution. Some of them have value 0 or 1, which means that the solution respectively excludes or includes them. Others have intermediate values: For example, the solution may feature two routes for a truck, each with a value of 0.5. This indicates that the solution does not select one route over the other. This inconclusive situation does not help us in selecting a route. On the other hand, a route with value close to 1 suggests that this route can be part of a good solution. We have designed our iterative solution procedure around this concept, as follows.

We first set the variables with a fractional value close to 1 to the value 1, thus "fixing" them. If any variables are at least 0.999999, they are chosen for such fixing. Otherwise, we choose the variable closest to 1 and fix only that variable before the next iteration. However, if fixing a variable to the value 1 makes the linear program infeasible, it is fixed to 0. Next, we reduce the size of the problem by removing all of the trucks, inventory, and fixed combat vehicles associated with the routes that are thus fixed. We then solve the linear relaxation again in order to optimize the smaller problem. We repeat these steps, fixing new variables close to 1, until no more variables are assuming a fractional value. We call this procedure *fix and resolve*. It is designed to produce a good solution quickly. (As we fix more variables, the linear programs become smaller, and we expect to solve them faster.)

Delivering all spare parts

The solution of the integer program assigns each truck to a repair route for broken combat vehicles and assigns load operations for the spare parts required. We can make additional use of this route to deliver to the companies the parts specified by the inventory optimizer. First, we mark all companies that have parts to be delivered to them as "unfulfilled." We then produce a random ordering of the trucks that are non-idle, that is, that have routes assigned to them. For each truck we carry out the following steps:

1. For each part, we compute the total quantity to be delivered to the companies on the route of the truck.
2. We examine whether the BSB has some or all of these parts. We also check whether some other company has excess quantities available for cross-leveling.
3. If some of the parts are available, we load the parts on the truck, up to its capacity limit.
4. As truck travels its route and delivers parts to the companies, we note whether or not the parts delivered meet the total demand of each company. In the first case, we mark the company as "fulfilled."

At the end of this algorithm, some companies may be unfulfilled; that is, their demand may have been filled only partially or not at all. For these companies, we reserve one truck to run a special route: It starts from the BSB, tours the unfulfilled companies delivering parts, and returns to the BSB. We schedule this truck using a closest-neighbor heuristic as follows. We greedily create a route for the truck, which starts at the BSB, then visits the unfulfilled company closest to the BSB, then the unfulfilled company closest to that company, and so on. Note that a greedy algorithm chooses the least expensive or most profitable option at each step without regard to cost or benefit in later steps. The last stop is the BSB. For each company on the route, we calculate the quantity of parts needed to reach "fulfilled" status. We create a load plan for the truck, in which the truck loads the parts needed at the BSB. If the BSB cannot supply the full quantity needed, we then (if possible) add load operations at companies that have excess inventory of the part needed, and these companies are visited earlier by the truck on the route. In all cases, we are careful not to exceed the capacity of the truck.

6. Local search algorithm

In contrast to our integer programming algorithm, our local search algorithm addresses both terms of the objective function simultaneously. The general idea for a local search algorithm is to start with a feasible schedule and improve it in small (local) steps until no further improvement can be found [19, 20]. At each step, we attempt to improve the scheduling of trucks for combat vehicle repair, as well as the overall distribution of

spare parts. This is accomplished by exploring the neighborhood of the current solution and replacing it with another, better solution in the neighborhood. Solutions in the neighborhood are evaluated according to the objective function in Equation (8). The process stops when a local optimum is reached, that is, when we have a schedule that cannot be improved by local improvement steps.

To define a local search algorithm precisely, we must specify how to find a feasible schedule that can be used as a starting point and how to define the search neighborhood, that is, the candidates for an improvement step.

In choosing the neighborhood we balance two opposing objectives. Since we need to improve the current schedule, the neighborhood must be sufficiently large and varied to contain a significantly better solution. However, we must be able to explore the neighborhood quickly. Techniques exist for exploring very large neighborhoods [21]; we manage the search by keeping the size of the neighborhood relatively small.

Several options also exist for choosing an improvement in a neighborhood: For example, we can take the best solution in the neighborhood, or we can take the first solution that is better than the current one. The second option decreases the running time of an iteration, but it may also decrease the magnitude of the improvement.

Details of the local search algorithm

Initialization

The starting point can be the schedule generated by the integer programming algorithm described above, a schedule produced by any other algorithm (for instance, a simple greedy algorithm), or no schedule at all. In the first option we are motivated by the fact that the integer programming method does not generate all possible routes—just a small, manageable subset of good routes. It is possible, at least in theory, that better routes exist, and we can use the local search method to improve upon the routes produced by integer programming. Sometimes an improvement that is relatively small in terms of the objective function, but obvious to the human analyst, may be missed by the global view of the IP solution, and a local search can find and correct this, which may increase user confidence in the IP solution. Section 7 briefly mentions another algorithm that could be used as a starting point of the search algorithm. In our prototyping effort, we implemented only the third option; that is, we, simply start with the empty solution, in which each truck stays idle, and make local improvement steps from there. As described below, one of our local improvement steps inserts a new action into a route; inserting into an empty route is simply a special case. However, we use specialized techniques, which we omit for brevity in this paper, to greatly speed up the initial iterations until we have a "reasonable" schedule to use as a starting point.

Improvement steps

The set of feasible improvement steps that we explore (i.e., the changes that define the neighborhood) are the following:

1. Insert a previously unscheduled delivery/repair load operation into one of the existing routes.
2. Perform a simple delete-and-reinsert swap as follows. Pick a route and an unload or repair operation and delete the operation. Delete the corresponding load operation in this route. Insert the deliver/repair operation into a different position in the same route, or insert it into some other route. For the repair/unload inserted operation, examine the route up to that operation in order to find a location and time to insert a corresponding load operation. Add the load operation to the route.
3. If the algorithm cannot accommodate unscheduled orders and insert them into a current schedule, delete one of the scheduled orders and insert one of the unscheduled ones. Although this action may seem not to make progress, it may improve overall availability, because it may repair a broken combat vehicle sooner or repair a higher-priority broken vehicle.

Our approach closely resembles that of work described in [22], which addresses a similar problem but with deliveries only rather than load/unload pairs.

7. Stabilized greedy algorithm

Our third transportation-scheduling algorithm can be thought of as a sophisticated improvement on a natural greedy strategy. The algorithm has three phases: oscillation avoidance, allocation of trucks to battalions, and routing of individual trucks. For brevity, the details of this approach are omitted from this paper; however, readers may contact the authors for more information.

8. Combining the algorithms

The algorithms described in the previous sections are combined into a single inventory and transportation optimizer as follows: On each incremental receipt of new data, we run the inventory allocator and then all three transportation algorithms in parallel. Each algorithm produces its own schedule. Three-hour granularity is typically used, but experiments were carried out with values as small as one hour and as large as 24 hours. We evaluate the quality of these schedules using the combined objective function in Equation (8) over the full 72-hour

planning horizon. We choose the best of the three schedules according to the objective function, and return the first increment of this schedule to the simulator. The first increment is executed, and the process repeats. The previous schedule is discarded, and a new schedule is derived using the updated snapshot of the situation.

Thus, we re-optimize at fine granularity and continually incorporate the latest available data into our solution. Below we refer to each such run of the optimizers as an "iteration." The 72-hour horizon is used to find the best plan, assuming that nothing changes, at each time increment. If new data becomes available at the next time increment, a better solution for the new situation may be possible.

Several benefits are derived from using multiple transportation algorithms. The first is high solution quality: We are able to choose the best of three available solutions. Different algorithms may have better success in different situations. Because we optimize at a fine time granularity, we obtain the best performance of all of the algorithms rather than having to use a solution from a single algorithm. Another benefit is robustness. The likelihood of all three algorithms failing or producing low-quality solutions is small. Thus, we are likely to have a high-quality solution at every time period.

9. Numerical results

The goal of our numerical experimentation was to determine whether NCL capabilities result in a significant increase in operational availability of the total force when compared with the capability provided by a traditional logistics distribution system, particularly under conditions of high uncertainty and frequent change. We also measured the computational resources required by NCL and compared the performance of the three different transportation scheduling algorithms.

Simulation scenario

The scenario used in this numerical study, provided by our customer, was designed to evaluate and compare the traditional and NCL approaches to logistics distribution over a 30-day simulation. The simulated brigade organization encompassed the headquarters and headquarters company, brigade combat team, three infantry battalions, a reconnaissance surveillance and target acquisition squadron, and anti-armor and engineer companies, together comprising 302 combat vehicles. The brigade combat team was assumed to be network-enabled on the basis of the emerging future combat system capabilities of the U.S. Army. More specifically, repair parts can be distributed to subordinate units carried on combat vehicles and on combat repair team vehicles. Combat vehicles are equipped with a sensor suite that provides continuous and total visibility of brigade repair parts inventory and parts requirements.

The operations plan of the brigade was based on a hypothetical five-phased military operation. For computer simulation purposes, the area of operations terrain was converted to a map with nodes representing key terrain features, cities, bridges, junctions, and arcs that represented roads connecting the nodes. Repair part delivery times from the external theater support base to the BSB were calculated on the basis of actual distances between the two locations and allowable ground transportation speeds. Various perturbations were introduced into the simulation in order to apply stresses that affect the repair part delivery process. Demand was increased by increased combat; inventory was lost because of enemy action; communications were degraded; and the road network was changed, for instance by the destruction of bridges. Each of these stresses was varied among three different levels of intensity.

Measures of performance (A_o and CWT)

The benefit of NCL with respect to the baseline system was assessed in terms of increased operational availability, A_o, and reduced customer wait time, CWT, as described in Section 2. We executed more than 700 experiments to gain insight and understanding of how NCL, enabled by state-of-the-art optimization algorithms, differs from and improves upon the traditional logistics system. Each experiment used a 30-day battlefield scenario, along with various combinations of "stressors" such as increased breakage rates, inventory losses, communications breakdowns, and road closures. Again, for brevity, we omit detailed discussion of the scenarios, which can be obtained from the authors. We found that NCL capabilities result in a significant increase in operational availability and reduction in customer wait time compared with a traditional logistics distribution system. **Figure 4** shows the brigade-wide operational availability that resulted from the baseline and NCL logistics systems in an unstressed scenario.

Because the BSB is not deployed until day 4, a backlog of parts requirements is created, and several days are required to overcome the backlog in both the NCL and the baseline cases. The value of A_o then becomes more level in both cases, but at a much higher level under NCL. The second dip in the baseline A_o value does not occur in the NCL case because of the on-board spare parts capability of the latter. In the unstressed scenario, the average NCL A_o value for the brigade was 0.96, compared with 0.86 for the baseline system. The average CWT value for the brigade overall was 7 hours (NCL) vs. 52 hours (baseline).

In the fully stressed case, NCL was again superior, as illustrated in **Figure 5**. The A_o value was 0.93 under NCL,

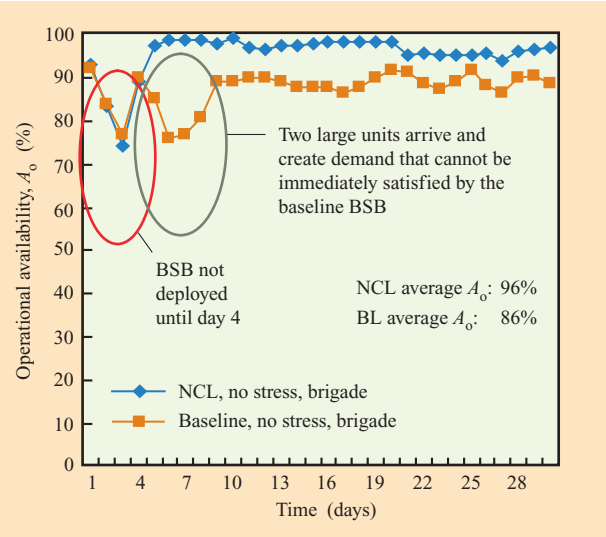

Figure 4

NCL vs. baseline brigade-level availability in an unstressed scenario.

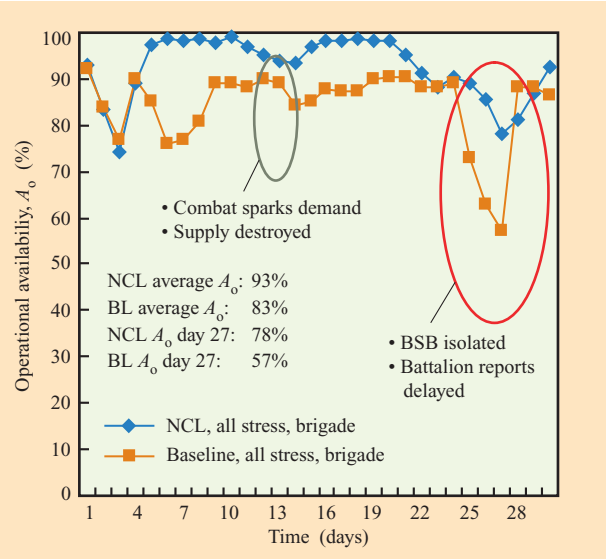

Figure 5

NCL vs. baseline brigade-level availability in a stressed scenario.

compared with 0.83 under the baseline system. Most dramatic was the performance during the period of highest stress, from day 24 to day 28, when combat units were cut off from the BSB. Although the baseline system recovered to its steady state a day sooner than the NCL system and achieved a better A_o value on day 28, the downward spike is much deeper for the baseline. The overall brigade baseline A_o value dropped to 0.57 on day 27 compared with the NCL A_o value of 0.78. The *CWT* value for the fully stressed brigade averaged 8 hours (NCL) compared with 106 hours (baseline).

Performance comparisons of transportation scheduling algorithms

To compare the solution quality performance of the three transportation scheduling algorithms, we collected statistics on a set of 75,300 iterations (about 2/5 of the total number of iterations in all experiments). Normalizing the objective function to a scale of 0 to 100, the average winning (largest) score was 89.6. The average gap between the best and worst of the three scores was 0.16, and the maximum gap was 14.4. **Table 2** shows for each algorithm the number of iterations on which the solution provided by that algorithm was best among the three (see the "Wins" row). It also shows the number of times the score of each algorithm exceeded the others' scores by more than 0.48, or three times the average gap. These are denoted "Big wins." The most frequent outcome was that the three algorithms achieved the same score, within a 0.01 margin. This was denoted "Three-way tie" in Table 2. Integer programming is the most frequent winner, but no algorithm dominated any other algorithm—i.e., no algorithm was as good as or better than the others in all cases.

Running time performance

In production-level testing, the running time of the inventory allocation module was always less than one second. In **Table 3** we show the statistics on the running times of the three transportation scheduling algorithms. These are taken over all iterations of all experiments. Each running time data point refers to one invocation of the optimizer. For example, note that 240 invocations correspond to a run of 720 hours at three-hour granularity. We show the overall minimum, maximum, and mean for each algorithm, and also the same statistics with the highest and lowest 5% of values removed, denoted by "no outliers" in Table 3. Our method for measuring running times of the local search algorithm differed slightly, and we have only upper bounds on its running times. As can be seen from the table, the computational requirements of the algorithms are quite modest.

10. Summary and conclusions

The methods and results presented in this paper indicate that mathematical optimization models have the potential to significantly improve military logistics operations, and in particular to increase operational availability of combat vehicles. The simulations considered here were

Table 2 Comparison of win tallies, i.e., the number of iterations, as explained in the text, among the three transportation scheduling algorithms.

	IP	*LS*	*SG*	*IP and LS*	*IP and SG*	*LS and SG*	*Three-way tie*
Wins	23,476	5,418	11,085	3,641	3,970	1,345	26,374
Big wins	2,635	1,597	899	123	19	11	—

limited to a small class of supplies. The very modest computational requirements suggest that problems of much larger scale can be solved by the proposed optimization techniques. We believe that further investigation is warranted to expand the scope of the problem in order to obtain greater insight and more accurate results regarding the amount and types of improvement, relative to different performance measures, that can be achieved.

Acknowledgments

We wish to thank our many colleagues contributing to this project: Marshall Brinn, Aaron Helsinger, John Phelan, Ray Tomlinson, and Todd Wright of BBN Technologies; Joseph Cross and Mark Greaves of DARPA; John Kirzl and Dave Signori of Evidence Based Research, Inc.; Steve Buckley, Igor Frolow, Spyros Kontogiorgis, Yahong Gu, and John McCann of IBM; Tony Rozga of LMI Government Consulting; Leo Pigaty of Los Alamos Technical Associates, Inc.; and Mike Dyson of the Schafer Corporation.

Table 3 Running times of the three transportation-scheduling algorithms.

Algorithm	*min* (s)	*max* (s)	*mean* (s)
IP	1.39	68.2	3.18
IP no outliers	1.62	6.01	2.97
LS	<1.0	<84.3	<1.60
LS no outliers	<1.0	<2.10	<1.06
SG	0.53	1.84	0.87
SG no outliers	0.63	1.00	0.87

References

1. M. Ettl, G. Feigin, G. Lin, and D. Yao, "A Supply Network Model with Base-Stock Control and Service Requirements," *Oper. Res.* **48**, No. 3, 216–232 (2000).
2. V. Deshpande, M. Cohen, and K. Donohue, "An Empirical Study of Service Differentiation for Weapon System Service Parts," *Oper. Res.* **51**, No. 4, 518–530 (2003).
3. S. Graves and S. Willems, "Supply Chain Design: Safety Stock Placement and Supply Chain Configuration," *Supply Chain Management: Design, Coordination and Operation*, A. De Kok and S. Graves, Editors, *Handbooks in Operations Research and Management Science* **11**, Elsevier, 2003, pp. 95–132.
4. J. Muckstadt, *Analysis and Algorithms for Service Parts Supply Chains*, Springer Series in Operations Research and Financial Engineering, Springer, New York, 2004.
5. F. Cheng, M. Ettl, G. Lin, and D. Yao, "Inventory-Service Optimization in Configure-to-Order Systems," *Manuf. Services & Oper. Manage.* **4**, No. 2, 114–132 (2002).
6. B. Dietrich, D. Connors, T. Ervolina, J. Fasano, R. Lougee-Heimer, and R. Wittrock, "Applications of Implosion in Manufacturing," *Supply Chain Management On Demand*, C. An and H. Fromm, Editors, Springer, New York, 2005, pp. 97–115.
7. K. Caggiano, J. Muckstadt, and J. Rappold, "Integrated Real-Time Capacity and Inventory Allocation for Repairable Service Parts in a Two-Echelon Supply System," *Manuf. Services & Oper. Manage.* **8**, No. 3, 292–319 (2006).
8. C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs," *Oper. Res.* **46**, No. 3, 316–329 (1998).
9. J. Desrosiers, F. Soumis, and M. Desrochers, "Routing with Time Windows by Column Generation," *Networks* **14**, No. 4, 545–565 (1984).
10. J. Bramel and D. Simchi-Levi, "On the Effectiveness of Set Covering Formulations for the Vehicle Routing Problem with Time Windows," *Oper. Res.* **45**, No. 2, 295–301 (1997).
11. J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis, "Time Constrained Routing and Scheduling," *Network Routing*, M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, Editors, *Handbooks in Operations Research and Management Science* **8**, North-Holland, Amsterdam, 1995, pp. 35–139.
12. M. Solomon and J. Desrosiers, "Time Window Constrained Routing and Scheduling Problems," *Transport. Sci.* **22**, No. 11, 1–13 (1988).
13. H. Psaraftis, "Dynamic Vehicle Routing: Status and Prospects," *Ann. Oper. Res.* **61**, 143–164 (1995).
14. M. Gendreau and J. Potvin, "Dynamic Vehicle Routing and Dispatching," *Fleet Management Logistics*, T. Crainic and G. Laporte, Editors, Kluwer, Boston, 1998, pp. 115–226.
15. M. Gendreau, F. Guertin, J. Y. Potvin, and R. Séguin, "Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-Ups and Deliveries," *Transport. Res. Part C* **14**, 157–174 (2006).
16. G. Clarke and J. V. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Oper. Res.* **12**, No. 4, 568–581 (1964).
17. G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1998.
18. COIN-OR, Computational Infrastructure for Operations Research; see *http://www.coin-or.org/Clp*.
19. E. Aarts and J. Lenstra, *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, NJ, 2003.
20. T. Ibaraki, K. Nonobe, and M. Yagiura, Editors, "Metaheuristics: Progress as Real Problem Solvers," *Operations Research/Computer Science Interfaces Series*, Vol. 32, Springer, New York, 2005.
21. R. Ahuja, O. Ergun, J. Orlin, and A. Punnen, "A Survey of Very Large-Scale Neighborhood Search Techniques," *Discrete Appl. Math.* **123**, Nos. 1–3, 75–102 (2002).

22. M. Dror and L. Levy, "A Vehicle Routing Improvement Algorithm Comparison of 'Greedy' and a Matching Implementation for Inventory Routing," *Computers & Oper. Res.* **13**, No. 1, 33–45 (1986).

Received September 15, 2006; accepted for publication October 16, 2006; Internet publication May 22, 2007

Francisco Barahona *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (barahon@us.ibm.com).* Dr. Barahona received a Ph.D. degree in operations research from the University of Grenoble, France. His research interests include combinatorial optimization, integer programming, and mathematical programming. He has been a professor at the University of Chile and the University of Waterloo, Canada, and a visiting professor at the University of Bonn, Germany. Dr. Barahona has received an IBM Outstanding Technical Achievement Award, and he holds two patents. He has been an associate editor of *SIAM Journal on Optimization*, *Operations Research*, and *Management Science*.

Pawan Chowdhary *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (chowdhar@us.ibm.com).* Mr. Chowdhary received his bachelor's degree in electronics engineering from Nagpur University, India, in 1996. During his ten-year career, he has been associated with various IBM divisions, where he has architected, designed, and implemented complex, high-performance, and scalable distributed object-oriented applications. Since joining the IBM Research Division in 2004 as an Advisory Software Engineer in the Analytic Models and Architecture Department, he has been working on technologies related to Sense-and-Respond and Business Performance Management (BPM), as well as model-driven software development techniques. Mr. Chowdhary writes extensively in the area of BPM and model-driven techniques.

Markus Ettl *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (msettl@us.ibm.com).* Dr. Ettl is a Research Staff Member at the IBM Thomas J. Watson Research Center. In 1995 he received his doctoral degree in computer science from Friedrich-Alexander University in Erlangen, Germany. Since joining IBM in 1995, he has focused on advanced research in supply chain management, and he holds several patents in this field. Dr. Ettl's current research interests lie in decision support for production systems and logistics networks, and sense-and-respond business management for adaptive organizations. In 1999 he received the INFORMS Franz Edelman Award. His other awards and commendations include two IBM Outstanding Technical Achievement Awards and several IBM Research Division Awards.

Pu Huang *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (puhuang@us.ibm.com).* Dr. Huang received his M.S. degree from the School of Computer Science and his Ph.D. degree from the Tepper School of Business, both at Carnegie Mellon University. He is a Research Staff Member at the IBM Thomas J. Watson Research Center. His current interests are in supply chain management, stochastic optimization, machine learning, and data mining. His work includes sense-and-respond systems that help businesses quickly detect emerging risks and react with corrective actions. Such systems typically integrate intelligent data mining techniques and sophisticated decision support algorithms to provide their functions. Recently, Dr. Huang has worked on extending the same idea to support business decision making in large-scale distributed environments.

Tracy Kimbrel *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (kimbrel@us.ibm.com).* Dr. Kimbrel received B.S., M.S., and Ph.D. degrees in computer science from the University of Washington, joining the IBM Research Division in 1997. His

research on the design of algorithms for optimization problems includes theoretical, experimental, and applied efforts in areas such as operating system scheduling, resource allocation in multi-server systems, and vehicle routing. He has received an IBM Research Division Award and two Research Division Technical Group Awards, and he holds two patents.

Laszlo Ladanyi *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ladanyi@us.ibm.com).* Dr. Ladanyi received an M.S. degree in mathematics from Eotvos Lorand University in Budapest, Hungary, and a Ph.D. degree in operations research from Cornell University. He joined the IBM Research Division in 1996. Dr. Ladanyi's main research areas are integer programming and parallel programming, with a focus on computational aspects.

Young M. Lee *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ymlee@us.ibm.com).* Dr. Lee received B.S., M.S., and Ph.D. degrees in chemical engineering from Columbia University. In 2002 he joined the IBM Research Division, where he has been working in the areas of supply chain simulation and optimization. Before joining IBM, Dr. Lee worked for 14 years for BASF, where he founded and managed the Mathematical Modeling Group and led the development of numerous optimization and simulation models for various logistics and manufacturing processes.

Baruch Schieber *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sbar@us.ibm.com).* Dr. Schieber is the manager of the Optimization Center in the Mathematical Sciences Department at the IBM Thomas J. Watson Research Center. In this capacity he leads a group of computer scientists, mathematicians, and operations researchers in activities combining world-class basic research with the design and implementation of state-of-the-art software in areas such as supply chain management, personnel and vehicle scheduling, production planning, print technology, and intrinsic function acceleration. Dr. Schieber received his Ph.D. degree in computer science from Tel Aviv University, Israel, in 1987 and joined IBM as a Postdoctoral Fellow. His main interests are approximation algorithms, scheduling, and routing. Dr. Schieber has published more than 50 papers in scientific journals; he is a regular presenter at leading theoretical computer science conferences.

Karthik Sourirajan *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ksourira@us.ibm.com).* Dr. Sourirajan is a Research Staff Member in the Mathematical Sciences Department at the IBM Thomas J. Watson Research Center. He received a B.E.(Hons.) degree in mechanical engineering at the Birla Institute of Technology and Science, Pilani, India, in 2000, followed by M.S. (2002) and Ph.D. (2006) degrees in industrial engineering from Purdue University. His research interests include the application of operations research techniques to solving real-world problems that are generally related to supply chains, such as forecasting, inventory management, integrated facility location, and logistics.

Maxim I. Sviridenko *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sviri@us.ibm.com).* Dr. Sviridenko received B.S., M.S., and Ph.D. degrees in applied mathematics and computer science from the Novosibirsk State University, Russia. He joined the IBM Research Division in 2000. He holds one patent and has received an IBM Research Division Award and a Research Division Technical Group Award for work on the design of algorithms for various optimization problems.

Grzegorz M. Swirszcz *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (swirszcz@us.ibm.com).* Dr. Swirszcz joined IBM in 2003 after receiving M.S. and Ph.D. degrees from the University of Warsaw, Poland, and spending three years as a Marie Curie Postdoctoral Fellow at the Centre de Recerca Matemática in Barcelona, Spain. He works on mathematical methods in modeling, optimization, qualitative theory of differential equations, and applications of mathematics in signal analysis and business processes. Dr. Swirszcz is currently working on fault-detection algorithms in electric networks. He also continues his fundamental research in various areas of pure and applied mathematics. His interests involve creative and innovative applications of advanced mathematics and problem solving.