Transportation Scheduling Methods

Benjamin K. Peterson

Tepper School of Business
Carnegie Mellon University

January 2010

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Algorithms, Combinatorics and Optimization.

Abstract

This thesis presents two algorithms for solving two combinatorial transportation problems and a model for optimizing a robust transportation schedule. The first algorithm solves a problem of operating multiple cranes used in a factory to transport equipment and materials. The second algorithm is a logic-based Benders decomposition method for a common routing and delivery problem. Lastly, we look at a model for designing a robust vehicle routing plan at a U.S. appliance factory.

Contents

Chapter 1: Introduction

Transportation logistics not only offers a wide array of interesting combinatorial problems, but is also one of the most valuable applications of operations research in industry. This thesis presents three specific applications, each motivated by a real-world industry problem. The applications are summarized below.

1.1 An Algorithm for Scheduling Multiple Factory Cranes on a Common Track

Chapter 2 presents an algorithm used to solve a machine scheduling problem in which the machines are mechanical cranes moving materials and equipment around a factory floor. The cranes are physically constrained from crossing paths in the longitudinal direction because they are mounted on the same overhead track. Since there is too much work for a single crane to complete in the time available, multiple cranes must be employed to perform the work while carefully coordinating their movements. The objective is to complete the set of required tasks on the cranes as quickly as possible. This research was motivated by a real-world case in which crane interference was causing production delays at a copper-smelting plant in Germany.

The algorithm works by constructing a solution in chronological order and branching on important scheduling decisions, such as choosing a crane to assign to a task or choosing the order in which the tasks are performed. By branching on each of these decisions, the algorithm constructs a tree of possible states of the crane system at times throughout the schedule. As the decision tree expands, the algorithm employs a pruning procedure to identify states in the tree which are infeasible or suboptimal and deletes those states.

If the algorithm manages to keep the size of the decision tree within memory limits, it will find an optimal solution to the crane scheduling problem. If not, it still finds a near-optimal solution and therefore serves as a very useful heuristic.

1.2 A Benders Approach to a Transportation Scheduling Problem

Chapter 3 presents a specialized Benders decomposition algorithm for solving a common routing and delivery problem. Our problem model contains a set of feasible routes and a set of required deliveries. Each route has a cost and capacity, and each delivery has a size and a subset of the routes on which it can be shipped. The objective is to find the minimum-cost set of routes which can ship all of the deliveries. We study the case in which some of the deliveries may be split into multiple shipments, whereas others must be shipped entirely by a single route. This research was motivated by a case at the United States Post Office in which departure times needed to be carefully selected for a set of routes carrying high-priority time-sensitive packages.

Our Benders decomposition separates the route selection problem into two components: the first is our master problem which simply solves for a minimum-cost set of routes. The second is our subproblem which tests whether there exists a feasible set of assignments for all deliveries to those routes. If not, a subset of the deliveries is identified which is itself infeasible, and a constraint is added to the master problem to ensure that this infeasibility does not reoccur in future master solutions.

Experiments with our Benders algorithm demonstrate improved performance over standard methods for problem instances with certain common characteristics.

1.3 Optimization of Inbound Freight Transportation at B/S/H/ North America

Chapter 4 presents a model for optimizing transportation costs at an assembly plant with many suppliers and a high uncertainty of demand. The B/S/H/ appliance factory in New Bern, NC employs thousands of trucks from third-party carriers every year to move parts and materials from vendors across North America. In order to maintain flexibility in its purchasing decisions from each vendor, each purchase is shipped individually from the vendor to the factory. We will explore a model for designing a robust shipping schedule that takes advantage of correlations and economies of scale by pooling freight from multiple vendors into single shipments while still maintaining an acceptable amount of demand flexibility.

Much like the model in Chapter 3, the model here includes a set of routes from which to choose a minimum-cost set that can ship the required freight. These routes are designed by exploring all combinations of 1-, 2- and 3-stop paths between vendors that then proceed to the B/S/H/ factory. For each path identified, various combinations of loads are explored that may divide the freight of individual vendors into multiple parts for shipment on multiple routes. For each load combination, a statistical analysis is used to estimate the probability that the load will meet the capacity requirements of the truck that hauls it. A lower bound is imposed on this probability as part of the constraints of the model.

A feasible solution to the model is a complete set of routes and loads that will carry the required freight to the B/S/H/ factory from its suppliers on a weekly basis. The optimal solution is 25% cheaper than the previously used shipping schedule with a similar degree of flexibility.

Chapter 2: An Algorithm for Scheduling Multiple Factory Cranes on a Common Track

2.1 Introduction

Factories often employ track-mounted cranes to move materials and equipment from one location to another. Such cranes typically hang from crossbars on which they move back and forth while the crossbars themselves move left and right along a common track (Figure 2.1). This combination allows full two-dimensional movement of the cranes across the factory floor.

Planning the movement of a crane to follow a manufacturing schedule is a real-world combinatorial problem. For a single-crane factory, the problem consists of sequencing the tasks on the crane in the order that minimizes their completion times subject to some constraints. For a multiple-crane factory, there exist additional problems of assigning cranes to tasks and selecting movement priority when the cranes interfere.

Constraints are used to model the variety of work that a crane performs. Each task may have an individual time window and priority. Some tasks must be performed on a specific crane, in a specific order, or in a consecutive sequence.

Since crane paths must be planned on a scale of seconds for work schedules on a scale of days, the crane scheduling problem is too complex for general solution methods. This paper presents an efficient specialized algorithm for the problem. Although the

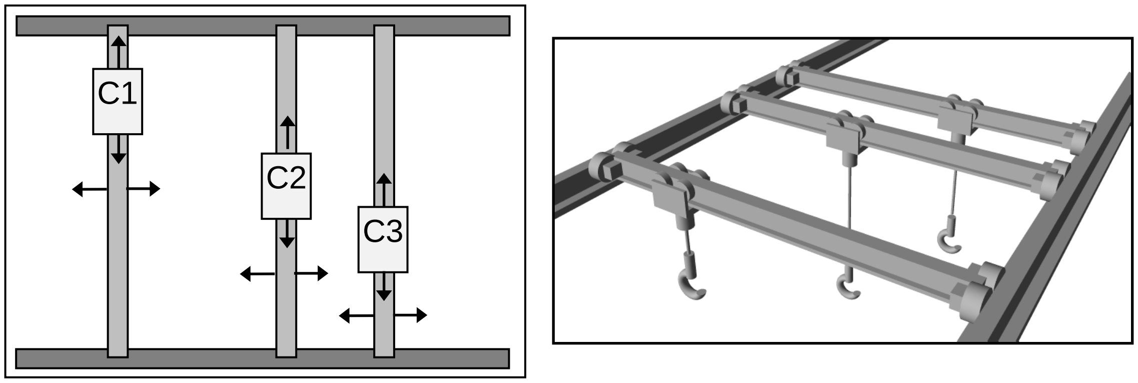


Figure 2.1. Typical configuration of factory cranes.

number of feasible crane trajectories grows exponentially with the length of the work schedule, our algorithm intelligently confines its search for optimal trajectories to a limited solution space. In cases where the work schedule is fairly constrained, the algorithm will often return an optimal solution. In cases where the schedule has a great deal of freedom, the solution space of trajectories may be too large, and the algorithm will return a near-optimal approximation.

2.1.1 Previous Work

The crane scheduling problem has been researched in several other contexts, including electroplating (often referred to a *hoist scheduling*) and port crane scheduling (Kim and Park 2004). Much of this research deals only with single-crane problems (Phillips and Unger 1976) (Chen, Chu and Proth 1998) (Lei and Wang., Determining optimal cyclic hoist schedules in a single-hoist electroplating line 1994), which makes the trajectory design component trivial. However, even this single-crane problem has been proven to be NP-complete (Lei and Wang, A proof: The cyclic hoist scheduling problem is NP-complete 1989).

Other research has dealt with problems involving multiple cranes. In many cases, the solution is an optimal partition of the movement space so that tasks in each partition can be worked on by a single crane (Lei and Wang, The minimum common-cycle algorithm for cycle scheduling of two material handling hoists with time window constraints 1991) (Zhou and Li 2009). Of course, a better solution can be obtained if the cranes are allowed to work in the same space. Some research has studied a more relaxed version of the problem in which cranes are allowed to work in the same space, but the solutions require tasks to be scheduled so that interference will never occur (Lei, Armstrong and Gu, Minimizing the fleet size with dependent time-window and single-track constraints 1993) (Rodo'sek and Wallance 1998) (Che and Chu 2004). Our algorithm instead allows a crane to move to an arbitrary location in order to yield to another crane.

2.2 The Crane Scheduling Problem

2.2.1 Task Assignment and Scheduling

The crane scheduling problem consists of a list of tasks, each of which has a time window (release time and deadline) derived from the production schedule of the factory. Each task τ represents an instruction to "go to location (X_τ,Y_τ) and wait L_τ seconds," where L_τ is the duration of the operation to be performed at that location. Each task also has a priority value to represent its urgency.

The decision variables of the crane scheduling problem have three components. First, each task is assigned to exactly one crane. Second, each task is assigned a start time. Third, a set of crane trajectories is constructed to comply with those assignments and start times. The trajectories describe paths for the cranes to follow to get them to their assigned tasks at their assigned times without interference.

In each problem, task start times are constrained by their time windows and crane trajectories are constrained by their physical movement restrictions. In addition, constraints may be imposed on the assignment and sequence of tasks for a specific problem instance. Our algorithm was designed to be compatible with the following three types of supplementary constraints:

$\tau \in c$ indicates task τ must be performed by crane c

$\tau_u < \tau_v$ indicates the start of task τ_v must (eventually) follow the completion of task τ_u (on any crane)

$\tau_u \rightarrow \tau_v$ indicates task τ_v must immediately follow task τ_u on the same crane

Production schedules often include short sequences of tasks we call jobs. For example, a 2-task job might be the pickup and delivery of a piece of equipment. The tasks in each job use the constraint $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \cdots$ to enforce their consecutive sequence on a single crane.

2.2.2 Problem Formulation

We formulate the crane scheduling problem as follows.

Parameters:	$\boldsymbol{C}$	= set of crane indices
	$\boldsymbol{T}$	= set of task indices
	P_τ	= priority of task τ
	R_τ	= release time of task τ
	D_τ	= deadline of task τ
	L_τ	= duration of task τ
	X_τ	= x-position of task τ
	Y_τ	= y-position of task τ
	V_x	= maximum x-axis crane velocity
	V_y	= maximum y-axis crane velocity
	ΔX	= minimum x-axis distance between adjacent cranes

Variables:	a_τ	= (*int*) assigned crane index for task τ (cranes indexed from left to right)
	s_τ	= (*real*) start time of task τ
	$x_c(t)$	= (*real*) x-position of crane c at time t
	$y_c(t)$	= (*real*) y-position of crane c at time t

(Note that time t is continuous.)

Full Crane Scheduling Problem

Objective

minimize $\sum_{\tau \in \mathbf{T}} P_\tau (s_\tau - R_\tau)$

Constraints

$$\begin{array}{r}\text{domains,}\\ \text{time windows}\end{array}\left\{\begin{array}{l}\forall \tau, a_\tau \in \mathbf{C}\\ \forall \tau, R_\tau \leq s_\tau \leq D_\tau - L_\tau\end{array}\right.$$

$$\begin{array}{c}\text{supplementary}\\ \text{task}\\ \text{constraints}\end{array}\left\{\begin{array}{ll}\forall\ \tau \in c, & a_\tau = c\\ \forall\ \tau < \tau', & s_\tau + D_\tau \leq s_{\tau'}\\ \forall\ \tau \rightarrow \tau', & \left\{\begin{array}{l}a_\tau = a_{\tau'}\\ \quad\text{and}\\ s_\tau \leq s_{\tau'}\\ \quad\text{and}\\ a_{\bar{\tau}} = c \Rightarrow s_{\bar{\tau}} \notin (s_\tau, s_{\tau'})\end{array}\right.\end{array}\right.$$

$$\begin{array}{c}\text{trajectory}\\ \text{constraints}\end{array}\left\{\begin{array}{l}\text{(continuity) } \forall c \in \mathbf{C},\ x_c(t) \text{ and } y_c(t) \text{ are continuous } \forall t \geq 0\\ \text{(position) } \forall \tau, c \in \mathbf{C},\ a_\tau = c \Rightarrow \big(x_c(t), y_c(t)\big) = (X_\tau, Y_\tau), \forall t \in [s_\tau, s_\tau + L_\tau]\\ \text{(velocity) } \forall c \in \mathbf{C},\ \left\{\begin{array}{l}x_c(t) - \partial V_x \leq x_c(t+\partial) \leq x_c(t) + \partial V_x\\ y_c(t) - \partial V_y \leq y_c(t+\partial) \leq y_c(t) + \partial V_y\end{array}\right., \forall t, \partial \geq 0\\ \text{(spacing) } \forall c_a, c_b : 1 \leq c_a \leq c_b \leq |\boldsymbol{C}|,\ x_{c_a}(t) + (c_b - c_a)\Delta X \leq x_{c_b}(t), \forall t\end{array}\right.$$

The objective function is the priority-weighted sum of the *tardiness* of each task. Tardiness is the difference between the task release and start time. The first set of constraints defines the feasible domains of assignments and start times. The next set enforces supplementary task constraints for the problem instance. The final set defines feasible crane trajectories by four properties: (1) continuity over all time, (2) the presence of each crane at its assigned task locations at their assigned times, (3) maximum velocity constraints, and (4) minimum crane spacing constraints. Note that no constraints explicitly define start or finish positions for the cranes, though this may be done by adding zero-length tasks at specified positions to the beginning and end of the schedule.

2.3 Problem Reduction

Our algorithm operates on a simplified form of the crane scheduling problem which differs from the standard form described in section 2.2.2. The validity of the simple form is not obvious however, and we will demonstrate the reduction of the original problem in three steps as follows.

2.3.1 Projection of Trajectories

The first problem reduction begins by decomposing the crane scheduling problem into two easier problems. The division is similar to a Benders decomposition, except that we do not use an iterative solution technique.

The full crane scheduling problem in section 2.2.2 uses a vector of crane assignments ($\mathbf{a}$), task start times ($\mathbf{s}$) and crane trajectories ($\boldsymbol{x}(t), \boldsymbol{y}(t)$) as decision variables. We decompose this formulation into a primary problem which solves for optimal assignments and start times ($\mathbf{a}, \mathbf{s}$) and a secondary problem which constructs feasible crane trajectories ($\boldsymbol{x}(t), \boldsymbol{y}(t)$) for the solution ($\mathbf{a}, \mathbf{s}$). By adding two special sets of constraints to the primary problem, we can define the feasible space of solutions ($\mathbf{a}, \boldsymbol{s}$) for which there exist feasible trajectories. The primary problem is therefore a projection of the full crane scheduling problem onto the ($\mathbf{a}, \mathbf{s}$) space.

Projected Crane Scheduling Problem

Objective

$$\text{minimize} \quad \sum_{\tau \in \mathbf{T}} P_\tau (s_\tau - R_\tau)$$

Constraints

$$\begin{array}{r} \text{domains,} \\ \text{time windows} \end{array} \begin{cases} \forall \tau, a_\tau \in \mathbf{C} \\ \forall \tau, R_\tau \leq s_\tau \leq D_\tau - L_\tau \end{cases}$$

$$\begin{array}{c} \text{supplementary} \\ \text{task} \\ \text{constraints} \end{array} \begin{cases} \forall\, \tau \in c, & a_\tau = c \\ \forall\, \tau < \tau', & s_\tau + D_\tau \leq s_{\tau'} \\ \forall\, \tau \rightarrow \tau', & \begin{cases} a_\tau = a_{\tau'} \\ \quad \text{and} \\ s_\tau \leq s_{\tau'} \\ \quad \text{and} \\ a_{\bar{\tau}} = c \Rightarrow s_{\bar{\tau}} \leq s_\tau \text{ or } s_{\tau'} \leq s_{\bar{\tau}} \end{cases} \end{cases}$$

$$\begin{array}{l} \text{time gap} \\ \text{constraints} \end{array} \begin{cases} \begin{array}{r} \text{(self-movement constraints)} \\ \forall\, \tau \neq \tau',\ a_\tau = a_{\tau'} \Rightarrow \end{array} \begin{cases} s_\tau + D_\tau + \max\left(\dfrac{|X_\tau - X_{\tau'}|}{V_x}, \dfrac{|Y_\tau - Y_{\tau'}|}{V_y}\right) \leq s_\tau \\ \quad \text{or} \\ s_{\tau'} + D_{\tau'} + \max\left(\dfrac{|X_\tau - X_{\tau'}|}{V_x}, \dfrac{|Y_\tau - Y_{\tau'}|}{V_y}\right) \leq s \end{cases} \\ \\ \begin{array}{r} \text{(interference constraints)} \\ \forall\, \tau \neq \tau',\ a_\tau < a_{\tau'} \Rightarrow \end{array} \begin{cases} X_\tau + (a_{\tau'} - a_\tau)\Delta X \leq X_{\tau'} \\ \quad \text{or} \\ s_\tau + D_\tau + \dfrac{|X_\tau + (a_{\tau'} - a_\tau)\Delta X - X_{\tau'}|}{V_x} \leq s_{\tau'} \\ \quad \text{or} \\ s_{\tau'} + D_{\tau'} + \dfrac{|X_{\tau'} - (a_{\tau'} - a_\tau)\Delta X - X_\tau|}{V_x} \leq s_\tau \end{cases} \end{cases}$$

The projected formulation is the full formulation with no trajectory variables and the trajectory constraints replaced by time gap constraints.

The time gap constraints have an intuitive interpretation. The self-movement constraints ensure that a crane has sufficient time to move between tasks which are assigned to it. After it completes one task τ at time $t = s_\tau + D_\tau$, the crane must schedule a minimum time gap of $max\left(|X_\tau - X_{\tau'}|/V_x, |Y_\tau - Y_{\tau'}|/V_y\right)$ before starting another task τ' at time $t = s_{\tau'}$. This gap allows sufficient time for both x- and y-axis movement. The interference constraints ensure that if two cranes are assigned to two tasks that can not be performed simultaneously, then one must wait for the other to finish before it can travel past the first task and perform the second task.

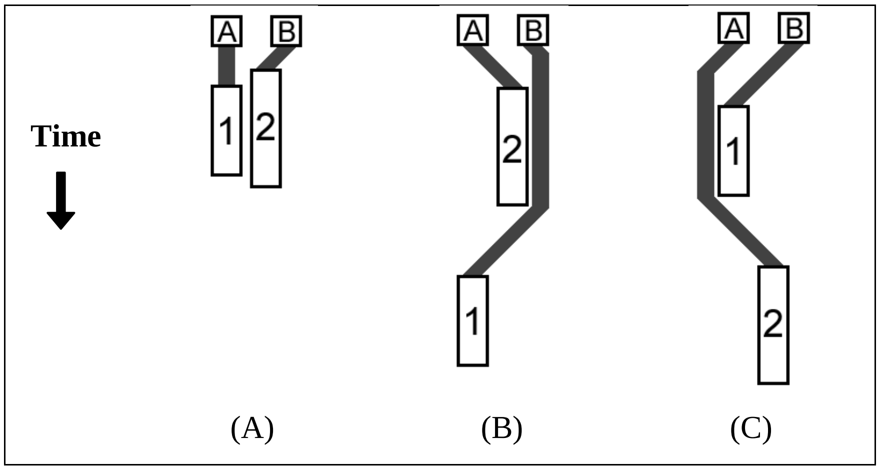
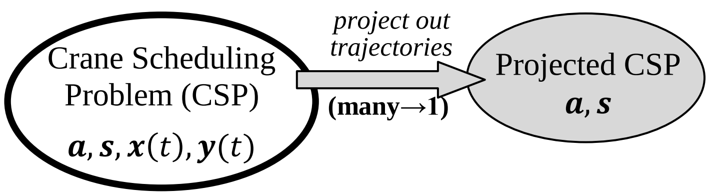


Figure 2.3.1a. Three interference possibilities: (A) no interference, (B) task 1 is performed first, or (C) task 2 is performed first.

It is easy to see that self-movement and interference constraints are necessary to ensure the existence of feasible trajectories. The more interesting result is that they are also sufficient. In fact, these constraints allow our algorithm to search only in the $(\boldsymbol{a}, \boldsymbol{s})$ space for optimal solutions since trajectories are not involved in the objective function. Trajectories can be constructed later once the optimal assignments and start times $(\boldsymbol{a}, \boldsymbol{s})$ have been determined.


Crane Scheduling Problem (CSP)
$\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{x}(t), \boldsymbol{y}(t)$
project out trajectories
(many→1)
Projected CSP
$\boldsymbol{a}, \boldsymbol{s}$

We demonstrate the validity of our projected formulation with the following lemma.

Lemma 2.3.1. For any vector of task assignments $\boldsymbol{a}$ and task start times $\boldsymbol{s}$, the solution $(\boldsymbol{a}, \boldsymbol{s})$ is feasible in the projected crane scheduling problem formulation if and only if there exists a feasible vector of crane trajectories $\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)$ that comply with $\boldsymbol{a}$ and $\boldsymbol{s}$. Specifically, $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ satisfy the following trajectory constraints:

(1) $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ are continuous,

(2) $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ conform to task positions:

$$a_\tau = c \Rightarrow \big(x_c(t), y_c(t)\big) = (X_\tau, Y_\tau), \forall \tau, \forall t \in [s_\tau, s_\tau + L_\tau]$$

(3) $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ conform to velocity constraints:

$$\begin{cases} x_c(t) - \partial V_x \le x_c(t + \partial) \le x_c(t) + \partial V_x \\ y_c(t) - \partial V_y \le y_c(t + \partial) \le y_c(t) + \partial V_y \end{cases}, \forall c, \forall t, \forall \partial \ge 0$$

(4) $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ conform to crane ordering/spacing constraints:

$$x_{c_a}(t) + (c_b - c_a)\Delta X \le x_{c_b}(t), \forall t, \forall 1 \le c_a < c_b \le |\boldsymbol{C}|$$

Proof: Assume the solution $(\boldsymbol{a}, \boldsymbol{s})$ is feasible in the projected crane scheduling formulation. We will define trajectories consistent with $(\boldsymbol{a}, \boldsymbol{s})$ in piecewise segments. Each segment will specify the path of a crane from its last task to its next and to the completion of the next task. We will prove inductively that a complete set of trajectories can be constructed by defining these segments in the order of the task start times $\boldsymbol{s}$.

Let $\boldsymbol{\sigma}$ be an ordering of the task indices that coincides with the order in which tasks are scheduled to start according to $\boldsymbol{s}$ (ties broken arbitrarily), such that $s_{\sigma(i)} \le s_{\sigma(i+1)}\ \forall 1 \le i \le T - 1$. For our base case, we begin with an empty set of trajectories $\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)$ which are consistent with $(\boldsymbol{a}, \boldsymbol{s})$ up to their zeroth elements by default. In other words, they are consistent with an empty set of task assignments and start times.

Assume that for all tasks $\sigma(i)$ where $0 \leq i \leq k$, the trajectory of the crane $a_{\sigma(i)}$ assigned to task $\sigma(i)$ has been defined for all times $t \in [0, s_{\sigma(i)} + D_{\sigma(i)}]$ and that all trajectory constraints are satisfied for those times. In other words, the trajectory of each crane has been defined up to the completion time of its last scheduled task that appears at or before position k in $\boldsymbol{\sigma}$. We will show that a path can be constructed that also satisfies the trajectory constraints for the next crane $c_{next} = a_{\sigma(k+1)}$ assigned to the next task $\tau_{next} = \sigma(k+1)$ starting at time $s_{next} = s_{\sigma(k+1)}$ up through its completion at time $s_{\sigma(k+1)} + D_{\sigma(k+1)}$.

Consider the cone of feasible locations in space-time in which the crane c_{next} must lie in order to reach its task τ_{next} at the location (X_{next}, Y_{next}) at time s_{next}. The cone has its vertex at the time and location of the start of the next task and extends backward in time, expanding in the x and y directions with slopes equal to the maximum velocities of the crane (Figure 2.3.1b). The path of the crane must lie inside this cone if it is to reach its next task on time while obeying its velocity and continuity constraints.

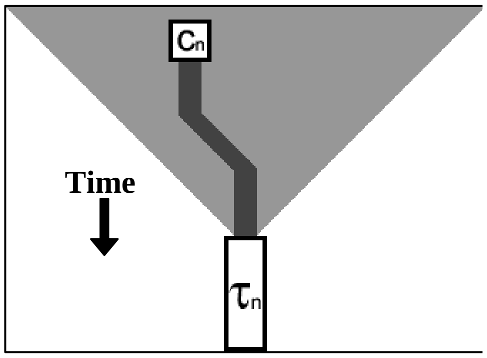


Figure 2.3.1b. Feasible cone

One reason that a feasible path may not exist is if previously-defined paths of other cranes interfere. Since all cranes move at the same maximum velocity, no other crane can “cut off” its path without it having time to move out of the way. Also, the other crane paths maintain a minimum x-axis spacing of $(c_b - c_a)\Delta X$ according to trajectory constraint (4), so no two cranes can be close enough that no path exists between them. Finally, we can use the interference time-gap constraint to show that no crane can force c_{next} out of its feasibility cone.

Interference time-gap constraints:

$$c_{other} < c_{next} \Rightarrow \begin{cases} (1A)\ X_{other} + (c_{next} - c_{other})\Delta X \leq X_{next} \\ \text{ or} \\ (1B)\ s_{other} + D_{other} + \dfrac{|X_{other} + (c_{next} - c_{other})\Delta X - X_{next}|}{V_x} \leq s_{next} \end{cases}$$

$$c_{next} < c_{other} \Rightarrow \begin{cases} (2A)\ X_{next} + (c_{other} - c_{next})\Delta X \leq X_{other} \\ \text{ or} \\ (2B)\ s_{other} + D_{other} + \dfrac{|X_{other} + (c_{other} - c_{next})\Delta X - X_{next}|}{V_x} \leq s_{next} \end{cases}$$

If crane c_{other} is to the left of c_{next} then it can force c_{next} out of its feasibility cone by being too far to the right. Consider the last task τ_{other} completed by crane c_{other}. If such a task does not exist then c_{other} has no path defined and can not interfere. Otherwise, if the first subconstraint (1A) is satisfied then τ_{other} is to the left of τ_{next} on the x-axis with sufficient space to satisfy trajectory constraint (4), so c_{other} performing τ_{other} does not interfere with the ability of c_{next} to perform τ_{next}. Since we are defining paths for tasks in chronological order we know $s_{other} \leq s_{next}$, so the only remaining possibility is that subconstraint (1B) is satisfied. In this case, (1B) ensures that τ_{next} was scheduled far enough after τ_{other} that crane c_{next} has time to move from the right side of c_{next} and to its task before the start time s_{next}.

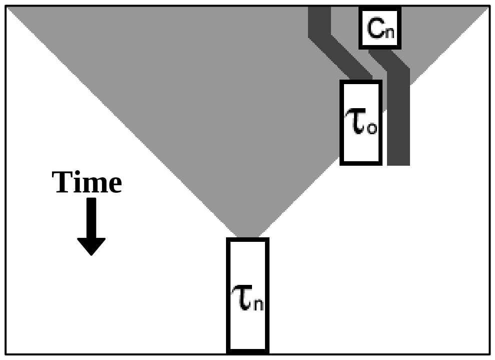


Figure 2.3.1c. Subconstraint (1B) is broken if crane c_{next} is forced out of its feasibility cone by crane c_{other}.

We can draw a similar conclusion that any crane c_{other} to the right of c_{next} can not force c_{next} out of its feasibility cone using subconstraints (2A) and (2B). Thus no other cranes can prevent c_{next} from reaching its next task τ_{next}.

Another reason that a feasible path would not exist is if crane c_{next} had a previously-scheduled task τ_{last} that required it to be positioned outside of its feasibility cone. The path of the crane would be defined up to the completion of τ_{last} because we are defining paths for tasks in chronological order. We can use the self-movement time-gap constraint to show that the space-time position of τ_{last} is inside the feasibility cone.

Self-movement time-gap constraint:

$$\mathrm{s_{last}} + \mathrm{D_{last}} + \max\left(\frac{|\mathrm{X_{last}} - \mathrm{X_{next}}|}{\mathrm{V_x}}, \frac{|\mathrm{Y_{last}} - \mathrm{Y_{next}}|}{\mathrm{V_y}}\right) \leq \mathrm{s_{next}}$$

The constraint ensures that task τ_{next} has been scheduled long enough after τ_{last} that the crane has time to move to τ_{next} in both the x- and y-dimensions before it starts.

Alternatively, if there are no previously-scheduled tasks on crane $\mathrm{c_{next}}$ then the path of $\mathrm{c_{next}}$ is completely undefined and we can start it anywhere along the top of the feasibility cone where $t = 0$. We know there exists a feasible starting point since we just showed that no other crane or cranes can force $\mathrm{c_{next}}$ to not be in the cone.

To review, the crane begins its path inside the feasible cone, remains inside the cone while performing all its previously-scheduled tasks, is inside the cone at its last-defined path position, and can not be cut off or forced out of the cone by another crane. We conclude therefore that a feasible path can be constructed for the crane c_{next} to reach its next task τ_{next} at time s_{next} and complete it at time $s_{next} + D_{next}$. The lemma then holds by induction. □

To further clarify the claim that a feasible path must exist, we demonstrate a rule for how to construct such a path. We call it the *lazy path rule,* because it dictates that cranes remain at rest unless forced to move by another crane or by reaching the edge of its feasibility cone. It is easy to see in Figure 2.3.1d that there is always a path available for c_{next} since cranes to its left and right can not move any faster than itself and they always leave sufficient space for it to pass through.

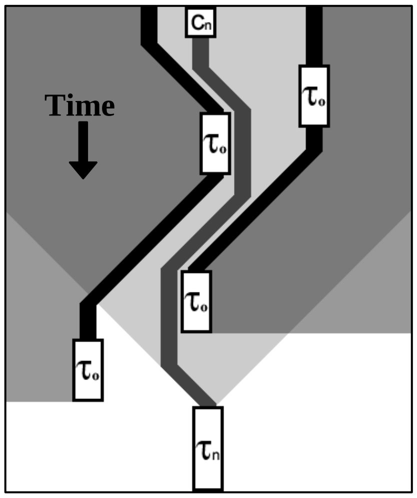
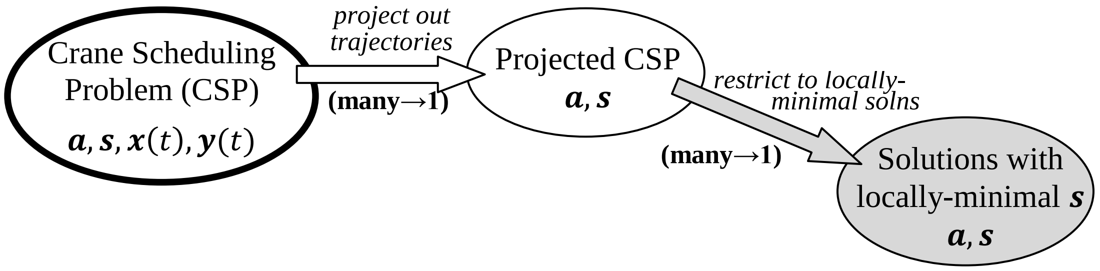


Figure 2.3.1d. Lazy cranes move only when forced and as late as possible.

2.3.2 Local Minimization

The second problem reduction is to restrict our feasible $(\boldsymbol{a}, \boldsymbol{s})$ space to solutions with *locally-minimal task start times*. We define a locally-minimal solution $(\boldsymbol{a}, \boldsymbol{s})$ as one in which no element $s_{\boldsymbol{\tau}}$ of the start time vector $\boldsymbol{s}$ can be individually decreased without breaking feasibility or changing other solution elements. This is an easy restriction to follow, since reducing individual start times whenever feasible will reduce the objective value if the task priority is positive or have no effect if the priority is zero.



Crane Scheduling Problem (CSP)
$\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{x}(t), \boldsymbol{y}(t)$
project out trajectories
(many→1)
Projected CSP
$\boldsymbol{a}, \boldsymbol{s}$
restrict to locally-minimal solns
(many→1)
Solutions with locally-minimal $\boldsymbol{s}$
$\boldsymbol{a}, \boldsymbol{s}$

Lemma 2.3.2. If the crane scheduling problem is feasible then there exists at least one optimal solution with locally-minimal start times.

Proof: If all task priorities are positive, then all optimal solutions will be locally-minimal, since reducing any start time would otherwise improve the objective. If there exist tasks with zero priority, then any optimal solution can be made locally-minimal by reducing the start times of those tasks to their lower bounds. □

2.3.3 Sequencing of Tasks

The final problem reduction is to replace the start time vector $\boldsymbol{s}$ decision variables with a task sequence vector $\boldsymbol{\sigma}$. The motivation for this change is that once the global task sequence is fixed, the optimal task start times for that sequence can be found by greedily choosing the earliest start time for each task in the sequence order. The task sequence permits a straightforward way to construct a solution chronologically from start to finish.

The use of task sequence vectors as decision variables requires the following lemma.

Lemma 2.3.3. For any solution $(\boldsymbol{a}, \boldsymbol{s})$ to the projected crane scheduling problem with locally-minimal task start times $\boldsymbol{s}$, there exists a task sequence $\boldsymbol{\sigma}$ such that greedily selecting the earliest feasible start time for each task in the order $\boldsymbol{\sigma}$ yields the vector of start times $\boldsymbol{s}$.

Proof: Let $\boldsymbol{a}$ be a vector of task assignments to cranes and $\boldsymbol{s}$ be a feasible vector of locally-minimal task start times (as defined in section 2.3.2) for the assignment $\boldsymbol{a}$. Let $\boldsymbol{\sigma}$ be an ordering of the task indices such that $s_{\sigma(i)} \leq s_{\sigma(i+1)}\ \forall\ 1 \leq i \leq T-1$. Let $\boldsymbol{g}$ be the vector of task start times generated by greedily selecting the earliest feasible start time for each task in the order $\boldsymbol{\sigma}$. We will prove by induction that $\boldsymbol{s} = \boldsymbol{g}$.

Suppose we have greedily selected a start time $g_{\sigma(i)}$ for each of the first k tasks in $\boldsymbol{\sigma}$ where $0 \leq k \leq |\boldsymbol{T}|$, and have found that $s_{\sigma(i)} = g_{\sigma(i)}\ \forall i \in [1, k]$. We now select a start time $g_{\sigma(k+1)}$ for task $\sigma(k+1)$. We know that $s_{\sigma(k+1)}$ is a feasible value of $g_{\sigma(k+1)}$ since it is feasible in the complete solution $(\boldsymbol{a}, \boldsymbol{s})$. We can also show by contradiction that no lower value of $g_{\sigma(k+1)}$ is feasible.

Suppose there exists a time $t < s_{\sigma(k+1)}$ that is also a feasible value of $g_{\sigma(k+1)}$. Then we can construct another complete solution $(\boldsymbol{a}, \boldsymbol{s}')$ where $s'_{\sigma(i)} = s_{\sigma(i)}\ \forall i \neq k+1$ and $s'_{\sigma(k+1)} = t$.

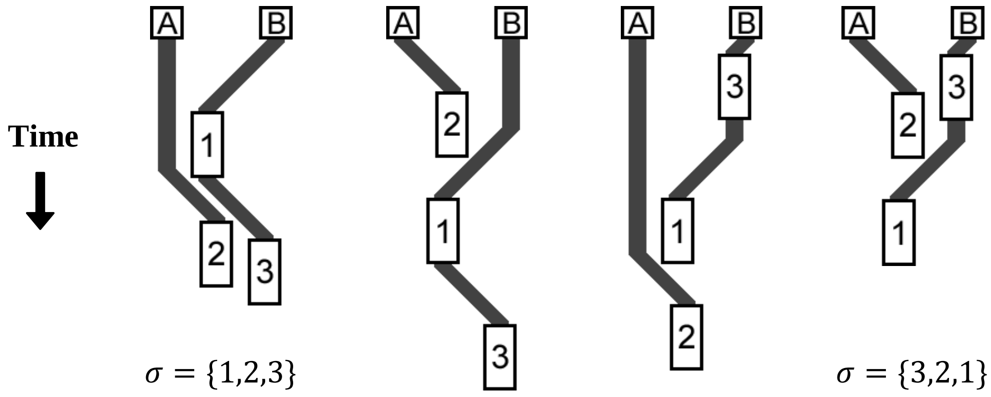
$\boldsymbol{s}$:	$s_{\sigma(1)}$	$s_{\sigma(2)}$	$\cdots$	$s_{\sigma(k)}$	$s_{\sigma(k+1)}$	$s_{\sigma(k+2)}$	$\cdots$	$s_{\sigma(\lvert\boldsymbol{T}\rvert-1)}$	$s_{\sigma(\lvert\boldsymbol{T}\rvert)}$
$\boldsymbol{s}'$:	$s_{\sigma(1)}$	$s_{\sigma(2)}$	$\cdots$	$s_{\sigma(k)}$	t	$s_{\sigma(k+2)}$	$\cdots$	$s_{\sigma(\lvert\boldsymbol{T}\rvert-1)}$	$s_{\sigma(\lvert\boldsymbol{T}\rvert)}$
$\boldsymbol{g}$:	$s_{\sigma(1)}$	$s_{\sigma(2)}$	$\cdots$	$s_{\sigma(k)}$	?	-	$\cdots$	-	-

In the projected crane scheduling formulation of section 2.3.1, all constraints involving task start times are of the form $A \leq s_x \leq B$ or $s_x + C \leq s_y$ where C is a nonnegative number. This means each task start time is bounded from below by earlier-scheduled tasks and from above by later-scheduled tasks. If t is a feasible value of $g_{\sigma(k+1)}$ then assigning $s'_{\sigma(k+1)} = t$ can not break any of its lower bounds since they are the same as in $\boldsymbol{g}$. We also know $s'_{\sigma(k+1)} = t$ does not break any upper bounds since $t < s_{\sigma(k+1)}$ and $s_{\sigma(k+1)}$ is feasible in $(\boldsymbol{a}, \boldsymbol{s})$. Therefore the solution $(\boldsymbol{a}, \boldsymbol{s}')$ is feasible, and since it equates to $(\boldsymbol{a}, \boldsymbol{s})$ with a single element of $\boldsymbol{s}$ decreased, it contradicts the local-minimality of $(\boldsymbol{a}, \boldsymbol{s})$.

So $\boldsymbol{s} = \boldsymbol{g}$ by induction. □

Since each task sequence $\boldsymbol{\sigma}$ generates one vector $\boldsymbol{s}$ of task start times, then there exist only a finite number of locally-minimal solutions to the projected crane scheduling problem, and that they can be enumerated by task sequences. However, it is likely that multiple task sequences will generate the same vector of task start times. The most obvious case of degenerate task sequences is when two tasks start at the same time and they may be greedily scheduled in either order. But this is only a special case of a more general cause for degeneracy.

Greedily scheduling earliest start times for tasks in the sequence $\boldsymbol{\sigma}$ does not imply that the resulting start times will have the ordering $\boldsymbol{\sigma}$. Consider the following solutions to a 3-task crane scheduling problem in which crane A is assigned to task 2 and crane B is assigned to tasks 1 and 3 (i.e. $\boldsymbol{a} = \{B, A, B\}$).



$\sigma = \{1,2,3\}$ $\sigma = \{3,2,1\}$

$\sigma = \{1,3,2\}$ $\sigma = \{2,1,3\}$ $\sigma = \{3,1,2\}$ $\sigma = \{2,3,1\}$

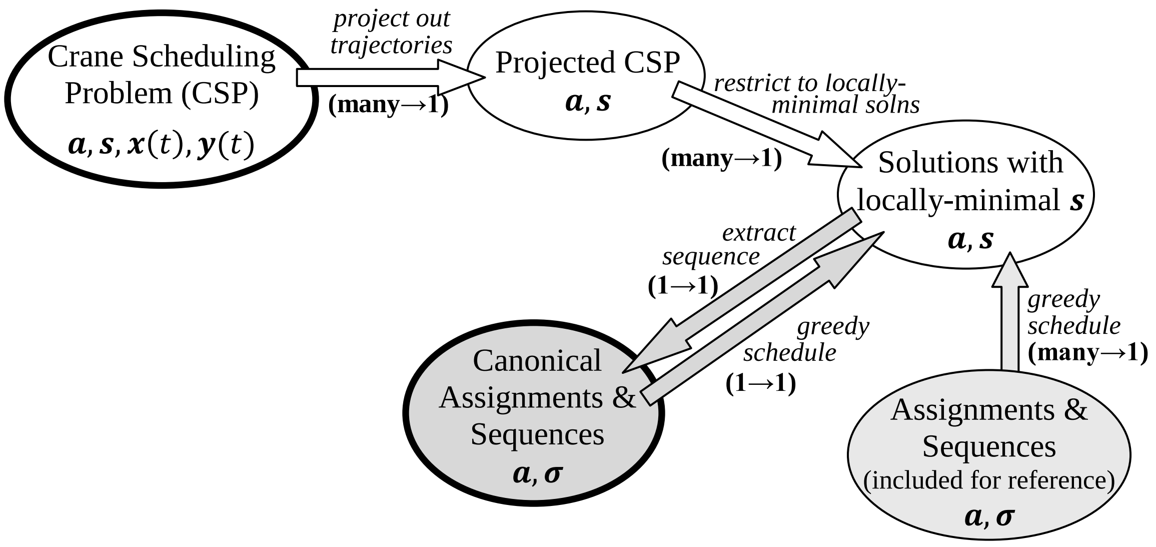
(A) (B) (C) (D)

If task start times are greedily selected in the order $\boldsymbol{\sigma} = \{1,2,3\}$ then solution A will result. The same solution will also result if $\boldsymbol{\sigma} = \{1,3,2\}$. Solution D is generated by either $\boldsymbol{\sigma} = \{3,1,2\}$ or $\boldsymbol{\sigma} = \{3,2,1\}$. In fact, whenever the assignments of two cranes to two tasks do not spatially interfere (tasks 2 and 3 in the figure), the order of those two tasks in the task sequence does not matter (though their order relative to other tasks still does).

The task sequence effectively holds two pieces of information: it specifies the order in which each crane performs its assigned tasks, and it specifies which crane should yield when task assignments interfere. When assignments do not interfere, the task sequence overspecifies the solution, and sequences with different orderings of non-interfering tasks result in the same start times. For this reason we define a *canonical task sequence* $\boldsymbol{\sigma}^*$ for every solution $(\boldsymbol{a}, \boldsymbol{s})$ to the crane scheduling problem.

The canonical task sequence $\boldsymbol{\sigma}^*$ for a locally-minimal solution $(\boldsymbol{a}, \boldsymbol{s})$ is that which generates the solution $(\boldsymbol{a}, \boldsymbol{s})$ using greedy scheduling and has the properties that $s_{\sigma^*(i)} \leq s_{\sigma^*(i+1)}$ $\forall 1 \leq i \leq T-1$ and $s_{\sigma^*(i)} = s_{\sigma^*(i+1)} \Rightarrow a_{\sigma^*(i)} < a_{\sigma^*(i+1)}$. In other words, $\boldsymbol{\sigma}^*$ is the actual order in which greedy scheduling will order the tasks with ties broken by crane index.

It is difficult to determine whether a particular task sequence is canonical without computing its greedy schedule and checking that its start times are properly ordered. However, since our algorithm constructs solutions by adding one task at a time to a partial task sequence, the sequence can be identified as non-canonical as soon as a greedily-scheduled task is assigned a start time that is out of sequence.


Crane Scheduling Problem (CSP)
$\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{x}(t), \boldsymbol{y}(t)$
project out trajectories
(many→1)
Projected CSP
$\boldsymbol{a}, \boldsymbol{s}$
restrict to locally-minimal solns
(many→1)
Solutions with locally-minimal $\boldsymbol{s}$
$\boldsymbol{a}, \boldsymbol{s}$
extract sequence
(1→1)
greedy schedule
(1→1)
greedy schedule
(many→1)
Canonical Assignments & Sequences
$\boldsymbol{a}, \boldsymbol{\sigma}$
Assignments & Sequences
(included for reference)
$\boldsymbol{a}, \boldsymbol{\sigma}$

2.4 The Algorithm

Section 2.3.1 presented a formulation for the projected crane scheduling problem using crane assignments ($\boldsymbol{a}$) and task start times ($\boldsymbol{s}$) as decision variables. This formulation can easily be translated into an integer program, for example by adding binary variables that activate or deactivate the time-gap constraints and bounds to ensure that one constraint from each time-gap set is satisfied. Unfortunately, this IP formulation is difficult to solve with standard branch-and-bound algorithms using linear programming relaxations. The LP relaxations of the subproblems are very weak since the ability to partially assign cranes to tasks or partially satisfy time-gap constraints results in unrealistically-early start times for nearly every task. Objective values for solutions to relaxations are far better than any integer feasible solution, and the bounds that they generate are nearly useless.

To alleviate this problem we employ a dominance relation to compare branches of the solution tree and identify infeasible, suboptimal or redundant branches that can be excluded from consideration without loss of optimality. The dominance relation depends heavily on our formulation of the problem as an assignment and sequencing of tasks as described in section 2.3.3.

Our algorithm consists of two main operations: *crane simulations* and *state pruning.* Crane simulation is a method of building solutions by assigning values to decision variables in chronological order. It assigns and sequences tasks on cranes as they become available in time according to their time windows and precedence conditions. The simulation branches on each of these decision variables, generating a tree of partially-fixed solutions.

As the simulation progresses the solution tree will grow. For this reason we frequently compare partial solutions in the tree using the dominance relation to determine if any can be excluded from the search. The dominance relation examines the *state* of the crane system at specific points in time based on the decision variables that have been fixed in the solution so far.

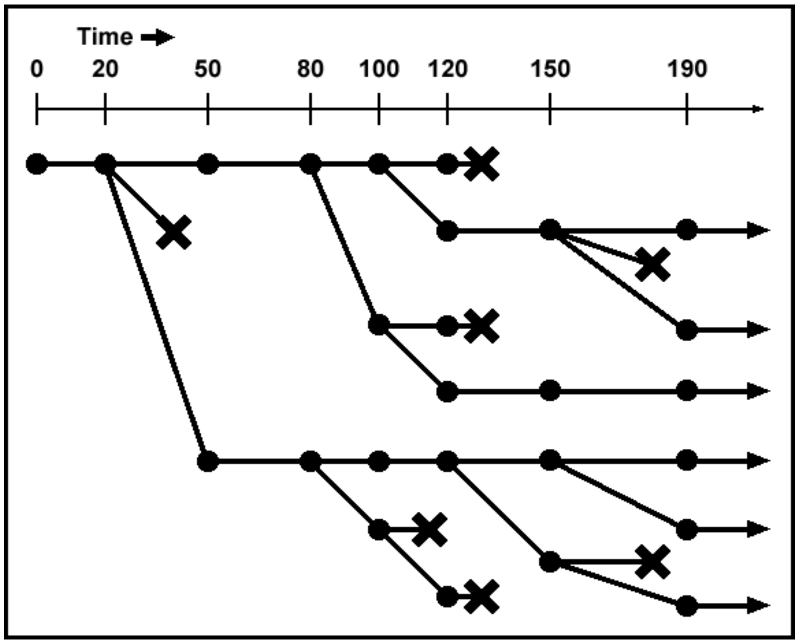


Figure 2.4. As time progresses in the crane simulation, states may divide at each decision point. State pruning eliminates many of these decisions after they are made.

2.4.1 State Definition

Our crane scheduling algorithm operates on an array of solution states. A state represents a branch or a node in the decision tree where certain decision variable values have been fixed and others are yet to be determined. A state encapsulates a partial solution to the problem in which a subset of the tasks has been assigned and sequenced.

The variables stored in each state contain a task sequence $\boldsymbol{\sigma}$**,** their assigned cranes $\boldsymbol{a_\sigma}$, and their start times $\boldsymbol{s_\sigma}$. Start times in this case are not decision variables, but merely store the greedy start times computed as tasks are sequenced. Each state also holds a *nogood* list of crane-task pairs that have been chosen not to be placed in the next sequence position. This ensures that if a pair is chosen in one branch of the solution tree, then it can not be chosen in the other. If we had used binary variables $x_{c\tau}^{i}$ to represent whether the crane c performs task τ as the ith sequenced task, placing an item in the nogood list would be equivalent to fixing $x_{c\tau}^{i} = 0$. Finally, each state has a clock that maintains a lower bound on the time that the next-sequenced task can start, a value which is implied by $\boldsymbol{\sigma}$, $\boldsymbol{a_\sigma}$ and the nogood list.

Example State (10-task instance)										Clock = t
$\boldsymbol{\sigma} =$	$\sigma(1)$	$\sigma(2)$	$\sigma(3)$	$\sigma(4)$	-	-	-	-	-	-
$\boldsymbol{a_\sigma} =$	$a_{\sigma(1)}$	$a_{\sigma(2)}$	$a_{\sigma(3)}$	$a_{\sigma(4)}$	-	-	-	-	-	-
$\boldsymbol{s_\sigma} =$	$s_{\sigma(1)}$	$s_{\sigma(2)}$	$s_{\sigma(3)}$	$s_{\sigma(4)}$	-	-	-	-	-	-
Nogoods:	$\neg\left(\sigma(5) = \tau_a \wedge a_{\sigma(5)} = c_a\right), \neg\left(\sigma(5) = \tau_b \wedge a_{\sigma(5)} = c_b\right), \cdots$									

Vectors $\boldsymbol{\sigma}$, $\boldsymbol{a_\sigma}$ and the nogood list represent explicit decisions. Vector $\boldsymbol{s_\sigma}$ and the clock are calculated values derived from those decisions.

2.4.2 Crane Simulation

We initialize our algorithm with a single state in which no tasks have been assigned, all state variables are empty, and the clock is set to zero. We then iterate the following procedure on each state.

Crane Simulation Procedure

1) Check all crane-task pairs to identify those which are feasible for the next task sequence position at the current clock time. Such pairs must satisfy the following conditions.

a) The crane is not busy.
(All its previously-assigned tasks are complete at the current clock time.)
(Otherwise, we can postpone the assignment of tasks to this crane.)

b) The clock is within the task time window.
(Otherwise, we can postpone the sequencing of the task.)

c) The task has all its supplementary task conditions satisfied.
(e.g. any prerequisite tasks must be complete at the current clock time.)
(Otherwise, we can postpone the sequencing of the task.)

2) For each crane-task pair (c, τ) that is feasible, we compute its earliest start time s_τ. If $s_\tau < s_{last}$ or $(s_\tau = s_{last} \wedge c < a_{last})$, where a_{last} and s_{last} are the assignment and start time of the last sequenced task, then sequencing (c, τ) as the next task would make $\boldsymbol{\sigma}$ a non-canonical sequence as described in section 2.3.3. We therefore add (c, τ) to the nogood list.

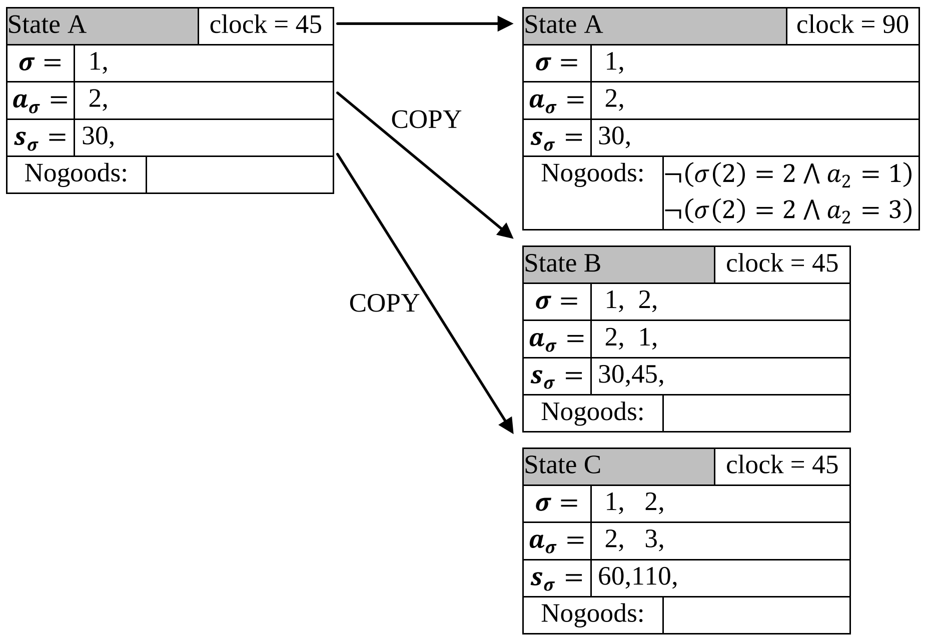
3) For each crane-task pair (c, τ) that is feasible and not in the nogood list we make a copy of the state. In the copy, we sequence τ as the next task and assign c as its crane. We also erase its nogood list since its information is now obsolete. In the original state, we add $(a_{next} = c \wedge \sigma_{next} = \tau)$ to the nogood list.

Example 2.4.2: Crane Simulation
In state A, task 1 has been assigned to the 2nd of 3 cranes, starting at time 30 (seconds). When the state clock reaches time 30, a second task to the left of task 1 is released. The state is copied onto state B in which crane 1 is assigned task 2, which can be performed simultaneously and starts at time 45. Another copy is made onto state C in which crane 3 is assigned task 2, but it must wait until time 90 for the other task to complete before starting at time 110 (it waits at position $(x = 30, y = 0)$). In the original state, both assignments are added to the nogood list so they can not be used in the next sequence position of state A.

Parameters:

Task	Release	Duration	X	Y
1	30 s	60 s	20 m	0 m
2	45 s	60 s	10 m	0 m

$\Delta X = 10\ m$
$V_x = 1\ m/s$
$V_y = 1\ m/s$



State A	clock = 45
$\boldsymbol{\sigma} =$	1,
$\boldsymbol{a_\sigma} =$	2,
$\boldsymbol{s_\sigma} =$	30,
Nogoods:	

State A	clock = 90
$\boldsymbol{\sigma} =$	1,
$\boldsymbol{a_\sigma} =$	2,
$\boldsymbol{s_\sigma} =$	30,
Nogoods:	$\neg(\sigma(2) = 2 \wedge a_2 = 1)$
	$\neg(\sigma(2) = 2 \wedge a_2 = 3)$

State B	clock = 45
$\boldsymbol{\sigma} =$	1, 2,
$\boldsymbol{a_\sigma} =$	2, 1,
$\boldsymbol{s_\sigma} =$	30,45,
Nogoods:	

State C	clock = 45
$\boldsymbol{\sigma} =$	1, 2,
$\boldsymbol{a_\sigma} =$	2, 3,
$\boldsymbol{s_\sigma} =$	60,110,
Nogoods:	

4) Once copies of the state have been made for all feasible crane-task pairs, all such pairs are in the nogood list of the original state. The current clock time must then be updated to a new lower bound. The clock is set to the next time that a task will be feasible, which is the earliest of (1) the next task release time, (2) the next task completion time (which may satisfy a prerequisite condition), and (3) the previous clock time if another task became feasible simultaneously.

We repeat steps one through four above on every state until there are no more tasks to be released or completed. At that point every state either contains a complete solution to the problem or it is infeasible. A state may be infeasible because it failed to schedule a task before its deadline or because it added every possible crane-task pair to its nogood list. The latter case will inevitably occur in some states but are a necessary part of the algorithm. States must be allowed to delay the sequencing of their next task without limit to ensure that every solution is enumerated.

Lemma 2.4.2. For any feasible task assignment $\boldsymbol{a}$ and sequence $\boldsymbol{\sigma}$ that greedily generates a task start time vector $\boldsymbol{s}$ with canonical ordering $\boldsymbol{\sigma}$, the crane simulation procedure generates a state containing the solution $(\boldsymbol{a}, \boldsymbol{\sigma}, \boldsymbol{s})$.

Proof: For any such triple $(\boldsymbol{a}, \boldsymbol{\sigma}, \boldsymbol{s})$, we will inductively prove that the steps of the crane simulation procedure outlined above generate a state containing the solution $(\boldsymbol{a}, \boldsymbol{\sigma}, \boldsymbol{s})$. For the base case let state S_0 be the initialized state in which no tasks have been assigned or sequenced, the nogood list empty and the clock time is zero.

Let state S be a state generated by the crane simulation procedure in which task $\sigma(i)$ has been assigned for all $0 \leq i \leq k$ in the order $\boldsymbol{\sigma}$ with assignments and start times consistent with $\boldsymbol{a}$ and $\boldsymbol{s}$. Consider state S immediately after task $\sigma(k)$ is sequenced (or at time zero if $k = 0$). The state clock at this time will be no later than $s_{\sigma(k)}$, the start time of task $\sigma(k)$, since the clock is a lower bound on $s_{\sigma(k)}$. The crane simulation will then execute the four steps on this new state.

Let $c_{k+1} = a_{\sigma(k+1)}$ and $\tau_{k+1} = \sigma(k+1)$. If the crane-task pair (c_{k+1}, τ_{k+1}) is feasible at the current clock time, then a copy of state S will be made in which the pair is sequenced next. In this case, redefine state S as the copy and continue the induction. If the (c_{k+1}, τ_{k+1}) is not yet feasible, it may be because crane c_{k+1} is busy, because task τ_{k+1} is not yet released, or because some prerequisite task is not yet complete. Note that since $(\boldsymbol{a}, \boldsymbol{\sigma}, \boldsymbol{s})$ is feasible, any tasks that are prerequisites of task τ_{k+1} must already be sequenced in state S and awaiting completion. So in any case, (c_{k+1}, τ_{k+1}) will eventually be feasible upon the completion of some task or upon the release of task τ_{k+1}. The clock will update to this time according to step 4 and (c_{k+1}, τ_{k+1}) will then be sequenced in some state copy.

By induction, we conclude that the crane simulation procedure generates a state containing the solution $(\boldsymbol{a}, \boldsymbol{\sigma}, \boldsymbol{s})$. □

2.4.3 State Pruning

Periodically during the crane simulation, states are checked to identify infeasible, suboptimal or redundant states which can be deleted.

To identify infeasible states, each state is checked whenever its clock time passes the deadline of a task. If the task has not been scheduled to complete before its deadline time in the state, its solution of the state is infeasible and the state is deleted.

To identify suboptimal states, we search for pairs of states D and S that satisfy the dominance relation. In such a case we say state D *dominates* state S and state S is deleted. We use the following conditions to identify state domination.

State Dominance Relation

(1) All tasks assigned in state S are also assigned in state D.

(2) The accumulated objective in state S is at least that in state D for all tasks assigned in state D. We define the accumulated objective for a set of tasks T as $\sum_{\tau \in T} P_\tau q_\tau$ where P_τ is the priority of task τ and q_τ is the start time s_τ of task τ if it has been assigned, otherwise t_τ is the clock time of the state, a lower bound on s_τ.

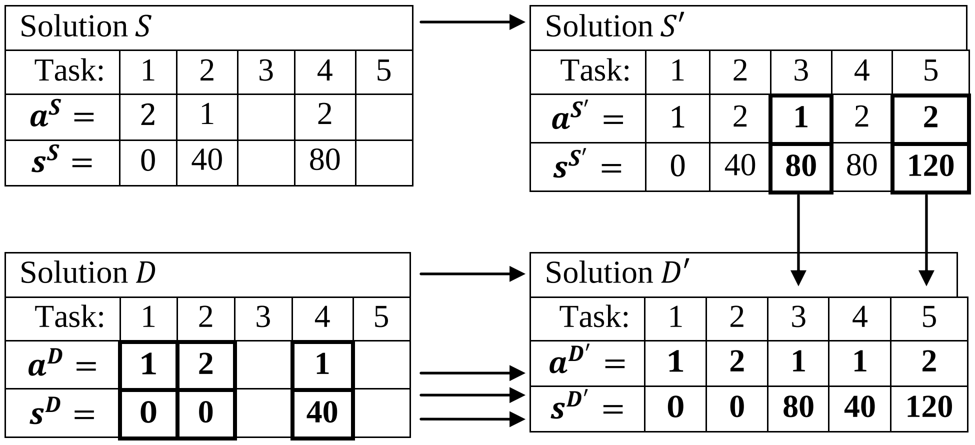
(3) Any task τ which can be sequenced next in state S, assigned to a crane c_s and started at time s_S is either already assigned in state D or can be accepted by the same-indexed crane c_D in state D and started at time $s_D \leq s_S$.

The goal of the dominance relation is to show that the best solution achievable in the solution branch represented by state D is at least as good as that in state S. State S is then either suboptimal or redundant and may be deleted. We claim that deleting state S when the dominance conditions are satisfied will not affect the ability of the algorithm to find an optimal solution.

Lemma 2.4.3. If state S and state D are both active (not deleted or excluded) in the solution tree and state D dominates state S according to the dominance relation, then deleting state S, and thus excluding the solution branch it represents, will not prevent the crane simulation from finding an optimal solution.

Proof: Consider a pair of active states D and S such that state D dominates state S. Let $(\boldsymbol{a^D}, \boldsymbol{s^D})$ and $(\boldsymbol{a^S}, \boldsymbol{s^S})$ be the vectors of task assignments and start times defined so far in states D and S respectively. Let $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$ be a solution to the *projected* crane scheduling problem (section 2.3.1) with the minimum objective value that is also an *extension* of $(\boldsymbol{a^S}, \boldsymbol{s^S})$, such that $a_\tau^{S'} = a_\tau^S$ and $s_\tau^{S'} = s_\tau^S$ for all tasks τ assigned so far in state S. That is, $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$ has the best objective of any solution in the branch represented by state S.

Consider the solution $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ that is an extension of $(\boldsymbol{a^D}, \boldsymbol{s^D})$ in which $a_\tau^{D'} = a_\tau^D$ and $s_\tau^{D'} = s_\tau^D$ for all tasks τ assigned so far in state D, and $a_\tau^{D'} = a_\tau^{S'}$ and $s_\tau^{D'} = s_\tau^{S'}$ for all tasks τ *not* assigned in state D. In other words, $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ contains all the assignments and start times defined in $(\boldsymbol{a^D}, \boldsymbol{s^D})$, plus elements of $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$ filled in for the remaining tasks. We claim that $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ is feasible and has a lower total objective than $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$.



Solution S

Task:	1	2	3	4	5
$\boldsymbol{a^S} =$	2	1		2	
$\boldsymbol{s^S} =$	0	40		80	

Solution S'

Task:	1	2	3	4	5
$\boldsymbol{a^{S'}} =$	1	2	**1**	2	**2**
$\boldsymbol{s^{S'}} =$	0	40	**80**	80	**120**

Solution D

Task:	1	2	3	4	5
$\boldsymbol{a^D} =$	**1**	**2**		**1**	
$\boldsymbol{s^D} =$	**0**	**0**		**40**	

Solution D'

Task:	1	2	3	4	5
$\boldsymbol{a^{D'}} =$	**1**	**2**	**1**	**1**	**2**
$\boldsymbol{s^{D'}} =$	**0**	**0**	**80**	**40**	**120**

Figure 2.4.3. The hypothetical solution $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ is constructed by copying the data from state D and filling in the gaps with data from the hypothetical solution $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$.

To show that $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ is feasible, consider the addition of elements of $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$ to $(\boldsymbol{a^{D}}, \boldsymbol{s^{D}})$ one task at a time in chronological order. For the first task τ that is appended to the schedule of a crane c in state D, dominance relation condition (3) says that task τ can be started at least as soon it can be on crane c in state S, so this additional assignment and start time is feasible. The assignment of task τ to crane c in state D places crane c in the same place for the same time in both solutions. So any subsequent movement of crane c that is feasible in state S is also feasible in state D. So condition (3) *still holds* for all remaining tasks. Repeating this argument, we conclude by induction that $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ is feasible.

To show that the objective value of $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ is lower than that of $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$, consider the following breakdown of the two complete objective values. Let t_S be the clock time in state S, let $\boldsymbol{A^S}$ be the set of tasks assigned so far in state S and let $\boldsymbol{A^D}$ be the same for state D. Note that $\boldsymbol{A^S} \subseteq \boldsymbol{A^D} \subseteq \boldsymbol{T}$ and all terms are nonnegative.

$$\sum_{\tau \in \boldsymbol{T}} \mathrm{P}_\tau \mathrm{s}_\tau^{\mathrm{S}'} = \underbrace{\sum_{\tau \in \mathbf{A^S}} \mathrm{P}_\tau (\mathrm{s}_\tau^{\mathrm{S}'} - \mathrm{R}_\tau) + \sum_{\tau \in \boldsymbol{A^D} \setminus \mathbf{A^S}} \mathrm{P}_\tau (\mathrm{t_S} - \mathrm{R}_\tau)}_{\text{accumulated objective for } \mathbf{A^D}} + \underbrace{\sum_{\tau \in \mathbf{T} \setminus \mathbf{A^D}} \mathrm{P}_\tau (\mathrm{s}_\tau^{\mathrm{S}'} - \mathrm{R}_\tau)}_{\text{remaining objective}}$$

$$\sum_{\tau \in \boldsymbol{T}} \mathrm{P}_\tau \mathrm{s}_\tau^{\mathrm{D}'} = \overbrace{\sum_{\tau \in \mathbf{A^D}} \mathrm{P}_\tau (\mathrm{s}_\tau^{\mathrm{D}'} - \mathrm{R}_\tau)} + \overbrace{\sum_{\tau \in \mathbf{T} \setminus \mathbf{A^D}} \mathrm{P}_\tau (\mathrm{s}_\tau^{\mathrm{D}'} - \mathrm{R}_\tau)}$$

The first two terms are the accumulated objective value of each state for the set $\mathbf{A^D}$ as defined in state-domination condition (2). We already know that this value in state S is at least that in state D. The last two terms are the remaining objective from tasks unassigned in both states. Since $\mathrm{s}_\tau^{\mathrm{D}'} = \mathrm{s}_\tau^{\mathrm{S}'}$ for all $\tau \in \mathbf{T} \setminus \mathbf{A^D}$, these two terms are equal. Therefore the total objective of the solution $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$ is at least that of $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$.

The solution $(\boldsymbol{a^{D'}}, \boldsymbol{s^{D'}})$ is therefore feasible and its objective value is at most that of the best solution $(\boldsymbol{a^{S'}}, \boldsymbol{s^{S'}})$ in the branch represented by state S. We conclude then that state S leads to an optimal solution only if state D does as well, so state S is unnecessary and we need not search its solution branch. □

2.4.4 State Chopping Heuristic

If the state domination procedure is not sufficiently effective during the execution of the algorithm, we may end up with too many states to hold in memory. If the number of states exceeds a certain user-defined threshold, we apply a heuristic we call the *state chop* that deletes states without proving that they are dominated. It is a less precise means of removing states that allows us to still generate a good solution to the problem, although its optimality is no longer guaranteed.

The chopping heuristic sorts the states according to their number of tasks completed (high to low), followed by their accumulated objective value (low to high), and deletes states from the end of the list until it is sufficiently short. Although this voids the optimality guarantee, the resulting solutions tend to be very good and often even optimal in practice.

2.5 Experimental Results

The algorithm is currently written in about 2,000 lines of C++ code. We also wrote a short program in *Mathematica 7.0* to generate sample problems on which to test the algorithm. The program generates tasks of random lengths and priorities, groups them into random job sequences, and assigns a few random precedence constraints. It then constructs a heuristic solution by assigning and sequencing tasks on cranes and greedily computing minimal start times. The start times are used as the release times for the tasks, so that the optimal solution to the problem is exactly 0. A deadline is chosen randomly after the finish time.

Experiments with the algorithm have demonstrated excellent performance. It consistently solves problems with two cranes and 100 tasks to optimality in around 10 seconds, with runtimes that vary almost linearly with the number of tasks. For problems with more than two cranes, the runtime depends significantly on the maximum size allocated to the state list, since the limit will often be reached. Allowing more states to be processed means longer total processing times.

The greatest weakness of the algorithm is likely the manner in which it assigns multiple cranes to multiple tasks near the same clock time: it generates a state for each possible combination of crane-task assignments and sequences for those assignments. Especially in problems with more than two cranes present, the number of such combinations can be quite large, multiplying the number of states substantially.

The state-domination pruning procedure can eliminate many of these states, but additional pruning by the heuristic is often necessary. Although heuristic pruning eliminates the guarantee of optimality, we have observed that it often preserves an optimal solution anyway. In cases that require a large number of states to guarantee optimality, the same optimal solution is usually returned even when the number of states is limited to 5% or 10% of the required number. Typical settings of at most 30,000 states total, with at most 10,000 states after pruning, often generate optimal solutions with runtimes on the order of minutes.

Another issue with the algorithm is that of infeasible schedules. The feasibility of a problem instance can be difficult to identify before running the algorithm, so we incorporated some flexibility to handle infeasibility. Whenever there does not exist a state that satisfies a task deadline, we allow the deadline to be repeatedly relaxed by a fixed interval until a state is found that has completed the task (the infeasibility is reported at completion). This technique is only effective for minor infeasibility problems, and issues arise when multiple overdue tasks are simultaneously active. If the algorithm cannot handle the extent of infeasibility, it will abort and display a list of jobs for which it could not meet deadlines.

2.6 Conclusions

The algorithm we have presented solves the crane scheduling problem by using a simulation to create a decision tree of possible states of the system and pruning to minimize the number of decision states that need to be explored. In problem instances with relatively few cranes, the number of simulation states remains small and the algorithm typically returns optimal solutions in less than a minute. In more complex cases, the runtime and solution quality are less certain, but we find them to be quite satisfactory in our experiments.

We believe there are still improvements to be made to our algorithm, particularly in our state-domination pruning procedure. When the state list is large, the time required to compare state pairs can become tedious. We hypothesize that further experimentation with the pruning procedure would be the greatest opportunity for improvement to the algorithm efficiency.

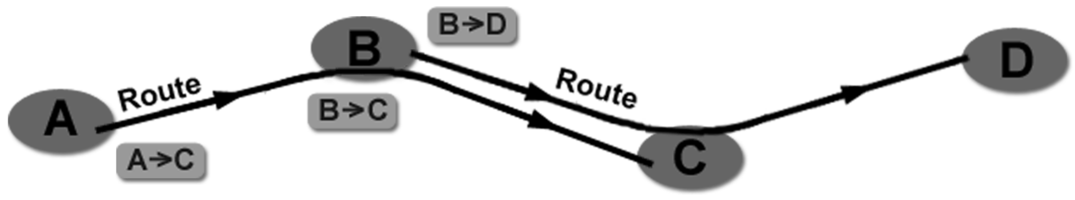
Chapter 3: A Benders Approach to a Transportation Scheduling Problem

3.1 Introduction

Benders decomposition is an iterative technique for mathematical programming in which a problem is decomposed into two parts: a master problem which solves for a subset of the variables, and a subproblem which solves for the remaining variables while treating the master variables as fixed. The technique uses information from the solution of the subproblem to design a constraint that is necessary for an optimal solution but unsatisfied by the current master solution. The master problem and subproblem are solved iteratively and new constraints are added to the master problem until no such constraint exists. At this point, the last master solution is optimal.

In traditional Benders decomposition (Benders 1962), the subproblem is a linear program, and a constraint is derived from its dual solution. More recently, logic-based (Geoffrion 1972) (Hooker, Integrated Methods for Optimization 2006) or combinatorial Benders techniques (Codato and Fischetti 2006) have been developed that generate constraints based on infeasibility or suboptimality. By not requiring a linear subproblem, logic-based Benders approaches are an excellent way to integrate disparate optimization methods. Logic-based Benders techniques have been successfully applied to other network optimization problems, including pickup and delivery problems (Cortés, Matamala and Contardo 2010), facility location (Fazel-Zarandi and Beck 2009), and road tolling (Bai and Rubin 2009). They have also been used in other scheduling contexts, such as sports season scheduling (Rasmussen and Trick 2007) and machine scheduling (Hooker, Planning and Scheduling by Logic-Based Benders Decomposition 2007).

The network problem on which we focus is fairly general and encountered frequently in the context of freight and parcel transportation. The problem consists of a set of deliveries which are required to move between pairs of points in a network, and a set of fixed-cost routes that travel along paths through the points. Any delivery required to travel from one point on the path of a route to a later point on the path can be carried by that route. Each delivery has a volume of space that it requires and each route has a maximum space capacity and cost. The objective is to find a minimum-cost set of routes that have capacity to carry all the deliveries.



This problem was studied in (Pajunas, et al. 2007) in the context of contracts for the United States Postal Service. In that work, an integer programming approach was developed, a key to which was adding a redundant constraint that required the total capacity of selected routes to be at least the total volume of the deliveries to be shipped. The addition of this constraint resulted in much faster solution times. It is this insight that inspired the constraints for our Benders approach.

Although we focus on the context of freight transportation, our method is also applicable to other capacitated assignment problems. For example, consider the problem of fixed-cost facility location. Instead of deliveries, we could use a set of geographically scattered customers, each with a required level of service. Instead of routes, we could use a set of potential locations for facilities to serve the customers, each with a service capacity. Our Benders algorithm could be applied to solve the same model used for this problem context as well.

3.2 Problem Description

Our mathematical model of the network optimization problem appeared in (Pajunas, et al. 2007). In this case, a route corresponded to a U.S.P.S. contract for a truck operating at a particular time. Packages could be assigned to routes that meet the service time requirements of the package.

Full Problem:

$$\min \sum_{r \in \mathcal{R}} C_r y_r$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} A_{rd} x_{rd} \geq 1, \forall d \in \mathcal{D}$$

$$\sum_{d \in \mathcal{D}} A_{rd} V_d x_{rd} \leq S_r y_r, \forall r \in \mathcal{R}$$

$$y_r \in \{0,1\}, \forall r \in \mathcal{R}$$

$$x_{rd} \geq 0, \forall d \in \mathcal{D}, r \in \mathcal{R}$$

$$x_{rd} \in \{0,1\}, \forall d \in \mathcal{U}, r \in \mathcal{R}$$

Variables:

y_r	=1 if route r is used, 0 otherwise
x_{rd}	=proportion of delivery d carried by route r

Parameters:

$\mathcal{R}$	=set of routes
$\mathcal{D}$	=set of deliveries
$\mathcal{U}$	=set of unsplittable deliveries ($\mathcal{U} \subseteq \mathcal{D}$)
C_r	=cost of route r
S_r	=capacity of route r
V_d	=volume of delivery d
A_{rd}	=1 if route r can carry delivery d, 0 otherwise

Note that some of the deliveries $d \in \mathcal{D}$ can be divided across multiple routes, while other deliveries $d \in U \subseteq \mathcal{D}$ can only be carried by one route.

3.2.1 Problem Decomposition

We decompose the full problem by assigning the route variables y to the master problem and the delivery assignment variables x to the subproblem. The goal is to find an optimal solution to the master problem that is also feasible in the subproblem using relatively few of the iteratively-generated constraints.

Full Problem:

$$\min \sum_{r\in\mathcal{R}} C_r y_r$$

$$\text{s.t.} \quad \sum_{r\in\mathcal{R}} a_r^i y_r \geq b^i, \forall i \in \mathcal{I}$$

$$y_r \in \{0,1\}, \forall r \in \mathcal{R}$$

where $\mathcal{I}$ is an index set of generated constraints.

Subproblem:

$$\min \; 0$$

$$\text{s.t.} \quad \sum_{r\in\mathcal{R}} A_{rd} x_{rd} \geq 1, \forall d \in \mathcal{D}$$

$$\sum_{d\in\mathcal{D}} A_{rd} V_d x_{rd} \leq S_r \bar{y}_r, \forall r \in \mathcal{R}$$

$$x_{rd} \geq 0, \forall d \in \mathcal{D}, r \in \mathcal{R}$$

$$x_{rd} \in \{0,1\}, \forall d \in \mathcal{U}, r \in \mathcal{R}$$

where $\bar{y}$ is the last computed solution to the master problem.

The master problem begins with only its binary domain constraints and the set $\mathcal{I}$ empty. As a result, the first solution is $\bar{y} = 0$. Each master solution $\bar{y}$ is then checked for feasibility in the subproblem. If it is infeasible, one or more cuts of the form $\sum_{r\in\mathcal{R}} a_r^i y_r \geq b^i$ is added to the master problem. If it is feasible, the procedure stops and the solution $\bar{x}$ is combined with the master solution to form the full solution $(\bar{y}, \bar{x})$.

3.3 Benders Cuts

We first demonstrate Benders cuts that we call volume set cuts, similar to the flow set cuts used in other network flow Benders decomposition methods (Gendron, Crainic and Franfioni 1997). These cuts define a lower bound on the capacity available among a set of routes. The cuts have a natural interpretation: for any subset $\mathcal{Z}$ of the deliveries $\mathcal{D}$, the total capacity of all routes used in the master solution $\bar{y}$ that are capable of carrying a delivery in $\mathcal{Z}$ must be at least the total volume of the deliveries in $\mathcal{Z}$. We introduce two types of cuts that apply this principle: a weak volume set cut and a strong volume set cut. We explain in section Implementation3.4 how we use each type.

We define the weak volume set cuts as follows.

Weak Delivery Set Cut:

Let $\mathcal{Z} \subseteq \mathcal{D}$ be any subset of the deliveries, and let

$$a_r^{\mathcal{Z}} = \begin{cases} S_r, \text{ if route } r \text{ can carry a delivery } d \in \mathcal{Z}, \text{ i.e. } \sum_{d \in \mathcal{Z}} A_{rd} \geq 1 \\ 0, \text{ otherwise, i.e. } \sum_{d \in \mathcal{Z}} A_{rd} = 0 \end{cases}$$

Then all feasible solutions to the full problem satisfy

$$\sum_{r \in \mathcal{R}} a_r^{\mathcal{Z}} y_r \geq \sum_{d \in \mathcal{Z}} V_d$$

Cuts of this form are traditional Benders cuts and can be constructed from linear combinations of constraints in the full problem. Because of this, they can only be used to eliminate master solutions which are infeasible in the linear relaxation of the subproblem, and thus they are of limited use. If the master solution is infeasible in the mixed-integer subproblem but feasible in its relaxation, there will be no cut of the above form that is broken by the master solution. Note that the weak delivery set cut for which $\mathcal{Z} = \mathcal{D}$ corresponds to the constraint used in (Pajunas, et al. 2007).

We also create a stronger version of the cut with reduced coefficients. In the strong version, we do not use the full capacity of each route as the coefficient of each route variable on the lefthand side. Instead, we use the maximum volume of deliveries in $\mathcal{Z}$ that could actually fit in the route as the coefficient. This quantity is no greater, and typically less, than the capacity. If the capacity of the route cannot be fully utilized by the deliveries in $\mathcal{Z}$ because of their volume or integrality constraints, then a smaller coefficient can be used on that route variable.

Strong Delivery Set Cut:

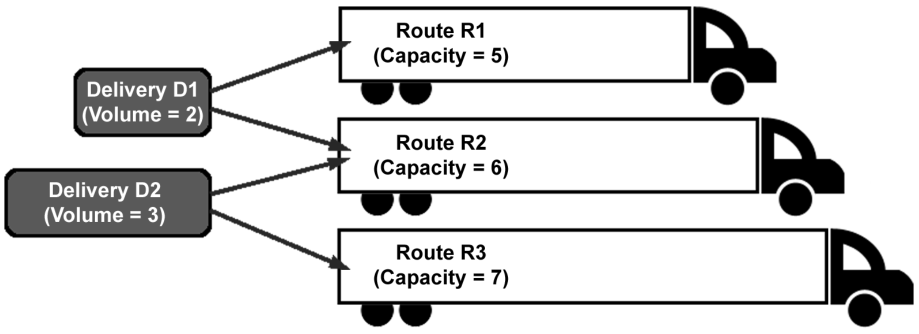
Let $\mathcal{Z} \subseteq \mathcal{D}$ be any subset of the deliveries, and let

$$a_r^{\mathcal{Z}} = max\left\{\sum_{d\in\mathcal{Z}} V_d x_{rd} \left| \begin{array}{l} \sum_{d\in\mathcal{Z}} A_{rd} V_d x_{rd} \le S_r \\ x_{rd} \ge 0, \forall d \in \mathcal{Z} \\ x_{rd} \in \{0,1\}, \forall d \in Z \cap \mathcal{U} \end{array} \right.\right\}$$

Then all feasible solutions to the full problem satisfy

$$\sum_{r\in\mathcal{R}} a_r^{\mathcal{Z}} y_r \ge \sum_{d\in\mathcal{Z}} V_d$$

For example, suppose there exist two deliveries, $D1$ with volume 2 and $D2$ with volume 3, that can be carried only by three available routes, $R1$ with capacity 5, and $R2$ with capacity 6, and $R3$ with capacity 7. Suppose that $D1$ can use only route $R1$ or $R2$, and that $D2$ can use only route $R2$ or $R3$.



Then our Benders cuts would appear in each form as follows.

Weak Delivery Set Cut: $5y_{R1} + 6y_{R2} + 7y_{R3} \ge 5$

Strong Delivery Set Cut: $2y_{R1} + 5y_{R2} + 3y_{R3} \ge 5$

Both constraints are valid for any feasible solution, but the second form is much stronger. It not only cuts off more infeasible fractional solutions than the weak form, but also cuts off more infeasible integer solutions. For example, $(y_{R1} = 1, y_{R2} = 0, y_{R3} = 0)$ is infeasible and allowed by the weak cut but forbidden by the strong cut.

Unlike the weak cuts, the strong delivery set cuts cannot be constructed from linear combinations of the full problem constraints. They are strictly logic-based Benders cuts, and can be used to cut off master problem solutions that are feasible in the subproblem relaxation but not the mixed-integer subproblem.

3.4 Implementation

We initialize our master problem with a single strong volume set cut generated from $\mathcal{Z} = \mathcal{D}$. That is, the total capacity of the chosen routes must be at least the total volume of the deliveries. In each iteration we compute an optimal solution to the master problem. Although previous work on Benders decomposition has found approximate master solutions to be faster (Gabrel, Knippel and Minoux, A Comparison of Heuristics for the Discrete Cost Multicommodity Network Optimization Problem 2003) (Bai and Rubin 2009), we find our master problems are relatively easy to solve and yield better cuts when solved to optimality.

After the master problem is solved in each iteration, we search for set cuts broken by the current solution. Consistent with previous research (Fischetti, Salvagnin and Zanette 2009), we find that generating cuts from the smallest possible infeasible delivery sets are most effective. We employ this goal in two of the following three methods for finding broken cuts.

3.4.1 Identifying Strong Cuts with Enumeration

Our first method to identify broken cuts enumerates delivery sets and tests the validity of their volume set cuts. We generate cuts for all delivery sets of a fixed size that successively increases from 1 to $|D|$. We use a heuristic to determine the amount of time spent enumerating cuts: enumeration continues until either (1) the time spent enumerating cuts is greater than the time spent solving the last master problem, (2) we find as many broken cuts as there are deliveries, or (3) all sets have been checked, whichever occurs first. The decision to identify multiple cuts in each step is one that has proven useful in other Benders contexts as well (Gabrel, Knippel and Minoux, Exact solution of multi-commodity network optimization problems with general step cost functions 1999).

We prefer to use weak volume set cuts in the enumeration procedure because their coefficients can be computed more quickly. However, if no broken weak cuts are found, we repeat the enumeration with strong cuts.

The enumeration method generates many good cuts for the first few iterations, causing the objective values of the master solutions to rise quickly. In some cases, problems are solved entirely by enumeration. For harder problems however, there eventually does not exist a broken cut among delivery sets that are small enough to enumerate. If we fail to find cuts by enumeration in the allotted time, we switch the first optimization method described in the next section.

3.4.2 Identifying Weak Cuts with Optimization

Another advantage of the weak volume set cuts is that they can be found by optimization. If there exists a weak volume cut that is broken by the current master solution then it can be found by solving the following cut-generating mixed-integer linear program.

Weak Cut-Generating MIP:

$$\min \sum_{d\in\mathcal{D}} z_d$$

$$\text{s.t.} \quad \sum_{r\in\mathcal{R}} a_r \bar{y}_r \leq b - \varepsilon$$

$$a_r \geq S_r A_{rd} z_d, \forall r \in \mathcal{R}$$

$$a_r \leq S_r \sum_{d\in\mathcal{D}} A_{rd} z_d \,, \forall r \in \mathcal{R}$$

$$a_r \leq S_r, \forall r \in \mathcal{R}$$

$$b = \sum_{d\in\mathcal{D}} V_d z_d \,, \forall d \in \mathcal{D}$$

$$a_r, b \geq 0, \forall r \in \mathcal{R}$$

$$z_d = \{0,1\}, \forall d \in \mathcal{D}$$

Variables:

z_d =1 if $d \in \mathcal{Z}$
0 otherwise

a_r =lefthand side coefficient of y_r in constraint

b =righthand side of constraint

The above MILP generates the lefthand side coefficients a_r and the righthand side bound b for the weak delivery set cut where $\mathcal{Z} = \{d|z_d = 1\}$. The constraints ensure that $a_r = 1$ if route r can carry a delivery d for which $z_d = 1$ and $a_r = 0$ otherwise.

If a broken weak volume set cut does not exist, the problem is infeasible. If it does exist, we identify an infeasible delivery set $\mathcal{Z}$. We can then strengthen the coefficients to a strong cut using the same delivery set $\mathcal{Z}$, and add the resulting strong delivery set cut to the master problem. Note that the objective of the cut-generating problem is to find the smallest delivery set that generates a broken cut.

3.4.3 Identifying General Cuts with Optimization

If both of the previous methods fail to find a broken Benders cut, we use another optimization method to identify broken strong general set cuts. Consider the following optimization problem used to compute a weight vector w for the strong general set cut that is most violated by the previous master solution $\bar{y}$.

$$\min_{a,w}\left\{\sum_{r\in\mathcal{R}} a_r^w \bar{y}_r - \sum_{d\in\mathcal{D}} w_d \;\middle|\; a_r^w = \max_x\left\{\sum_{d\in\mathcal{D}} w_d x_{rd} \;\middle|\; \sum_{d\in\mathcal{D}} V_d x_{rd} \le S_r\ \forall r\in\mathcal{R}\right\}\right\}$$

While we can not solve this minimax problem directly, we can solve it using a secondary Benders decomposition. In the master problem we solve for the weights w_d while leaving the route coefficients a_r unconstrained. In the subproblem we compute the true values of the route coefficients $a_r^{\bar{w}}$ for the optimal weights $\bar{w}_d$ computed in the master problem.

General Cut-Generating Master Problem:

$$\min \sum_{r\in\mathcal{R}} a_r \bar{y}_r - \sum_{d\in\mathcal{D}} w_d$$

s.t. $a_r \ge \sum_{d\in\mathcal{D}} w_d \bar{x}_{rd}^i, \forall \bar{x}^i$

$a_r \ge 0, \forall r\in\mathcal{R}$

$w_d \ge 0, \forall d\in\mathcal{D}$

General Cut-Generating Subproblem:

$$\max \sum_{r\in\mathcal{R}} a_r^{\bar{w}}$$

s.t. $a_r^{\bar{w}} = \sum_{d\in\mathcal{D}} w_d x_{rd}, \forall r\in\mathcal{R}$

$\sum_{d\in\mathcal{D}} V_d x_{rd} \le S_r, \forall r\in\mathcal{R}$

$0 \le x_{rd} \le 1, \forall r\in\mathcal{R}, d\in\mathcal{D}$

$x_{rd} \in \{0,1\}, \forall r\in\mathcal{R}, d\in U$

If the resulting cut $\sum_{r\in\mathcal{R}} a_r^{\bar{w}} y_r \ge \sum_{d\in\mathcal{D}} \bar{w}_d$ is broken by the current master solution then we stop and use the cut. If not, the solution $\bar{x}$ to the subproblem gives us a new lower bound on the true value of each a_r^w for any weight assignment w. Since $\bar{x}$ is a feasible assignment of deliveries to routes then $a_r \ge \sum_{d\in\mathcal{D}} w_d \bar{x}_{rd}$. We add such a cut to the cut-generating master problem for each route to make the variables a_r more accurately estimate the values of a_r^w for any weights w. The master problem will then return a new weight vector in each iteration until a weight assignment $\bar{w}$ is found that generates a broken cut.

Not only are the cut-generating master and subproblems relatively easy to solve, but the cuts generated by the assignments $\bar{x}^i$ in each iteration can be reused to find broken cuts for other solutions $\bar{y}$ to the primary master problem. Thus the bounds generated by $\bar{x}^i$ make the variables a_r estimate the true values of a_r^w more accurately each time it is used.

3.4.4 Still No Cuts Found

In the rare case that none of the above methods are able to find a broken cut, we simply add a *nogood* constraint to the primary master problem to ensure that the last solution $\bar{y}$ can not be returned again.

Nogood Constraint: $\sum_{r\in\mathcal{R}}(2\bar{y}_r - 1)y_r \leq -1 + \sum_{r\in\mathcal{R}} \bar{y}_r$

The nogood constraint has a coefficient of 1 for every route used in the infeasible solution and -1 for every route not used. The lefthand side will be less than the number of routes used in the last solution if and only if the solutions are not identical.

In practice, nogood constraints are not very effective as Benders cuts; they make only miniscule changes to the feasible region of the master problem in each iteration. Fortunately, it is rare that a broken cut is not found by the first two methods, so they are rarely necessary to use. If nogood constraints are needed for several iterations in a row, we allow our algorithm to abandon the Benders approach. In this case, we add all the master constraints generated so far to the full problem and solve it using standard methods. Often times these new constraints allow the full problem to be solved in a fraction of the time it would otherwise. In other cases, it takes about as long.

3.5 Test Problem Instances

3.5.1 Instance Generation Methods

Although we did not have any real network data on which to test our Benders algorithm, we designed three methods of generating random problem instances that result in a wide variety of test problem types.

The first method we call "random" and is the simplest of the three. We generate a set of deliveries with random volumes and a set of routes with random capacities and costs proportional to $capacity^{\wedge}(2/3)$ to reflect economies of scale. We then assign the ability to carry each delivery to a random set of routes. Over the instances we generated, we varied the proportion of unsplittable deliveries (members of $\mathcal{U}$), the uniformity of delivery volumes, and the number of deliveries assigned to each route.

The second method we call "geographical" and is similar to the first, except that routes are not assigned to deliveries at random. Instead, we randomly place hypothetical cities on a 2D map and assign each delivery to a pair of cities. Routes are randomly assigned paths through multiple cities on the map with costs proportional to their capacity and length, and any delivery needing to go from one city on a route to a later city is given the ability to be carried by that route. This adds correlation between the sets of routes that can carry each delivery, because packages moving in the same direction are often paired with the same routes. Like the random method, we varied delivery volumes, unsplittable proportions, and numbers of deliveries per route (by changing the number of cities per route).

The third method we call "timeline" and is intended to represent the case when deliveries need to be sent only from one facility to another but on specific time schedules. Each delivery is randomly assigned a time window indicating when it becomes available and the latest time it can be sent. Routes are randomly assigned times at which they leave the facility, and any delivery whose time window includes the route departure time can be carried by that route. This problem setup was motivated by real cases encountered at the U.S. Postal Service (Pajunas, et al. 2007). Like the other methods, we varied delivery volumes, unsplittable proportions, and numbers of deliveries per route (by changing the width of delivery time windows).

3.5.2 Timeline Set Cuts

Problem instances created by our timeline method were a great fit for our Benders algorithm. In these problems, the majority of broken volume set cuts take the form of time windows: that is, they include all deliveries whose time windows lie within some fixed, larger time window. For these problems we made a slight modification to our algorithm to search for broken cuts among time window delivery sets first. Unlike general delivery sets of which there are $2^{\|\mathcal{D}\|}$, time window delivery sets number less than $\|\mathcal{D}\|^2$. This makes it possible to check all such sets rather quickly.

3.6 Experimental Results

We solved each of the randomly-generated problem instances using two methods. First was the default MIP solver of *CPLEX 12.1* applied to the full network problem. This method was used as an experimental control. Second was our Benders algorithm, in which each master and subproblem was solved by the default *CPLEX 12.1* MIP solver. Problems were solved on an Intel Core2 Duo T9600 CPU with 1 GB of allocated RAM. Problem sizes for each instance were set small and gradually increased until their full (control) solution time was at least 60 seconds.

3.6.1 “Random” and “Geographical” Results

The results of the experiments using the “random” and “geographical” problem generation methods were basically identical. They are outlined below. Runtime entries containing the ‘>’ symbol were terminated before reaching optimality.

Method	% Integer	# Routes	# Dels	Average Dels/Route	Std CPLEX Runtime	Benders Runtime	Runtime Ratio
Random	0%	200	70	5	291 s	> 600 s	> 2.1
Random	0%	200	70	10	116 s	23 s	0.20
Random	0%	200	70	20	219 s	8.7 s	0.04
Random	0%	200	70	40	164 s	3.3 s	0.02
Random	50%	140	50	5	130 s	> 600 s	> 4.6
Random	50%	140	50	10	118 s	41 s	0.35
Random	50%	140	50	20	116 s	2.3 s	0.02
Random	50%	200	70	40	96 s	1.3 s	0.01
Random	100%	200	70	5	73 s	> 600 s	> 8.2
Random	100%	140	50	10	178 s	135 s	0.76
Random	100%	140	50	20	99 s	6.2 s	0.06
Random	100%	200	70	40	188 s	10 s	0.06

Method	% Integer	# Routes	# Dels	Average Dels/Route	Std CPLEX Runtime	Benders Runtime	Runtime Ratio
Geographical	0%	700	200	5	98 s	58 s	0.58
Geographical	0%	700	200	10	79 s	25 s	0.31
Geographical	0%	1200	300	20	154 s	2.7 s	0.02
Geographical	0%	1200	300	40	93 s	2.3 s	0.02
Geographical	50%	700	200	5	63 s	> 600 s	> 9.5
Geographical	50%	1200	300	10	99 s	167 s	1.67
Geographical	50%	2100	500	20	92 s	4.9 s	0.05
Geographical	50%	1200	300	40	146 s	2.2 s	0.01
Geographical	100%	700	200	5	60 s	> 600 s	> 9.9
Geographical	100%	1200	300	10	96 s	59 s	0.62
Geographical	100%	2100	500	20	63 s	8.3 s	0.13
Geographical	100%	1200	300	40	62 s	5.8 s	0.09

Only one parameter seemed to affect the performance of our Benders algorithm in all cases: the number of deliveries that could be carried by each route. The other parameters, namely the number of integrality constraints and the amount of volume fluctuation, seemed to have no effect on performance. In particular, the number of deliveries per route determined whether our Benders algorithm was much faster or much slower than the control method. Around five to ten deliveries per route, the algorithms achieved comparable runtimes, with lower values favoring the control algorithm and higher values favoring the Benders algorithm.

All of the experimental results exclusively used strong delivery set cuts in the master problem as described in section 3.4. We also tested the result of using only weak delivery set cuts. The results are similar to those presented here, however the strong cuts generally performed better on problems with integer subproblem constraints.

3.6.2 "Timeline" Results

The results of the experiments using the "timeline" problem generation method were very different. They are outlined below.

Method	% Integer	# Routes	# Dels	Average Dels/Route	Std CPLEX Runtime	Benders Runtime	Runtime Ratio
Timeline	0%	200	80	5	242	117	0.46
Timeline	0%	200	80	10	230	125	0.54
Timeline	0%	200	80	20	62	7.5	0.12
Timeline	0%	200	80	40	94	0.8	0.01
Timeline	50%	200	80	5	91	278	3.03
Timeline	50%	200	80	10	212	132	0.62
Timeline	50%	200	80	20	178	16	0.09
Timeline	50%	200	80	40	71	0.3	0.01
Timeline	100%	200	80	5	196	> 600	> 5.31
Timeline	100%	200	80	10	119	> 600	> 5.07
Timeline	100%	200	80	20	180	211	1.17
Timeline	100%	200	80	40	86	3.5	0.04

Recall that for the “timeline” problems, a special volume set cut search was used which focused on delivery sets contained in time windows, allowing efficient enumeration of all sets of this type. In this case, the number of deliveries per route was a relevant factor, but less so than in the previous experiments. Also, the number of integrality constraints was more relevant in the “timeline” problem, with greater numbers of integrality constraints leading to generally higher runtimes.

3.7 Conclusion

We devised a type of logic-based Benders cut to apply in a decomposition of a network design problem. To construct these cuts, we identify sets of deliveries for which no feasible assignment exists among the currently selected routes. The Benders cuts we add to the master problem ensure that this same infeasibility will not result again in future solutions. We presented methods for finding good cuts to add in each iteration of our Benders method, and examples of their effectiveness on typical problem instances.

Overall, the performance of our Benders algorithm was not as consistent as the standard CPLEX algorithm. However, for certain characteristic classes of problems, our Benders algorithm consistently performed much faster. Our algorithm has the greatest relative performance on problems where the number of deliveries serviceable by each route is large and in some cases when the number of integrality constraints on the deliveries is low. It is possible that other designs of logic-based Benders cuts would be useful to combine with those we have presented to solve the remaining instances. In any case, the Benders approach seems to be an effective tool for this problem, and we believe there is potential for additional progress using this idea.

Chapter 4: Optimization of Inbound Freight Transportation at B/S/H/ North America

4.1 Introduction

B/S/H/ (Bosch and Siemens Home Appliances Group) is among the world's leading manufacturers of home appliances. In 1991, B/S/H/ launched its dishwasher business in North America, which initially imported dishwashers made in Germany. But in 1997, it started production of its European-designed dishwashers in New Bern, North Carolina. In the years thereafter, the site at New Bern was expanded to include the production of cook tops and laundry washers and dryers. To date, the New Bern site hosts three factories: dishwashing, cooking, and laundry that operate almost independently.

Currently, each manufacturing plant orders its supplies separately, even though they share a number of suppliers. Most suppliers ship weekly or bi-weekly. Third-party logistics providers are hired externally, and most often 'full-truckloads' can be negotiated. For smaller shipments, 'less-than-truckloads' are usually used. A full-truckload, or FTL, is a full, dedicated trailer with a typical weight capacity of 45,000 lb. In contrast, freight shipped by less-than-truckload, or LTL, is grouped with other shipments by the carrier and is subject to lower weight limits. The exact pricing of FTL and LTL shipments depends on many factors, the most important of which are the total distance traveled and total weight transported. In addition, the recently introduced fuel surcharges play an increasingly important role. In general however, FTL shipments are more efficient in terms cost per pound-mile when the amount shipped is at least 10,000 lb

(Murphy and Wood 2008). For smaller amounts, LTL shipments typically provide a lower unit cost.

Even though B/S/H/ uses FTL routes whenever possible, LTL routes are applied in many cases, especially when demand is greater than anticipated. Namely, when the amount to be shipped would cause an overload of the FTL truck, the remaining freight must be transported by an LTL shipment. The current situation is depicted in Table 4.1. It shows the relative amount shipped and the relative cost spent for each shipment type. For example, even though 25% of all freight is shipped using LTL, it constitutes 37% of the total transportation costs. FTL, on the other hand, is used to ship 72% of all freight, and only constitutes 47% of the total cost.

Type	**Weight**	**Cost**
LTL	25%	37%
FTL	72%	47%
Other	3%	16%

Table 4.1

Market fluctuations have a huge impact on sourcing volumes. The current operating plan is able to react quickly to changes in volume: each supplier is ordered from independently, so a change in volume does not affect the shipments of other suppliers. Moreover, the addition of LTL trucks allows rapid adaptation to volume increases. This flexibility comes with a cost however. LTL shipments are much more expensive than FTL shipments. Furthermore, the weekly or bi-weekly FTL routes are often under-utilized: the average FTL capacity utilization is only 38%.

The observations above have led us to identify the following opportunities for improvement: (1) since we have found that the three plants share a number of suppliers, is it possible to cut costs by consolidating shipments from one supplier to different plants? (2) Is it possible to cut costs by consolidating shipments from different suppliers?

At first glance, it might seem impossible to put this potential into practice without giving in on flexibility with respect to volume fluctuations. That is, shipment consolidation might appear to be orthogonal to flexibility. Nevertheless, we show in this paper that consolidating shipments can significantly reduce transportation costs, while the resulting solution is actually *more robust* with respect to volume changes.

Our proposed solution involves the careful selection of suppliers into consolidated routes we call *flex-runs*. Flex-runs are essentially flexible flex-runs (the common term for consolidated routes in logistics) designed to handle volume swings along the route. The basic idea of route consolidation is to reduce the relative contribution of the LTL shipments by converting them into consolidated FTL shipments. This is a well-known concept that is typically used by logistics providers to efficiently combine shipments of different customers. An illustrative example of such consolidation is given in Figure 4.1. In this small example, we need to ship freight from four suppliers (the nodes) to one plant. The amount to be shipped is indicated above each supplier node. In the picture on the left, we apply the current strategy: if the amount to be shipped is less than 10,000 lb, use an LTL, otherwise use an FTL. In the picture on the right, we combine the three routes into two FTL flex-runs, FTL1 and FTL2. Given that the per-unit cost for FTL is lower than for LTL, we expect these two flex-runs to be less expensive than the scenario

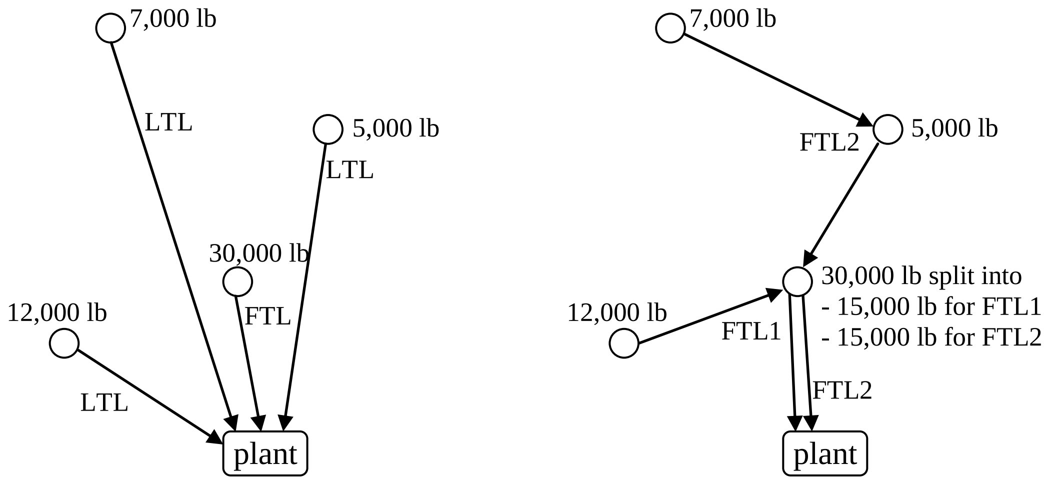


Figure 4.1. The three LTL routes and one FTL route are combined into two FTL routes with a lower total cost.

on the left. In fact, as our experimental results show, we are able to design flex-runs that are expected to reduce the total inbound transportation cost for B/S/H/ by up to 25%.

An important aspect of shipment consolidation is to maintain flexibility with respect to variation in shipment volumes. To account for this, we design our flex-runs in such a way that the expected volume change for one supplier off-sets the expected volume change for the other suppliers. In other words, by exploiting negative correlations between the volume changes of suppliers, we are able to guarantee a lower bound (typically 90-99%) on the probability that the FTL trucks on our flex-runs will remain within capacity. Therefore, our flex-runs turn out to be more robust than the current situation and there is less need for ad-hoc LTL shipments.

Designing the optimal set of flex-runs is a computationally challenging problem (NP-hard to be precise). We propose to model and solve the problem as an integer linear program using column generation. With this technology, we are able to solve the B/S/H/ problem to within 1% of optimality when we allow up to three suppliers in each flex-run. An initial set of our proposed flex-runs is currently being implemented at B/S/H/.

4.2 Problem Description

B/S/H/ has about 600 active suppliers for its three assembly plants. Among these, only 75 suppliers make up 80% of the total shipping volume. We chose to focus our study on these highly active vendors. Each makes at least 40 shipments per year to one or more of the B/S/H/ plants, making them practical targets for one or more weekly flex-runs.

So far, logistics planning at B/S/H/ has focused more on the production schedule of the plants than on freight costs. Each supplier is assigned to a material planner, each of whom has several suppliers to manage. The material planner is responsible for monitoring the production schedule and ensuring that all parts and materials from their suppliers are present and ready when needed by the assembly line. Material planners take quantity discounts into account when placing orders but not shipping costs. Each order is assigned a truck based on its weight: LTL for orders under 10,000 lb, and FTL otherwise. For this reason, many orders between 10,000 and the maximum 45,000 lb are carried on partially-empty or even mostly-empty FTL trucks.

Trucks ordered from LTL carriers are priced according to published, zip code-based rate tables. A fixed discount percentage is also applied that is negotiated annually between B/S/H/ and each carrier. The amount of this discount depends on the quantity and frequency of LTL usage by B/S/H/ and on how the movements of these trucks fit into the carrier's overall network strategy. Sometimes the discount percentage varies by the shipment's state of origin.

Trucks ordered from FTL carriers are priced according to a fixed, negotiated rate per load. Each year, B/S/H/ distributes a list of locations to each of its carriers from which it plans to order many FTL shipments. A multiple-round bidding process takes place that identifies the lowest price a carrier will accept for each shipment made from each location. An exclusive contract is granted to each carrier for those locations on which it is the lowest bidder. B/S/H/ can then order FTL shipments at these fixed prices when needed. Because FTL shipments are not priced by weight, they are somewhat flexible to changes in quantities if they are under capacity.

Data was collected from records kept by a third-party logistics provider. The data identified the date, source, type, weight, and cost of each shipment. Unfortunately, the records did not indicate to which of the three plants each shipment was made. Thus, while we could analyze the benefits of consolidating any set of supplier shipments, we could not analyze the intermediate, and possibly more manageable, benefit of consolidating only within each plant.

4.3 Proposed Solution

Our proposed solution is to group suppliers into carefully-designed flex-runs. As stated previously, these flex-runs are consolidated routes (or milk-runs) that are flexible with respect to fluctuations in shipping volumes. Our experimental results demonstrate that transportation costs can be decreased up to 25%, while changes in demand are better handled and trucks are better utilized.

The computational heart of our approach is an integer programming model based on column generation, where the columns of the model correspond to truck routes. This idea is originally due to Balinski and Quandt (Balinski and Quandt 1964), and has been successfully applied to several large-scale vehicle routing applications (Desrochers, Desrosiers and Solomon 1992)(Barnhart, et al. 1998)(Pajunas, et al. 2007). In our approach, we specify for each route the suppliers visited as well as the amount picked up at each supplier and the expected cost to execute the route. In a pre-processing phase, we generate routes that include all possible combinations of up to three suppliers. Naturally, we could include more than three suppliers per route and use delayed column generation to manage the increased problem size. However, routes of such length are not desirable for B/S/H/. As will be discussed in the next section, we generated several million routes from which we selected about 600,000 useful candidates. To find a solution, the integer programming model selects those routes that satisfy all demand constraints, while minimizing the total cost.

More formally, let $\mathcal{R}$ denote the set of all generated routes. Let $\mathcal{S}$ denote the set of all suppliers. For each route $r \in \mathcal{R}$, let C_r denote its cost. For each route $r \in \mathcal{R}$ and each supplier $s \in \mathcal{S}$, let p_{rs} denote the proportion of the freight picked up at supplier s by route r. We introduce variables y_r for all routes $r \in \mathcal{R}$ representing the frequency of route r, where y_r is a non-negative integer. The problem of finding the set of routes that pick up all freight at minimum total cost can be formulated as the following set-covering problem.

$$\min \sum_{r\in\mathcal{R}} C_r y_r$$

$$\text{s.t.} \quad \textstyle\sum_{r\in\mathcal{R}} \rho_{rs} y_r \geq 1, \forall s \in \mathcal{S} \qquad (1)$$

$$y_r = \{0,1\}, \forall r \in \mathcal{R}$$

Note that constraints $\sum_{r\in\mathcal{R}} \rho_{rs} y_r \geq 1, \forall s \in \mathcal{S}$ (1) are of the form "greater than or equal", whereas commonly equality is used in set covering problems. The reason for this is that we allow some of our generated routes to be combined to (artificially) pick up slightly more than the required amount, provided that this would dominate other solutions in terms of cost and overload risk; see section 4.4 for more details.

This integer programming model contains one variable for each candidate route and one constraint for each supplier. It has one nonzero coefficient for each supplier visited by each route. In our experiments, this model found optimal or near-optimal solutions for problems of 75 suppliers and about 600,000 routes.

4.4 Flex-Run Design

To find an optimal trucking plan, we generate a set of flex-runs (routes) from which our integer program chooses an optimal (or near-optimal) set. The design of our flex-runs consists of three components: route generation, route pricing, and modeling flexibility.

4.4.1 Route Generation

The routing aspect of a flex-run consists of the supplier stops, and for each stop a pickup proportion of the weekly shipment volume. Therefore, we first divide each supplier's weekly shipment into a set of equally-weighted part, each to be assigned to one route. This allows shipments to be split up in a variety of ways, but with finitely many combinations. We chose to divide shipments into packets of around 2,000 lb, though a higher or lower precision may be used in general. For suppliers shipping more than

10,000 lb per week, we divided the delivery into $\lfloor weight/2000 \rfloor$ equal packets. For suppliers shipping less than 10,000 lb per week, we decided not to allow delivery splitting as it would impose an excessive inconvenience on the supplier. In other words, for each supplier $s \in \mathcal{S}$, the number of packets is given by:

$$\#\, packets(s) = \begin{cases} 1 & if\ weight(s) < 10{,}000\ lbs/week \\ \lfloor weight(s)/2000 \rfloor & if\ weight(s) \geq 10{,}000\ lbs/week \end{cases}$$

where $weight(s)$ denotes the total weight of the weekly freight to be picked up at supplier s.

We first generated one-stop LTL routes for all suppliers. If the shipping requirement for a route is less than 10,000 lb, we added a single LTL route for the entire shipment. Otherwise, we added an LTL route for each integer number of packets not exceeding 10,000 lb, which is a reasonable LTL upper bound. An example is shown in the table below, for two suppliers A and B with weekly shipments of 8,000 lb and 11,000 lb, respectively.

Supplier	Weekly Shipment	Capacity of LTL Routes Added
A	$8{,}000\ lbs$	$8{,}000\ lbs$
B	$11{,}000\ lbs$	$(1/5) \times 11{,}000\ lbs = 2{,}200\ lbs$ $(2/5) \times 11{,}000\ lbs = 4{,}400\ lbs$ $(3/5) \times 11{,}000\ lbs = 6{,}600\ lbs$ $(4/5) \times 11{,}000\ lbs = 8{,}800\ lbs$

We next generated FTL routes for all combinations of one, two and three suppliers; it is undesirable for B/S/H/ to have more than three stops on a route. The number of all such combinations is $\sum_{k=1}^{3} \binom{75}{k} = 70{,}375$. Note that each supplier combination can be used with various shipment configurations. For each combination, we checked all possible orders in which the suppliers could be visited before driving to B/S/H/, and selected the sequence with the lowest total mileage. We used *Microsoft*

MapPoint North America with the *MileCharter* add-in to compute a distance matrix for this calculation.

Finally, we need several copies of each route to cover all meaningful combinations of pickup proportions. We first generated all possible combinations for each route, which totaled $\prod_s \#packets(s)$ combinations, where $\#packets(s)$ again denotes the number of packets into which the shipment of supplier s is divided.

Note that not all generated routes will be included in the model. We describe below how we select the flex-runs that are flexible enough to handle volume changes without overloading. In addition to excluding potential route combinations that exceed the allowed overload risk, we also exclude combinations that are unnecessarily small. For example, an FTL route shipping one tenth of a 20,000 lb shipment would be a waste of capacity. However, since lower weight assignments are less likely to overload, we included them as long as their expected excess weight, computed as $E[max(0, weight - 45000)]$, was at least 1 lb less than that of another included route.

4.4.2 Route Pricing

For each of the generated LTL routes, we looked up a rate quote on the web site of the LTL carrier used by B/S/H/ in the region of the supplier. For the FTL routes, we created a model that captures the cost structure as follows.

For a list of previously negotiated FTL routes used by B/S/H/, we plot the cost against the length of those routes in Figure 4.4.2. From this figure, it is apparent that a close relationship exists between the mileage of a route and the minimum price an

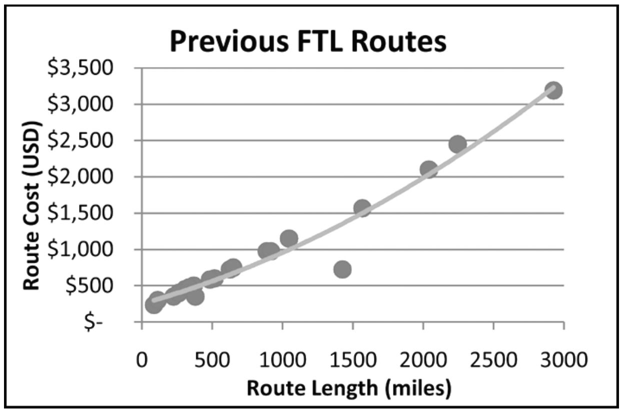


Figure 4.4.2

FTL carrier would accept to haul it. We fit a quadratic curve to this data and used it to estimate prices for our proposed FTL routes. We also added a stopover charge found on the carrier's website for those routes visiting more than one supplier.

4.4.3 Modeling Flexibility

As mentioned above, we only wish to include those routes that are flexible enough to handle volume swings. To model the probability that a generated route would overload, we defined a random variable X_s for each supplier $s \in \mathcal{S}$ representing its weekly shipment. We defined μ_s as the average of X_s and σ_{st} as the covariance in the weekly shipment between two suppliers $s, t \in \mathcal{S}$. For a multiple-stop route r, we compute the mean and variance of the total weekly shipment as follows:

$$\mu_r = \sum_{s \in \mathcal{S}} p_s \mu_s, \quad \sigma_r^2 = \sum_{s,t \in \mathcal{S}} p_s \sigma_{st} p_t$$

From this we generate for each route a Gamma distribution $Gamma(k, \theta)$ with the same mean μ_r and variance σ_r^2 to model the probability distribution of the load of the route. We compute the parameters k and θ as follows:

$$k = \mu_r^2 / \sigma_r^2 , \quad \theta = \sigma_r^2 / \mu_r$$

While a normal distribution may seem more applicable for the load distribution, we find it significantly underestimates the frequency of capacity overload, since much of the outlying area of the normal distribution lies in negative values when variances are as high as those we encountered.

We then selected a service level α, the minimum probability that the route will be within capacity. If the α%-quantile of the $Gamma(k, \theta)$ exceeds the truck capacity, the requirement is not met and the route is excluded. That is, if more than $(1 - \alpha)$% of the distribution is above the 45,000 lb, that weight combination is considered a high overload risk and is excluded from consideration.

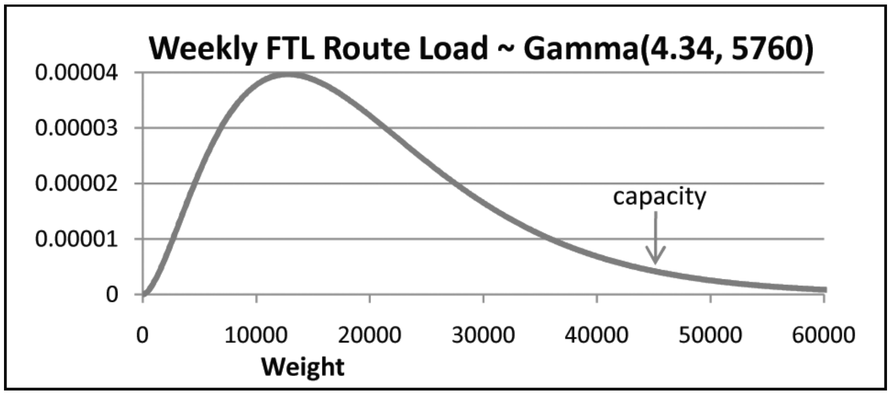
Example:

- Stop 1: Supplier A, 20,000 lb load, ρ_1 =50% pickup
- Stop 2: Supplier B, 15,000 lb load, ρ_2 =100% pickup
- Drive to B/S/H/

Mean load = 25,000 lb, Std. dev. = 12,000 lb

Total weekly load $= \sum_{s=1}^{2} \rho_s L_{sw}$ where L_{sw} = load at stop s in week w

Expected weekly load $\sim Gamma(k, \theta)$, where $\alpha = \frac{mean^2}{variance}, \beta = \frac{variance}{mean}$



The cumulative distribution of Gamma(4.34, 5760) at 45,000 lb is 0.934, which means the route is expected to be withing capacity 93.4% of the time.

The resulting set of flex-runs, containing all LTLs and FTLs of one to three suppliers using all sensible weight combinations, number around 600,000. Consequently, our IP model consists of 600,000 variables and 75 constraints (for 75 suppliers).

4.5 Computational Results

In this section we provide a detailed analysis of the solution we proposed to B/S/H/. In addition to a single solution, B/S/H/ management wanted us to test the robustness of our model parameters. These parameters include the maximum number of stops on each flex-run, the minimum service level α that we require for each flex-run, and the shape of the weight distribution assumed for each flex-run load. We performed several experiments to test different values of the parameters and confirm the accuracy of the model. Throughout this section, cost figures have been rescaled by a fixed multiplier for confidentiality.

For our experiments we have used the integer programming solver CPLEX 11.0 (ILOG S.A. 2007) on a 2.4 GHz machine with 8 GB of RAM running Windows Vista 64 Pro. For the 75-supplier instances, we report the solutions after a duality gap of 1% was reached, i.e., the solution found is provably within 1% of the optimal solution. Finally, we note that for the larger problem instances on around 600,000 variables, most of the available RAM was used. All reported solutions were found within 24 hours total.

4.5.1 Number of Stops Experiments

Our truck routing model contains 75 supplier locations that must be visited by one or more trucks each week. As described in our route generation procedure, we considered only those routes visiting at most three suppliers. This both limits the size of the model for computational tractability and limits the amount of coordination required for B/S/H/ to implement the solution. To study the effects of this maximum-stops parameter on an optimal routing cost, we tested the inclusion of routes with more stops on a smaller example problem containing 15 suppliers. For a problem of this size, the number of these routes is small enough to enumerate and solve to optimality. We also used a distribution of weight and locations similar to our B/S/H/ data.

The following table display the results of this experiment. In this table, "# Suppliers" indicates the number of suppliers in the problem, "Max Stops per Route" indicates the maximum number of stops (i.e., suppliers) on each route, "Min Reliability" indicates the reliability of each route to be within the truck capacity (in this experiment it is set to 95%), and "Optimal Cost" indicates the optimal weekly shipping cost for this problem.

Total Suppliers	Max Suppliers per Route	Min Service Availability	Optimal Cost
15	1	95%	$30,553
15	2	95%	$19,292
15	3	95%	$14,427
15	4	95%	$12,916
15	5	95%	$12,714
15	6	95%	$12,714
15	7	95%	$12,714
15	8	95%	$12,714

As the results show, cost savings improve as more stops are added to the feasible routes, but the marginal value of the extra stops diminishes quickly.

To observe this effect on our full 75-supplier problem, we tested our model with a maximum of 1, 2 and 3 stops per route. The results are depicted in the following table. In this table, we compare four different transportation schedules: the original schedule of B/S/H/ and our three optimized schedules based on routes including a maximum of one, two, and three stops per route, respectively. Here "# Feasible Routes" indicates the number of feasible routes that were generated, "Est. Weekly Fixed Cost" indicates the estimated weekly fixed transportation cost of the schedule, and "Savings" indicates the savings of the optimized schedules with respect to the original schedule.

Schedule	# Suppliers	Max per Route	Min Reliability	# Feasible Routes	Est. Weekly Fixed Cost	Savings
Original	75	1	N/A	75	$134,063	-
Optimization 1	75	1	95%	293	$123,331	8.0%
Optimization 2	75	2	95%	13,052	$105,337	21.4%
Optimization 3	75	3	95%	592,975	$100,000	25.4%

Note that the optimized schedule with 1 stop per route has a lower cost than the original schedule. This is the result of multiple LTL routes being consolidated into single FTL routes for suppliers shipping to multiple plants at B/S/H/.

Because the number of feasible routes increases exponentially with respect to the maximum number of stops, we found it impossible to add routes with more than three stops using out model. The problem files became to large to load into memory. However, we felt that the above data was sufficient to conclude that most of the available savings is captured by our one- to three-stop routes.

4.5.2 Reliability Experiments

The significant cost reduction of our solution is attributable to an increased use of FTL routes, since FTL costs less per pound of capacity than LTL. But to exploit this cost reduction, one must consistently have enough freight to utilize the large, fixed capacity of the FTL. This can be difficult to achieve under conditions of high demand variability: while assigning several suppliers to an FTL route may reduce per-unit costs, it also results in frequent overloading of the truck capacity. In this case, an additional LTL route may be necessary, negating the savings of the FTL route and increasing managerial costs.

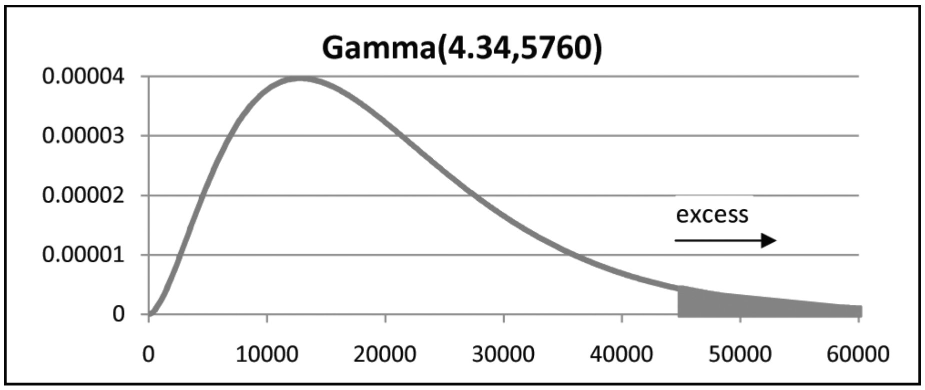
To balance the two goals of minimizing cost and maintaining reliability, we consider only those routes in our model which achieve some minimum threshold of reliability. That is, the proportion of supplier freight assigned to the route must be within its capacity during some fraction of the times it operates. In the following table we report the results of an experiment comparing the savings at various thresholds of reliability, namely 99%, 95% and 90%, referred to as Optimization 3a, 3b, and 3c, respectively. In these experiments, the maximum number of stops is always three. The experiment “Optimization 3” in the previous table is the same as “Optimization 3b” in the following table.

Schedule	# Suppliers	Max per Route	Min Reliability	# Feasible Routes	Est. Weekly Fixed Cost	Savings
Original	75	1	N/A	75	$83,522	-
Optimization 1	75	3	99%	291,110	$69,537	16.7%
Optimization 2	75	2	95%	592,975	$60,888	27.1%
Optimization 3	75	3	90%	851,446	$55,527	33.5%

Based on these results, we decided the solution with 95% reliability was the best trade-off of route costs and managerial costs.

4.5.3 Capacity Overload Experiments

In order to further quantify the cost of unreliability in our solution, we wished to measure the expected cost incurred during the 5% or fewer of cases in which FTL route capacity is overloaded. For this we used a gamma distribution fit to each route's weekly load in our route-generation procedure. To find the expected overload weight of each FTL route, we integrated the excess function $\max(0, weight - 45000)$ over the distribution. We then multiplied this weight by the approximate cost per pound of an additional LTL route to find the expected overload cost of the FTL route. Recalling the example of section 4.4.3, a route with mean load 25,000 lb and standard deviation 12,000 lb has an expected overload weight of 558 lb.



The following table shows the expected overload cost associated with our proposed solution. We estimate this cost for all feasible routes that were included in the model ("All Feasible Routes"), and for the subset of routes included in the solution to two optimizations. The first ("Min Fixed Cost Routes") corresponds to the optimal solution that we have considered thus far, using a fixed cost for each route. The second ("Min Total Cost Routes") adds to the fixed cost the overload cost for each route. Therefore, the objective is slightly higher, and the solution changes slightly.

Route Set	Approximate Weekly Fixed Cost	Expected Weekly Overload Cost	Total Weekly Cost	2007 Estimated Overload Cost
All Feasible Routes	$2,874,994,855	$19,177,002	$2,894,171,856	$11,063,102
Minimum Fixed Cost Routes	$100,000	$4,091	$104,090	$1,821
Minimum Total Cost Routes	$100,144	$3,489	$103,634	$1,561

For each of these three route sets, we report the approximate weekly fixed cost of the routes, the expected weekly overload cost, and the total weekly cost of the routes. For all these numbers, the loads of the routes are modeled by a Gamma distribution. We compare this (in the last column) to the actual overload costs, based on the B/S/H/ routes of 2007. Note that the latter is computed from the data itself, not the distribution fit to the data.

A first observation is that the expected overload cost for the routes included in the optimal solution is relatively high, around 4% of the weekly cost, compared to that of all feasible routes, which is around 0.67%. We see further that the overload costs are somewhat overestimated, but at the same time constitute only a marginal amount of the overall transportation costs. Further, adding the overload costs to the objective did only slightly change the optimal solution, and we therefore omitted the overload costs in our basic model.

To verify the accuracy of our overload cost estimations, we decided to rebuild our model using only 8 months of our available 2007 historical data. That is, estimations of load distribution and reliability used for designing routes were based on only 8 of our 12 months of data. We used months 1, 2, 4, 5, 7, 8, 10, and 11 to eliminate seasonality effects. After optimizing this new model, we compared estimated overload costs during the 8 calibration months and the remaining 4 months. The results are displayed in the following table.

Route Set	Approximate Weekly Fixed Cost	Expected Weekly Overload Cost	Total Weekly Cost	8-Month Calibrated Overload Cost	4-Month Calibrated Overload Cost
All Feasible Routes	$2,850,174,001	$17,916,572	$2,868,090,573	$21,776,023	$8,233,524
Min Fixed Cost Routes	$98,059	$3,873	$101,932	$1,701	$5,696
Min Total Cost Routes	$98,431	$3,391	$101,822	$1,583	$5,174

From this experiment we see that overload costs will inevitably be minimized during the period over which the routes are calibrated, as reflected in the column "8-Month Calibrated Overload Cost". Also, the optimization tends to select routes for which the overload cost is underestimated. However, the overload cost of the four uncalibrated months ("4-Month Uncalibrated Overload Cost") is fairly accurate in the sense that the weekly costs are comparable to those in the previous table, which are based on the model that was calibrated on the entire 2007 data set. Therefore, we consider our estimate to be sufficiently accurate to conclude that overload costs negate only a small fraction of the total savings achieved.

4.6 Analyzing the Proposed Solution

As indicated above, we chose to submit to B/S/H/ the optimal solution using the 95% reliability level with respect to the overload risk. We next provide a detailed analysis of this solution and compare it to the original shipping situation at B/S/H/.

We first discuss the structure of the original routes and the optimized routes in terms of the amount of LTL and FTL shipments. To allow a meaningful comparison, the reported costs of the original routes are based on the model we presented in the Flex-Run Design section. In the following tables, we report for each route type the average mileage, the average cost per week, the average ton-mileage per week, and the average cost per ton-mile. A first observation is that the number of routes dramatically decreased from 165 to 60 routes in total, most of which are 3-stop FTLs. Interestingly, the average mileage for the resulting optimized LTL routes is higher than the original LTL routes.

Indeed, an analysis of our solution showed that the best candidates for optimized LTL routes are those that are geographically isolated from the others, while other factors are low shipping weight and high variability.

Original Routes					
Route Type	#/week	Avg. Mileage	$/week	ton*mi/week	$/ton*mi
1-Stop LTL	126	623 mi	$ 62,853	122,706	$ 0.49
1-Stop FTL	39	436 mi	$ 71,210	258,966	$ 0.29

Optimized Routes, 95% Reliability Level					
Route Type	#/week	Avg. Mileage	$/week	ton*mi/week	$/ton*mi
1-Stop LTL	19	880 mi	$ 11,873	30,813	$ 0.39
1-Stop FTL	9	128 mi	$ 7,871	23,612	$ 0.33
2-Stop FTL	9	305 mi	$ 11,807	49,309	$ 0.24
3-Stop FTL	23	897 mi	$ 68,449	341,852	$ 0.20

To identify which routes and suppliers attribute most to the total cost savings of about 25%, we present in the following table the average unit costs for the different route conversion: from LTL to Optimized LTL, from LTL to Optimized FTL, and from FTL to Optimized FTL. The most important contribution stems from the conversion of LTL routes into Optimized FTL routes (from $0.44 to $0.21 per ton-mile), while converting LTLs and FTLs into optimized LTLs and FTLs contribute relatively less to the savings.

Type Conversion	Original Routes	Optimized Routes 95% reliability	
	$/ton*mi	ton*mi/week	$/ton*mi
LTL → LTL Optimized	$ 0.44	30,813	$ 0.39
LTL → FTL Optimized	$ 0.44	91,893	$ 0.21
FTL → FTL Optimized	$ 0.29	258,966	$ 0.21

Finally, we analyze the truck utilization of our optimized solution, as compared to the original shipping situation at B/S/H/. As was reported in Table 4.1, the current shipping situation uses only 38% of the FTL truck capacity, on average. We report in the following table the average truck utilization of our proposed solution, as well as the 'peak utilization'. The latter is defined as the average of the highest loads over all trucks. For example, the 10% peak utilization collects for all trucks the 10% highest load volumes, and the average is reported as the 10%-peak volume (in percentage of the capacity). Similar numbers are displayed for the 5%-peak volume.

FTL	Original	95% reliability	90% reliability
Average Utilization	38.1%	48.5%	55.5%
10%-Peak Utilization	-	81.9%	96.5%
5%-Peak Utilization	-	96.6%	115.6%

The results clearly show an improvement in average truck utilization with respect to the original situation (from 38.1% to 48.5%). We note that the average truck utilization is deliberately kept at a relatively low level by our optimization model to allow a reliability buffer for the flex-runs. That this is indeed needed is shown by the peak utilization of the trucks. For example, for the 95% reliability level, the 10%-peak utilization is 81.9% of the truck capacity. This means that many flex-runs will in fact be almost completely filled during roughly 10% of the schedule execution. We note that for the 90% reliability level, the higher chance of overloading is witnessed by the 5%-peak utilization of 115.6% of the truck capacity (i.e., an overload of 15.6%).

4.7 Conclusion

We have described an approach to optimize the inbound freight logistics for Bosch/Siemens in North America, a leading manufacturer of home appliances. The computational model of our approach is based on combining individual supplier's shipments into "flex-runs" (milk-runs that are flexible with respect to shipping volume fluctuations) such that total logistic costs are reduced while maintaining, if not increasing, the reliability of the shipping schedule. To this end, we exploit correlations between the variations in shipping volume for different supplier locations, to ensure that each potential route does not overload with respect to a given level of reliability. The design of our routes further allows the shipping volumes to be combined in various proportions using any desired level of precision. Our approach applies integer linear programming and column generation, where the columns of the model correspond to feasible (flex-run) routes.

We have demonstrated the benefits of our approach on real-life data from Bosch/Siemens. For this data set, we pool volumes from up to three suppliers onto a single route, which allows us to generate the required feasible routes in a pre-processing phase. When more than three stops are demanded, the number of feasible routes quickly grows too large, and delayed column generation can be employed instead.

Our most important finding is that with our optimized solution, expected cost savings of up to 25% can be achieved with respect to the current shipping situation, while at the same time, the robustness of the schedule with respect to changes in shipping volume increases. An initial implementation of our proposed solution is currently initiated at Bosch/Siemens. Our approach is not restricted to the situation at Bosch/Siemens however, but in fact is broadly applicable.

References

Bai, Lihui, and Paul A. Rubin. "Combinatorial Benders Cuts for the Minimum Tollbooth Problem." *Operations Research* 57, no. 6 (2009): 1510-1522.

Balinski, M. L., and R. E. Quandt. "On an integer program for a delivery problem." *Operations Research* 12, no. 2 (1964): 300-304.

Barnhart, C., E. L. Johnson, , G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. "Branch-and-Price: Column Generation for Solving Huge Integer Programs." *Operations Research* 46, no. 3 (1998): 316-329.

Benders, J. F. "Partitioning Procedures for Solving Mixed Variables Programming Problems." *Numerische Mathematik* 4 (1962): 238-252.

Che, A., and C. Chu. "Single-track multi-hoist scheduling problem: A collisionfree resolution based on a branch-and-bound approach." *International Journal of Production Research* 42 (2004): 2435-2456.

Chen, H., C. Chu, and J.-M. Proth. "Cyclic scheduling of a hoist with time window constraints." *IEEE Transactions on Robotics and Automation*, no. 14 (1998): 144-152.

Codato, G., and M. Fischetti. "Combinatorial Benders' Cuts for Mixed-Integer Linear Programming." *Operations Research* 54, no. 4 (2006): 756-766.

Cortés, Cristián E., Martín Matamala, and Claudio Contardo. "The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method." *European Journal of Operational Research* 200, no. 3 (2010).

Desrochers, M., J. Desrosiers, and M. Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows." *Operations Research* 40, no. 2 (1992): 342-354.

Fazel-Zarandi, Mohammed M., and J. Christopher Beck. "Solving a Location-Allocation Problem with Logic-Based Benders Decomposition." *Principles and Practice of Constraint Programming*, no. 5732 (2009): 244-351.

Fischetti, M., D. Salvagnin, and A. Zanette. *Minimal infeasible subsystems and Benders cuts.* University of Padua, Italy: Technical report, 2009.

Gabrel, Virginie, Arnaud Knippel, and Michel Minoux. "A Comparison of Heuristics for the Discrete Cost Multicommodity Network Optimization Problem." *Journal of Heuristics*, no. 9 (2003): 429-445.

Gabrel, Virginie, Arnaud Knippel, and Michel Minoux. "Exact solution of multi-commodity network optimization problems with general step cost functions." *Operations Research Letters*, no. 25 (1999): 15-23.

Gendron, Bernard, Teodor Gabriel Crainic, and Antonio Franfioni. "Multicommodity Capacitated Network Design." In *Telecommunications Network Planning*, by B. Sanso and P. Soriano, 1-19. Norwell, MA: Kluwer Academics Publishers, 1997.

Geoffrion, A. M. "Generalized Benders decomposition." *Journal of Optimization Theory and Applications*, no. 10 (1972): 237-260.

Hooker, J. N. *Integrated Methods for Optimization.* Springer, 2006.

Hooker, J. N. "Planning and Scheduling by Logic-Based Benders Decomposition." *Operations Research* 55, no. 3 (2007): 588-602.

ILOG S.A. *ILOG CPLEX 11.0 Reference Manual*. 2007.

Kim, K. H., and Y.-M. Park. "A crane scheduling method for port container terminals." *European Journal of Operational Research* 156 (2004): 752-768.

Lei, L., and T. J. Wang. *A proof: The cyclic hoist scheduling problem is NP-complete.* Working Paper, Rutgers University, 1989.

Lei, L., and T. J. Wang. "The minimum common-cycle algorithm for cycle scheduling of two material handling hoists with time window constraints." *Management Science* 37 (1991): 1629-1639.

Lei, L., and T. J. Wang. "Determining optimal cyclic hoist schedules in a single-hoist electroplating line." *IIE Transactions*, no. 26 (1994): 25-33.

Lei, L., R. Armstrong, and S. Gu. "Minimizing the fleet size with dependent time-window and single-track constraints." *Operations Research Letters* 14 (1993): 91-98.

Leung, and Levner. "An efficient algorithm for multi-hoist cyclic scheduling with fixed processing times." *OR Letters* 34, no. 4 (2006): 465-472.

Murphy, Jr., P. R., and D. F. Wood. *Contemporary Logistics.* 9. Pearson Prentice Hall, 2008.

Pajunas, A., E. J. Matto, M. A. Trick, and L. F. Zuluaga. "Optimizing Highway Transportation at the United States Postal Service." *Interfaces* 37, no. 6 (2007): 515-525.

Phillips, L. W., and P. S. Unger. "Mathematical programming solution of a hoist scheduling problem." *AIIE Transactions*, no. 8 (1976): 219-321.

Rasmussen, Rasmus V., and M. A. Trick. "A Benders Approach for the Constrained Minimum Break Problem." *European Journal of Operational Research* 177, no. 1 (2007): 198-213.

Rodo'sek, R., and M. Wallance. *A generic model and hybrid algorithm for hoist scheduling problems.* Vol. 1520, in *Principle and Practice of Constraint Programming (CP-1998)*, by M. Maher and J.-F. Puget, 385-399. Berlin: Springer, 1998.

Zhou, Z., and L. Li. "A solution for cyclic scheduling of multi-hoists without overlapping." *Annals of Operations Research (Online)*, 2009.