Extreme-Point Search Heuristics for Solving Interval-Flow Transshipment Networks

Richard S. Barr[*] Robert H. Jones [†]

June, 2012

Abstract

Interval-flow networks are a special class of network models that can include minimum-flow requirements on some or all active arcs in a feasible solution. While this extension expresses constraints often encountered in practice, the resulting NP-hard problems are challenging to solve by standard means. This work describes a heuristic that explores adjacent extreme-point solutions to quickly find high-quality feasible solutions to large-scale instances of this problem class. Computer software implementing this approach is tested on problems with up to 40,000 nodes and 1 million arcs, giving solution speeds over 400 times faster than a leading commercial optimizer.

1 Introduction

This paper describes solution methods for *interval-flow transshipment networks*, a relatively new class of network flow models in which disjunctive constraints restrict each arc's flow to be either zero or within a stated interval. As detailed below, the traditional pure network formulation is extended by including *conditional lower bounds* to require minimum flow levels on arcs with activity.

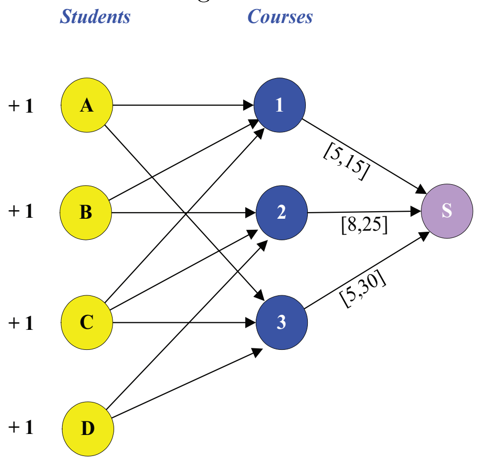
The addition of minimum-use activity levels expands the applicability of the popular network-flow models to include quantity discounts, economic viability thresholds, lot-sizing considerations, and demand triggers for technology decisions. A familiar examples would be minimum class sizes for a college course to be offered (see Figure 1) and minimum purchase amounts to receive a lower price. Figure 2 depicts a software licensing model that provides quantity discounts for bulk purchases and costs of delaying and expediting license renewals to take advantage of the lower prices.

Interval-flow modeling enhances logistics networks [9, 18] such as trucking, shipping, or road toll planning [10] by requiring a minimum shipment size for

[*]Department of Engineering Management, Information, and Systems, Lyle School of Engineering, Southern Methodist University, Dallas, TX 75275, USA, barr@smu.edu

[†]Oncor Electric Delivery, Dallas, TX, USA, robert.jones@oncor.com

Figure 1: Interval-flow class assignments with minimum enrollment levels



transport to occur. Production planning often involves minimum lot sizes based on line configurations or the physical characteristics of the equipment. This is common in the manufacturing of low-quantity or specialized parts, such as military aircraft connectors, where a minimum order quantity is needed to justify a production run [16]. Interval-flow arcs can add realism to waste-water treatment system models (such as found in Jarvis et al [13, 17]) to refine the location and plant capabilities. The design of wireless data networks to support smart-grid electrical distribution systems are readily modeled as in Figure 3 with minimum and maximum equipment usage levels and concentrator selection [7].

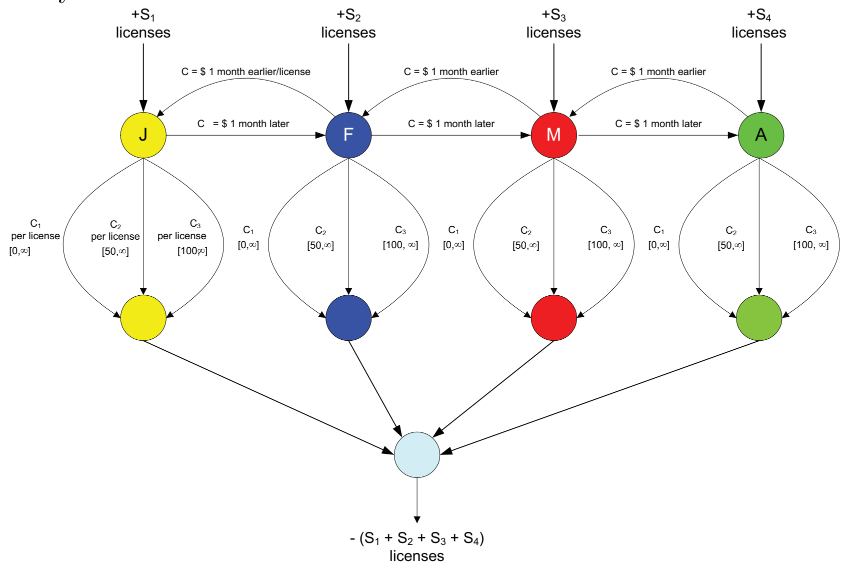
The addition of conditional lower bounds and their associated logical constraints to a linear network model creates a disjunctive program [3, 4, 19] that can be solved via mixed integer programming methods. The heuristic methods developed herein build on the highly efficient algorithms developed for pure network problems [5, 8, 11] to make large-scale instances of interval-flow networks readily solvable.

The following sections mathematically define the interval-flow transshipment problem and describe the heuristic approach to solving the problem. A computer code based on these methods is tested on a set of randomly generated transshipment problems and compared with state-of-the-art commercial software to demonstrate the effectiveness of the approach.

2 Problem Definition

The interval-flow transshipment network problem (IFN) can be defined for a directed graph with sets of nodes N and arcs A as:

Figure 2: Interval-flow network for coordinating software license renewals and quantity discounts

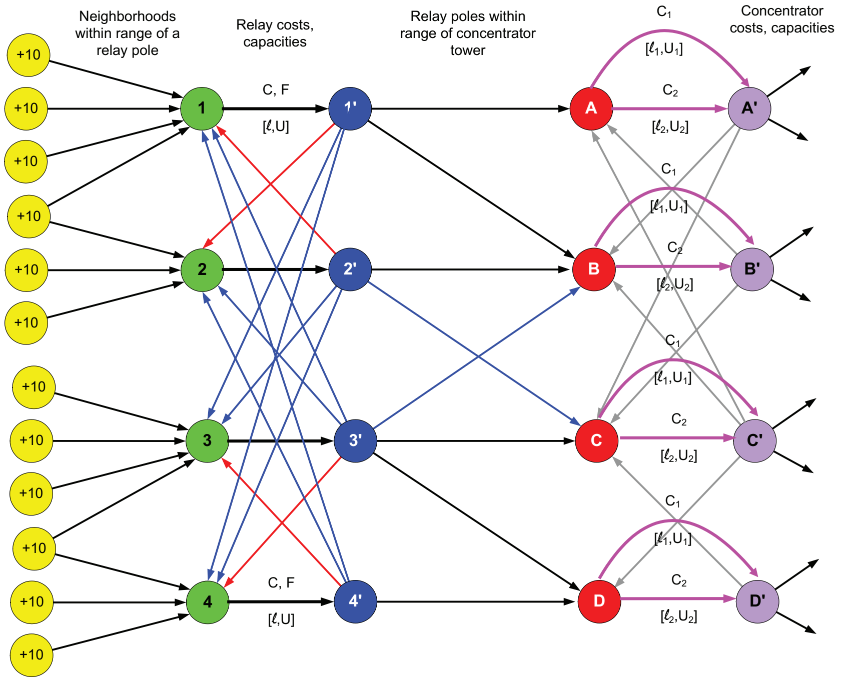


$$
\begin{aligned}
\text{IFN:} \qquad \min \sum_{(i,j)\in A} c_{ij}x_{ij} &= z && (1)\\
s.t. \qquad \sum_{(i,k)\in A} x_{ik} - \sum_{(k,j)\in A} x_{kj} &= b_k, \forall k \in N && (2)\\
\ell_{ij}y_{ij} \le x_{ij} &\le \upsilon_{ij}y_{ij},\ \forall (i,j) \in A && (3)\\
0 \le y_{ij} &\le 1,\ \forall (i,j) \in A && (4)\\
y_{ij} & \quad \text{integer},\ \forall (i,j) \in A && (5)
\end{aligned}
$$

where, for each arc $(i,j) \in A$, x_{ij} is the arc flow, $y_{ij} \in \{0,1\}$ the activity-indicator variable, c_{ij} the unit cost of flow, ℓ_{ij} the conditional lower bound, and υ_{ij} the upper bound. For each node $i \in N$ there is a requirement b_i, positive if supply, negative if demand, and zero if a pure transshipment node. Requirements and all bounds are assumed to be integer-valued constants and that the total supply, ξ, is equal to the total demand. To simplify exposition, the development given here is for the uncapacitated case, where all $\upsilon_{ij} = \xi$.

The objective is to minimize total cost z, subject to node balance constraints (2) and flow bounds (4)–(5). Nonzero flow activity on an arc is enabled by $y_{ij} = 1$, which also enforces the conditional lower bound in (4). (Such a pairing of x_{ij} and y_{ij} variables is sometimes referred to in modeling languages as a "semi-continuous variable.") Note that if arc (i,j) has $\ell_{ij} = 0$ or 1, it is the equivalent of a pure network arc, whose flows are nonnegative integers.

Figure 3: Interval-flow sensor network with minimum usage levels and equipment choices


Neighborhoods within range of a relay pole
Relay costs, capacities
Relay poles within range of concentrator tower
Concentrator costs, capacities
+10
1
2
3
4
1'
2'
3'
4'
C, F
[ℓ,U]
A
B
C
D
A'
B'
C'
D'
C₁
[ℓ₁,U₁]
C₂
[ℓ₂,U₂]

3 Interval-Pivoting Heuristics

The algorithmic approach taken is based on the extreme-point search method developed by Walker [20] and expanded by Apte and Barr[2] for interval-flow transportation problems to exploit the underlying network structure. As can be noted from the IFN formulation above, by fixing all y_{ij} values to some combination of zeros and ones (i.e., setting the arcs' active/inactive status) a pure network problem results. Hence, the optimal solution to IFN lies at an extreme point of a related transshipment problem whose basis is a spanning tree of the nodes. Note that when some y_{ij} is fixed to 1, then $\ell_{ij} \leq x_{ij} \leq \upsilon_{ij}$ and arc (i, j) can be nonbasic at either ℓ_{ij} or υ_{ij}.

With this property, a heuristic can be built on existing primal network simplex technology to exploit the solution's structure, including spanning-tree solution representation and streamlined execution of simplex steps—such as pivoting and pricing—via specialized data structures. The heuristics described herein extend the interval-pivoting heuristics for transportation problems [2] to transshipment problems and explore a series of adjacent-extreme-point search methods to capitalize on the mathematical structure of IFN solutions.

3.1 Overview

The interval-pivoting algorithm for IFN consists of three steps.

1. Relax the integrality constraints (5) of IFN and solve as a pure network. If the resulting solution is feasible to IFN, then the problem is solved and the solution is optimal.
2. Seek an *interval-feasible solution* [one satisfying the interval-flow restrictions (3)–(5)] by pivoting in non-basic arcs that affect the flow on the interval-infeasible arcs (interval-pivoting). The goal is to move an interval-infeasible arc to be non-basic with a flow of 0 or ℓ.[1]
   (a) H_1: Escalate costs on IF-infeasible flow arcs and re-optimize
   (b) H_2: Use interval-pivoting to reduce the number of IF-infeasible arcs
   (c) H_3: Use interval-pivoting to minimize total distance to feasibility (infeasible flow distance from nearest bound)
3. H_4: Once a feasible solution is found, seek an improved solution via strategic oscillation [12], temporarily relaxing the interval-flow constraints, finding an improved solution, and re-applying step 2 to move back to interval feasibility.

The interval-pivoting method is demonstrated in Figure 4 where the current flow level for each arc is shown by • on its respective axis. Assume that arcs x_1, x_2, and x_3 are active and form the basis-equivalent path (representation) of x_{in}, a potential incoming arc. In the case where the flow on x_{in} is increased by amount A, this will also increase flow on x_1 and x_2 and decrease x_3 by A. However, since x_{in} must have a minimum flow of ℓ (indicated by point B) to join the basis and be interval-feasible, the flow on x_2 would be increased past its upper bound υ_2; hence x_{in} is blocked from joining the basis. If υ_2 had been at point B, x_{in} would be pivoted into the basis at level ℓ and x_2 would leave the basis at its upper bound.

The components that make up the problem definition and the heuristic to solve it are shown in Table 1. For this formulation, it is assumed that all network information such as A, N, $\mathbf{b}$, the total supply available, and the conditional lower bound values for each arc are given.Let P be the set of all arcs that have the flow $x_{ij} > \ell_{ij}$. These are the arcs that have a feasible flow and could be part of a final feasible solution. Let L be the set of arcs where $x_{ij} = \ell_{ij}$. These arcs are currently feasible but any flow decrease must equal ℓ_{ij} to maintain feasibility. Let ZE be the set of all arcs where $x_{ij} = 0$ and $\ell_{ij} > 0$. These arcs are currently feasible but any flow increase must be greater than or equal to ℓ_{ij} to maintain feasibility. Let IF be the set of all arcs $0 < x_{ij} < \ell_{ij}$. These arcs are currently infeasible and must be driven into one of the other sets to create

[1]Note that when some y_{ij} is fixed to 1, then $\ell_{ij} \leq x_{ij} \leq \upsilon_{ij}$ and arc (i,j) can be nonbasic at either ℓ_{ij} or υ_{ij}. The heuristic, therefore, seeks to make each arc's flow be either 0, within the flow interval, or "nonbasic" at ℓ_{ij} or υ_{ij}.

Figure 4: Solution tree with incoming arc, effect on interval-flow arc flows

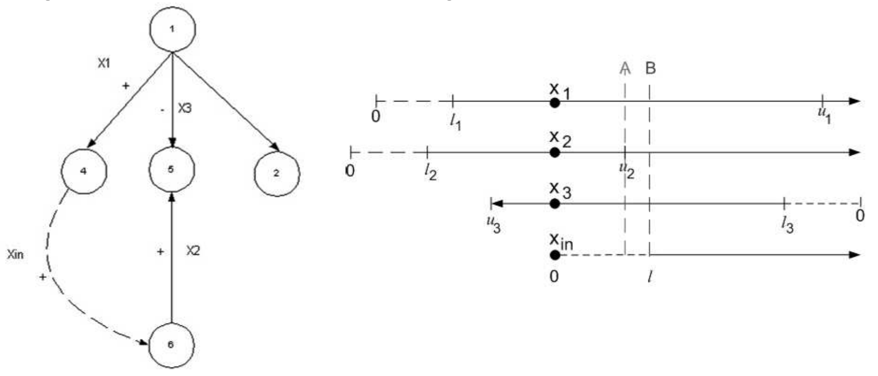


Table 1: Interval-Flow Network Component Definitions

Component	Type	Definition
$\mathbf{X}$	variable	vector containing all x_{ij}s
$\mathbf{C}$	constant	vector containing all c_{ij}s
$\mathbf{C}^m$	variable	vector containing modified c_{ij}^ms
$\mathbf{LB}$	constant	vector containing all ℓ_{ij}s
$P \subset A$	variable	set of all arcs with $x_{ij} > \ell_{ij}$
$L \subset A$	variable	set of all arcs with $x_{ij} = \ell_{ij}$ and $\ell_{ij} > 0$
$IF \subset A$	variable	set of all arcs with $0 < x_{ij} < \ell_{ij}$
$ZE \subset A$	variable	set of all arcs with $x_{ij} = 0$ and $\ell_{ij} > 0$
$T \subset A$	variable	set of all arcs that form a network tree
$\mathbf{b}$	constant	vector with supply or demand for each node in N

a feasible solution. A feasible solution requires $IF = \emptyset$. Set T contains those arcs that make up a spanning-tree extreme-point solution for IFN or one of its relaxations.

The heuristic requires an initial solution to begin but that solution does not need to be interval-feasible. The approach to generate an initial solution is solve IFN with constraint (5) relaxed. With this relaxation, IFN now becomes a standard minimum cost flow network formulation and the standard approach described by Bradley et al. [8] with enhancements to the spanning tree labeling described in [5] and the candidate list approach described in [15] can be used to efficiently solve this problem. This approach is used repeatedly in the heuristic and will be noted as SOLVE in the heuristic description. SOLVE always returns set T, the objective function value z, and the flows x_{ij} for every arc (i, j) in A.

The solution to the relaxed problem provides a lower bound to the solution of IFN. It is not expected that the solution to the relaxed problem is also a feasible solution to IFN but if it is, the solution is optimal and the heuristic terminates.

Table 2: Modified Cost Values for Force Zero Algorithm

Arc Flow	Previous Cost Value	Modified Cost Value
$\ell_{ij} \leq x_{ij} \leq \xi$	c_{ij}^m	c_{ij}
$0 < x_{ij} < \ell_{ij}$	c_{ij}^m	αc_{ij}^m
$x_{ij} = 0$ and $\ell_{ij} > 0$	c_{ij}^m	$c_{ij}^m + c_{ij}$ if $c_{ij}^m \leq \beta c_{ij}$ otherwise unchanged

Figure 5: Force Zero(A, N, T, $\mathbf{X}$, $\mathbf{C}$, $\mathbf{LB}$, $feasloop$) Algorithm

```
Input: $A, N, T$, $\mathbf{X}$, $\mathbf{C}$, $\mathbf{LB}$, $feasloop$
Output: $IF, T$, $\mathbf{X}$, $z$
  $subfeas \leftarrow 0$
  $\mathbf{C}^m \leftarrow \mathbf{C}$
  $(IF, \mathbf{C}^m) \leftarrow IFEAS(IF, \mathbf{C}^m, \mathbf{C}, \mathbf{X}, \mathbf{LB}, A)$
  while $IF \neq \emptyset$ and $subfeas \leq feasloop$ do
    $(T, z, \mathbf{X}) \leftarrow SOLVE(A, N, \mathbf{X}, \mathbf{C}^m)$
    $(IF, \mathbf{C}^m) \leftarrow IFEAS(IF, \mathbf{C}^m, \mathbf{C}, \mathbf{X}, \mathbf{LB}, A)$
    $subfeas \leftarrow subfeas + 1$
  end while
  Return $(IF, T, \mathbf{X}, z)$
```

The overall heuristic consists of three main algorithms that approach the interval-flow network problem in different ways. When grouped together, the heuristic algorithm is effective in solving IFN. The following sections describe the three main methods.

3.2 Method H_1: Force Zero Algorithm

The force zero algorithm—shown in Figures 5-6—attempts to solve the problem described in equations 6 - 8. The problem defined by FZ is identical to the standard minimum cost flow network problem. What makes this solve an interval-flow problem is the modified cost value for each arc shown as c_{ij}^m.

$$\text{FZ:} \qquad \min \sum_{(i,j)\in A} c_{ij}^m x_{ij} = z \tag{6}$$

$$s.t. \quad \sum_{(i,k)\in A} x_{ik} - \sum_{(k,j)\in A} x_{kj} = b_k, \forall k \in N \tag{7}$$

$$0 \leq x_{ij} \leq \xi,\ \forall (i,j) \in A \tag{8}$$

Once an initial solution is found, the flow on each arc is examined to determine if the flow is interval-feasible. The cost on each arc is temporarily modified as

Figure 6: IFEAS($IF, \mathbf{C}^m, \mathbf{C}, \mathbf{X}, \mathbf{LB}, A$) Algorithm

```
**Input:** $IF, \mathbf{C}^m, \mathbf{C}, \mathbf{X}, \mathbf{LB}, A$
**Output:** $IF$, $\mathbf{C}^m$
  **for all** arc$(i, j) \in A$ **do**
    **if** $0 < x_{ij} < \ell_{ij}$ **then**
      arc$(i, j) \in IF$
      $c^m_{ij} \leftarrow \alpha c^m_{ij}$
    **end if**
    **if** $x_{ij} \geq \ell_{ij}$ **then**
      $c^m_{ij} \leftarrow c_{ij}$
    **end if**
    **if** $x_{ij} = 0$ and $\ell_{ij} > 0$ and $c^m_{ij} \leq \beta c_{ij}$ **then**
      $c^m_{ij} \leftarrow c^m_{ij} + c_{ij}$
    **end if**
  **end for**
  Return ($IF$, $\mathbf{C}^m$)
```

shown in Table 2 with all modified costs c^m_{ij} initialized to c_{ij}. The cost value is unchanged for any arc that has a flow in the feasible range. The cost of an arc that is infeasible is increased by a factor of α each time the IFEAS routine is executed. As the cost is increased with each loop iteration, the arc is eventually driven out of the solution if possible forcing the flow on the infeasible arc to zero. Any arc that currently has a flow of zero but has a conditional lower bound value that is greater than zero also has a cost penalty applied. The value of β is set to allow these arcs to have flow but are penalized because it is possible that any flow put on these arcs could be infeasible. Using these arcs is therefore discouraged but not prohibited.

The values of $\alpha = 3$ and $\beta = 3$ are determined by experimentation to give the best results overall. The loop counter $feasloop$ is set to a value of 200 although most problems take less than 10 iterations to find a feasible solution. The IFEAS routine that populates set IF and calculates the modified cost values is shown in Figure 6.

3.3 Method H_2: Exchange Algorithm

The purpose of the exchange algorithm (given in Figure 7) is to move arcs that are in set IF to either the L or the ZE sets by adjusting network flows. The exchange algorithm accomplishes this by interval-pivoting steps that seek to solve problem EX:

Figure 7: Exchange(A, IF, L, ZE, T, $\mathbf{X}$, z, $tarcs$) Algorithm

```
Input: $A$, $IF$, $L$, $ZE$,$T$, $\mathbf{X}$, $z$, $tarcs$
Output: $IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$
  $loopcount \leftarrow 0$
  $candidate \leftarrow$ CANDLISTIF($IF, L, ZE, A$)
  {Find an arc that could change flow of an infeasible arc}
  while $candidate \neq 0$ and $loopcount \leq tarcs$ and $IF \neq \emptyset$ do
    $(IF, L, ZE, z, \mathbf{X}, flowchange) \leftarrow$ SOLVEIF($A, IF, L, ZE, \mathbf{X}, z$)
    if $flowchange = 0$ then
      $loopcount \leftarrow loopcount + 1$
      tabu level$_{pq} \leftarrow tabutime$
    else
      $loopcount = 0$
    end if
    $candidate \leftarrow$ CANDLISTIF($IF, L, ZE, A$)
  end while
  Return ($IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$)
```

$$\begin{aligned}
&\text{EX:} & \min Card(IF) && && (9)\\
&s.t. & \sum_{(i,k)\in A} x_{ik} - \sum_{(k,j)\in A} x_{kj} &= b_k, \forall k \in N && (10)\\
& & 0 \leq x_{ij} &\leq \xi,\ \forall (i,j) \in A && (11)
\end{aligned}$$

The constraints (10)–(11) are identical to those in the minimum cost flow network problem. The objective function in equation 9 is to minimize the cardinality of set IF.

A modified version of SOLVE was created to solve EX. It starts by examining all arcs that are not currently a member of the set T. For each arc $(i,j) \notin T$ considered, the basis equivalent path (BEP) is traced to determine if the flow on arc $(p,q) \in IF$ would be affected if arc (i,j) is included in T. If arc (i,j) would not affect an arc in IF it is skipped and removed from further consideration. Otherwise, it is maintain in a candidate list for further examination and possible inclusion into T.

Once an arc (i,j) is selected for testing, the algorithm determines if an increase in the flow on (i,j) would increase or decrease the flow on (p,q). If the flow on (p,q) decreases then the amount of flow change in the BEP must be x_{pq} moving (p,q) into set ZE. But, if the flow on (p,q) increases then the amount of flow change in the BEP must be $\ell_{pq} - x_{pq}$ moving (p,q) into set L. This operation is to strictly decrease the cardinality of IF so no other arc in the BEP can enter IF when (p,q) leaves IF. If all of these conditions are met, the flows on the BEP are updated, (i,j) becomes an element in T, $(p.q)$ leaves T and IF and becomes an element of either L or Z.

To avoid cycling at various stages, elementary techniques from the Tabu Search literature [12] are employed. Therefore, if any of the above conditions are violated, a tabu value is assigned to (i,j) to prevent (i,j) from being considered again for at least *tabutime* inclusion attempts. The value of $tabutime = 4$ is determined by experimentation to give the overall best results for arc selection while minimizing cycling. The algorithm repeatedly checks arcs until either $IF = \emptyset$, there are no arcs that can be found that affect the remaining arcs in IF, or the overall loop counter in the algorithm expires. The loop counter expiration value is determined by experimentation. The loop counter expires when the total number of consecutive failed update attempts exceeds the total number of arcs in the network.

The exchange algorithm is shown in Figure 7. The routine CANDLISTIF finds arcs to populate the candidate list based on the previously described rules. The routine SOLVEIF is a modified version of SOLVE that uses all of the same tree management routines as SOLVE with the modified flow change rules previously described. SOLVEIF returns the variable *flowchange* as either 0, x_{pq}, or $\ell_{pq} - x_{pq}$. A value of 0 indicates that the flow in the BEP could not change enough to drive (p,q) out of IF. The constant *tarcs* is set to the total number of arcs in the network and is used as part of the loop termination conditions.

3.4 Method H_3: Minimize Distance to Feasibility Algorithm

The purpose of the minimize distance algorithm is to minimize the sum of the distance to feasibility as defined by equation 12. The distance to feasibility for any arc $(p,q) \in IF$ is $min(x_{pq}, \ell_{pq} - x_{pq})$ and indicates the smallest flow change on arc (p,q) required to move the arc out of set IF.

$$\text{DM:} \qquad \min \sum_{(i,j)\in IF} min(x_{ij}, \ell_{ij} - x_{ij}) \tag{12}$$

$$s.t. \quad \sum_{(i,k)\in A} x_{ik} - \sum_{(k,j)\in A} x_{kj} = b_k, \forall k \in N \tag{13}$$

$$0 \leq x_{ij} \leq \xi, \ \forall (i,j) \in A \tag{14}$$

The constraints shown in equations 13 and 14 are identical to those in the minimum cost flow network problem.

The problem defined by DM can be solved using algorithms similar to those used in solving the MCFN. A modified version of SOLVE was created to solve DM. It starts by examining all arcs that are not currently a member of the set T. For each arc $(i,j) \notin T$ considered, the basic equivalent path (BEP) is traced to determine if the flow on arc $(p,q) \in IF$ would be affected if arc (i,j) is included in T. If arc (i,j) would not affect an arc in IF it is skipped and removed from further consideration. Otherwise, it is maintain in a candidate list for further examination and possible inclusion into T. This is identical to the approach used in the exchange algorithm.

The main difference between the exchange algorithm and the distance minimization algorithm is the determination as to when an arc should be pivoted into the tree. Once an arc (i, j) is selected for testing, the algorithm allows an arc to enter the tree and flows to change provided that the sum of the distance to feasibilities for the entire problem strictly decreases. This could cause the cardinality of IF to increase or decrease as arcs enter the tree. As the algorithm continues to strictly decrease the distance to feasibility, the flow on all arcs in IF are adjusted until a feasible solution is found indicated by $\sum_{(p,q)\in IF} min(x_{pq}, \ell_{pq} - x_{pq}) = 0$ or there are no other arcs to test. Enforcing the strictly decreasing requirement helps prevent cycling.

If any of the conditions are violated, a tabu value is assigned to (i, j) to prevent it from being considered for at least *tabutime* inclusion attempts. The value of $tabutime = 4$ was determined by experimentation to give the overall best results for arc selection while minimizing cycling.

The minimize distance to feasibility algorithm is shown in Figure 8. The routine CANDLISTIF is the same routine used in the exchange algorithm. The routine SOLVEIF2 is a modified version of SOLVE that uses all of the same tree management routines as SOLVE with the modified flow change rules previously described. SOLVEIF2 returns the variable $flowchange$ to indicate the amount of flow change in the current BEP. A value of 0 indicates that the flow in the BEP could not change and a pivot is not performed. The constant *tarcs* is set to the total number of arcs in the network and is used as part of the loop termination conditions.

3.5 Method H_4: Objective Function Improvement Algorithm

Once a feasible solution is found, the objective function improvement algorithm attempts to find better feasible solutions through the use of strategic oscillation and repeated application of the feasibility routines previously described.

The algorithm starts by relaxing the constraints 3 and 5 as previously described. The algorithm selects each arc $(i, j) \notin T$ and determines if adding flow to arc (i, j) would decrease the objective function in equation 1. Each arc is tested only once so that the relaxed solution does not move too far from feasibility. The routine that performs this operation is called SOLVEIMP and is similar to the SOLVE routine previously described except that SOLVEIMP does not solve the relaxed problem to optimality. The exchange algorithm, the distance minimization algorithm, and the force zero algorithm are executed to bring the solution back to feasibility. If the resulting feasible solution is better than the previous incumbent, it is set as the new incumbent and the process is repeated.

The improvement algorithm is shown in Figure 9. This algorithm terminates when the loop counter expires, the routine SOLVEIMP does not find a relaxed solution that is better than the current incumbent, or if a cycle is detected. The

Figure 8: Distance Minimization(A, IF, L, ZE,T, $\mathbf{X}$, z, $tarcs$) Algorithm

```
**Input:** $A$, $IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$, $tarcs$
**Output:** $IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$
  $loopcount \leftarrow 0$
  $candidate \leftarrow$ CANDLISTIF$(IF, L, ZE, A)$
  {Find an arc that could change flow of an infeasible arc}
  while $candidate \neq 0$ and $loopcount \leq tarcs$ and $\sum_{(i,j)\in IF} min(x_{ij}, \ell_{ij} - x_{ij}) \neq 0$ do
    $(IF, L, ZE, z, \mathbf{X}, flowchange) \leftarrow$ SOLVEIF2$(A, IF, L, ZE, \mathbf{X}, z)$
    if $flowchange = 0$ then
      $loopcount \leftarrow loopcount + 1$
      tabu level$_{pq} \leftarrow tabutime$
    else
      $loopcount = 0$
    end if
    $candidate \leftarrow$ CANDLISTIF$(IF, L, ZE, A)$
  end while
  Return ($IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$)
```

constant $improveloop = 10$ was determined by experimentation. It is found that in general, the best improvements, if any, where found in the first few cycles of the improvement loop.

3.6 The Interval-Pivoting Algorithm

The previous sections describe a collection of methods used to solve IFN using different approaches. The complete Interval-Pivoting heuristic, IFNET, is shown in Figure 10. The INITIALSOL routine provides an initial solution to the relaxed problem using the Big-M methodology.

4 Computational Experiments

This section describes an experimental design for comparing the effectiveness of an IFNET implementation with a commercial state-of-the-art solver. The experiments are designed to test the quality of solution and the time needed to find the solution. The software used and the problem set descriptions are detailed in the following sections.

Figure 9: Improvement(A, IF, L, ZE, T $\mathbf{X}$, $\mathbf{C}$, z, $tarcs$, incumbent) Algorithm

```
**Input:** $A$, $IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $\mathbf{C}$, $z$, $tarcs$, incumbent
**Output:** $IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$, incumbent
  $loopcount \leftarrow 0$
  $(T, z, \mathbf{X}) \leftarrow$ SOLVEIMP$(T, \mathbf{X}, z)$
  if $IF \neq \emptyset$ then
    while $loopcount \leq improveloop$ and $z <$ incumbent do
      ($IF$, $L$, $ZE$, $\mathbf{X}$, $z$) $\leftarrow$ Exchange($A$, $IF$, $L$, $ZE$, $\mathbf{X}$, $z$, $tarcs$)
      if $IF = \emptyset$ and $z <$ incumbent then
        incumbent $\leftarrow z$
      else
        ($IF$, $L$, $ZE$, $\mathbf{X}$, $z$) $\leftarrow$ Distance Minimization($A$, $IF$, $L$, $ZE$, $\mathbf{X}$, $z$,
        $tarcs$)
        if $IF = \emptyset$ and $z <$ incumbent then
          incumbent $\leftarrow z$
        else
          $(IF, T, \mathbf{X}, z) \leftarrow$ Force Zero$(A, N, T$, $\mathbf{X}$, $\mathbf{C}$, $\mathbf{LB}$, $feasloop)$
          if $IF = \emptyset$ and $z <$ incumbent then
            incumbent $\leftarrow z$
          end if
        end if
      end if
      if Cycle Detected then
        BREAK
      end if
      $loopcount \leftarrow loopcount + 1$
      $(T, z, \mathbf{X}) \leftarrow$ SOLVEIMP$(T, \mathbf{X}, z)$
      if $IF = \emptyset$ and $z <$ incumbent then
        incumbent $\leftarrow z$
        BREAK
      end if
    end while
  end if
  Return ($IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$, incumbent)
```

Figure 10: IFNET(A, N, $\mathbf{LB}$, $\mathbf{C}$) Algorithm

```
**Input:** $A$, $N$, $\mathbf{LB}$, $\mathbf{C}$
**Output:** incumbent
  incumbent $\leftarrow 0$
  $(T, z) \leftarrow$ INITIALSOL$(A, N)$
  $(T, z, \mathbf{X}) \leftarrow SOLVE(A, N, \mathbf{X}, \mathbf{C})$
  $(IF, \mathbf{C}^m) \leftarrow IFEAS(IF, \mathbf{C}^m, \mathbf{C}, \mathbf{X}, \mathbf{LB}, A)$
  **if** $IF = \emptyset$ **then**
    incumbent $\leftarrow z$
    BREAK
  **end if**
  $(IF, T, \mathbf{X}, z) \leftarrow$ Force Zero$(A, N, T,$ $\mathbf{X}$, $\mathbf{C}$, $\mathbf{LB}, feasloop)$
  **if** $IF = \emptyset$ **then**
    incumbent $\leftarrow z$
  **else**
    ($IF$, $L$, $ZE$, $\mathbf{X}$, $z$) $\leftarrow$ Exchange($A$, $IF$, $L$, $ZE$, $\mathbf{X}$, $z$, $tarcs$)
    **if** $IF = \emptyset$ **then**
      incumbent $\leftarrow z$
    **else**
      ($IF$, $L$, $ZE$, $\mathbf{X}$, $z$) $\leftarrow$ Distance Minimization($A$, $IF$, $L$, $ZE$, $\mathbf{X}$, $z$, $tarcs$)
      **if** $IF = \emptyset$ **then**
        incumbent $\leftarrow z$
      **end if**
    **end if**
  **end if**
  ($IF$, $L$, $ZE$, $T$, $\mathbf{X}$, $z$, incumbent) $\leftarrow$ Improvement($A$, $IF$, $L$, $ZE$, $T$ $\mathbf{X}$, $\mathbf{C}$,
  $z$, $tarcs$, incumbent)
  Return (incumbent)
```

4.1 Solution Codes

The commercial software used for comparison purposes is CPLEX version 10.0 from IBM. In testing, each code is given a solution time limit of 7200 seconds for each problem. Reported solution times are in CPU seconds for the solution process and do not include input, output, problem-generation, or report-generation time.

An additional routine outside of the timing parameters checks the final CPLEX solution for infeasible arcs. This is inserted after CPLEX returned a results that violated the conditional lower bound constraints.

The IFNET code is written in ANSI C and compiled using `gcc` with optimization option `-O3`. The SOLVE routine is based on the NETSTAR code developed by Barr [1, 6]. The solution tree data structure is maintained using the Extended Threaded Index method of [5]. Additional structures of length $|A|$ store the original problem data plus reduced costs, tabu number, flow, set T flag for each arc, augmented by forward-star and backward-star structures.

4.2 Problem Set Generation

A series of benchmark problems for testing interval-flow transshipment problems are generated using a two step process that required a slightly modified version of NETGEN [14] and a post-generation utility. NETGEN is a machine-independent program for the generation of random feasible network problems with pre-defined characteristics, including size, arc density, cost and requirement ranges, and number of transshipment source and sink nodes.

For interval-flow networks, the challenge is to create large-scale test problems that are feasible with respect to the conditional lower bounds. NETGEN creates pure network problems starting with a skeleton solution, which ensures feasible routing of supplies to demand nodes, then adds arcs randomly while to ensure a connected network. Since all lower bounds are assumed to be zero, the random addition of conditional lower bound constraints throughout the network could result in an infeasible problem.

Our process avoids this by only adding these constraints to non-skeletal arcs, thus preserving feasibility. (A statistical analysis determined that IFNET did not favor the skeleton arcs in finding feasible solutions.) There is user control over the percentage of arcs that have a conditional lower bound along with the minimum and maximum values of those bounds.

4.3 Problem Set Definition

Problem set A is designed to compare the solution time and quality of IFNET and CPLEX. An analysis of variance determines whether significant differences between the two codes exist in terms of run-time or final solution values.

The six test set factors and their levels are shown in Table 3. These factors are chosen after preliminary test results showed possible variations between the two codes when adjusting these factors. The test problem parameters are

Table 3: Test Set A: Problem Factors and Levels

Factor	Level 1	Level 2
Number of nodes	5,000	10,000
Number of arcs (nested within nodes)		
In 5,000-node problems	50,000	100,000
In 10,000-node problems	50,000	100,000
Percent source nodes	2%	5%
Percent demand nodes	2%	5%
Percent arcs with clb	25%	50%
Range of clb	50 to 100	100 to 200

shown in Tables 4 and 5. Each problem's total supply is fixed at 100,000 and the variable cost for each arc ranged from 0 to 50.[2]

4.4 Computational Results

All computer runs for both codes are performed on a single UNIX-based machine located at Southern Methodist University: a Dell PE 2650 with dual Intel Xeon processors running at 3.2 GHz with 4 gigabytes of random access memory. All tests are performed at similar times with similar system loading.

Tables 6 - 9 list the codes' performance for the specified test problems. Each table shows the best solution for each code; z_Δ, the difference between the IFNET and the CPLEX solutions divided by the best solution; the execution time of IFNET, the solution time for CPLEX, and the time multiple that is the ratio of CPLEX time to IFNET time.[3] CPLEX did not find a feasible solution for all problems and boldface solution values indicate that an infeasible solution is returned.

IFNET finds a feasible solution for every test problem and CPLEX fails to find a feasible solution to 17.2% of them. As shown in Tables 6 - 9, the 64 test problems can be organized by factor and level into 16 groups of four problems each. Figure 11 shows the percentage of feasible solutions for each test group that each code finds and the time required. It shows that IFNET found solutions for 100% of all test groups with times substantially faster than CPLEX. The results from CPLEX show that eight different test groups contained at least one test problem that CPLEX could not solve within the time limit specified. The times shown are average times for each test group.

The average z_Δ value was 0.00049, meaning that the solutions found by the two codes were extremely close in value. The average solution times were

[2]Preliminary testing showed that changing the variable cost range did not affect the solution quality or solution time. Preliminary testing also showed that fixing the total supply and varying the number of source nodes had the same effect as varying the total supply but keeping the number of source nodes fixed. This experimental design varies the number of source nodes as a factor.

[3]The time for CPLEX only includes the solution time as provided by the CPLEX timing parameter.

Table 4: Test Set A: Problem 1-8 Definitions

Test #	Number of Nodes	Number of Arcs	Source Nodes	Sink Nodes	% CLB Arcs	CLB Min	CLB Max
1a	5,000	50,000	2%	2%	25%	50	100
1b	5,000	50,000	2%	2%	50%	50	100
1c	5,000	50,000	2%	2%	25%	100	200
1d	5,000	50,000	2%	2%	50%	100	200
2a	10,000	50,000	2%	2%	25%	50	100
2b	10,000	50,000	2%	2%	50%	50	100
2c	10,000	50,000	2%	2%	25%	100	200
2d	10,000	50,000	2%	2%	50%	100	200
3a	5,000	100,000	2%	2%	25%	50	100
3b	5,000	100,000	2%	2%	50%	50	100
3c	5,000	100,000	2%	2%	25%	100	200
3d	5,000	100,000	2%	2%	50%	100	200
4a	10,000	100,000	2%	2%	25%	50	100
4b	10,000	100,000	2%	2%	50%	50	100
4c	10,000	100,000	2%	2%	25%	100	200
4d	10,000	100,000	2%	2%	50%	100	200
5a	5,000	50,000	5%	2%	25%	50	100
5b	5,000	50,000	5%	2%	50%	50	100
5c	5,000	50,000	5%	2%	25%	100	200
5d	5,000	50,000	5%	2%	50%	100	200
6a	10,000	50,000	5%	2%	25%	50	100
6b	10,000	50,000	5%	2%	50%	50	100
6c	10,000	50,000	5%	2%	25%	100	200
6d	10,000	50,000	5%	2%	50%	100	200
7a	5,000	100,000	5%	2%	25%	50	100
7b	5,000	100,000	5%	2%	50%	50	100
7c	5,000	100,000	5%	2%	25%	100	200
7d	5,000	100,000	5%	2%	50%	100	200
8a	10,000	100,000	5%	2%	25%	50	100
8b	10,000	100,000	5%	2%	50%	50	100
8c	10,000	100,000	5%	2%	25%	100	200
8d	10,000	100,000	5%	2%	50%	100	200

Table 5: Test Set A: Problem 9-16 Definitions

Test #	Number of Nodes	Number of Arcs	Source Nodes	Sink Nodes	% CLB Arcs	CLB Min	CLB Max
9a	5,000	50,000	2%	5%	25%	50	100
9b	5,000	50,000	2%	5%	50%	50	100
9c	5,000	50,000	2%	5%	25%	100	200
9d	5,000	50,000	2%	5%	50%	100	200
10a	10,000	50,000	2%	5%	25%	50	100
10b	10,000	50,000	2%	5%	50%	50	100
10c	10,000	50,000	2%	5%	25%	100	200
10d	10,000	50,000	2%	5%	50%	100	200
11a	5,000	100,000	2%	5%	25%	50	100
11b	5,000	100,000	2%	5%	50%	50	100
11c	5,000	100,000	2%	5%	25%	100	200
11d	5,000	100,000	2%	5%	50%	100	200
12a	10,000	100,000	2%	5%	25%	50	100
12b	10,000	100,000	2%	5%	50%	50	100
12c	10,000	100,000	2%	5%	25%	100	200
12d	10,000	100,000	2%	5%	50%	100	200
13a	5,000	50,000	5%	5%	25%	50	100
13b	5,000	50,000	5%	5%	50%	50	100
13c	5,000	50,000	5%	5%	25%	100	200
13d	5,000	50,000	5%	5%	50%	100	200
14a	10,000	50,000	5%	5%	25%	50	100
14b	10,000	50,000	5%	5%	50%	50	100
14c	10,000	50,000	5%	5%	25%	100	200
14d	10,000	50,000	5%	5%	50%	100	200
15a	5,000	100,000	5%	5%	25%	50	100
15b	5,000	100,000	5%	5%	50%	50	100
15c	5,000	100,000	5%	5%	25%	100	200
15d	5,000	100,000	5%	5%	50%	100	200
16a	10,000	100,000	5%	5%	25%	50	100
16b	10,000	100,000	5%	5%	50%	50	100
16c	10,000	100,000	5%	5%	25%	100	200
16d	10,000	100,000	5%	5%	50%	100	200

Table 6: Empirical Results: 5,000 nodes, 50,000 arcs

Prob	Final Objective Value			Time, (Seconds)		Time
#	IFNET	CPLEX	z_Δ	IFNET	CPLEX	Multiple
1a	2,842,530	2,842,530	0.0000	1.0	2.01	2.01
1b	2,845,713	**2,844,871**	0.0003	1.0	55.47	55.47
1c	2,843,523	2,842,538	0.0003	1.0	2.72	2.72
1d	2,854,087	2,846,294	0.0027	7.0	68.44	9.78
5a	2,845,254	2,844,277	0.0003	1.0	2.59	2.59
5b	2,852,427	2,847,437	0.0018	6.0	327.07	54.51
5c	2,859,129	2,854,210	0.0017	3.0	44.43	14.81
5d	2,903,090	2,871,747	0.0109	4.0	7,306.21	1,826.55
9a	3,139,780	3,137,985	0.0006	1.0	4.19	4.19
9b	3,155,629	3,145,132	0.0033	5.0	162.37	3.27
9c	3,148,602	3,142,321	0.0020	5.0	24.83	4.97
9d	3,187,297	3,158,699	0.0091	3.0	562.77	187.59
13a	2,409,963	2,408,044	0.0008	1.0	31.07	31.07
13b	2,430,406	2,419,552	0.0045	3.0	7,296.85	2,432.28
13c	2,429,500	2,423,287	0.0026	1.0	971.97	971.97
13d	2,498,761	2,460,795	0.0154	2.0	7,265.46	3,632.73

Boldface

indicates that an infeasible solution is returned

Table 7: Empirical Results: 10,000 nodes, 50,000 arcs

Prob	Final Objective Value			Time, (Seconds)		Time
#	IFNET	CPLEX	z_Δ	IFNET	CPLEX	Multiple
2a	5,527,735	5,527,328	0.0002	3.0	5.78	1.93
2b	5,563,667	5,544,184	0.0067	9.0	1,096.54	121.84
2c	5,530,289	5,529,420	0.0007	14.0	30.28	2.16
2d	5,670,711	5,584,685	0.0261	9.0	7,248.77	805.42
6a	4,450,433	4,448,363	0.0005	3.0	16.62	5.54
6b	4,520,343	4,483,040	0.0083	6.0	7,272.46	1,212.08
6c	4,469,606	4,464,982	0.0010	2.0	30.85	15.43
6d	4,735,590	4,611,682	0.0269	13.0	7,246.24	557.40
10a	5,081,454	5,076,958	0.0009	2.0	4.19	2.10
10b	5,138,118	**5,109,163**	0.0057	5.0	162.37	32.47
10c	5,097,934	5,087,033	0.0021	2.0	24.83	12.42
10d	5,364,896	5,218,763	0.0280	8.0	562.77	70.35
14a	4,546,788	4,544,516	0.0005	3.0	64.49	21.50
14b	4,671,573	4,600,026	0.0156	11.0	7,242.06	658.37
14c	4,583,220	4,564,462	0.0041	3.0	484.29	161.43
14d	5,048,887	**4,820,087**	0.0468	8.0	7,223.42	902.93

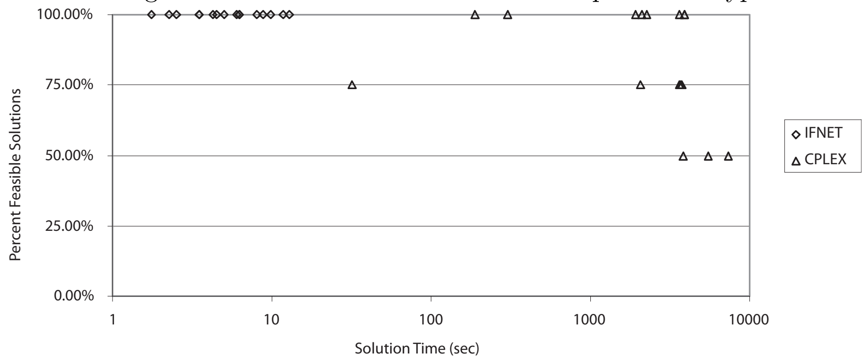
Table 8: Empirical Results: 5,000 nodes, 100,000 arcs

Prob	Final Objective Value			Time, (Seconds)		Time
#	IFNET	CPLEX	z_Δ	IFNET	CPLEX	Multiple
3a	1,828,313	1,826,769	0.0008	3.0	22.05	7.35
3b	1,831,667	1,828,620	0.0017	10.0	101.64	10.16
3c	1,836,415	1,831,473	0.0027	14.0	422.69	30.19
3d	1,848,919	1,836,218	0.0069	5.0	669.15	133.83
7a	1,510,774	1,510,414	0.0002	2.0	14.28	7.14
7b	1,518,295	1,514,705	0.0024	10.0	1,595.54	159.55
7c	1,518,705	1,515,548	0.0021	7.0	152.76	21.82
7d	1,551,475	1,532,385	0.0125	6.0	7,346.09	1,224.35
11a	1,660,313	1,659,650	0.0004	3.0	20.31	6.77
11b	1,664,416	1,661,668	0.0017	2.0	510.16	255.08
11c	1,667,578	1,664,049	0.0021	7.0	456.65	65.24
11d	1,699,788	**1,679,575**	0.0120	6.0	7,327.32	1,221.22
15a	1,486,097	1,484,687	0.0009	5.0	79.02	15.80
15b	1,498,415	1,490,120	0.0056	6.0	7,318.88	1,219.81
15c	1,497,159	1,489,071	0.0054	3.0	902.40	300.80
15d	1,544,898	1,513,453	0.0208	6.0	7,369.32	1,228.22

Table 9: Empirical Results: 10,000 nodes, 100,000 arcs

Prob	Final Objective Value			Time, (Seconds)		Time
#	IFNET	CPLEX	z_Δ	IFNET	CPLEX	Multiple
4a	3,019,560	3,017,350	0.0007	12.0	215,21	17.93
4b	3,041,750	**3,027,042**	0.0049	14.0	7,324.39	523.17
4c	3,028,933	**3,024,811**	0.0014	8.0	479.99	60.00
4d	3,117,512	3,059,584	0.0189	13.0	7,356.66	565.90
8a	2,191,788	2,190,132	0.0008	1.0	101.05	101.05
8b	2,221,209	2,206,196	0.0068	3.0	7,300.09	2,433.36
8c	2,238,187	2,212,035	0.0181	2.0	190.88	95.44
8d	2,394,119	**2,284,831**	0.0478	3.0	7,278.26	2,426.09
12a	2,964,529	2,959,156	0.0018	6.0	161.21	26.87
12b	3,011,434	2,975,091	0.0122	13.0	7,318.48	562.96
12c	3,006,934	**2,981,226**	0.0086	4.0	7,252.20	1,813.05
12d	3,191,883	**3,076,985**	0.0373	16.0	7,336.92	458.56
16a	2,570,734	2,554,133	0.0065	12.0	7,256.26	604.69
16b	2,637,668	**2,974,412**	-0.1277	11.0	7,503.13	682.10
16c	2,670,466	2,621,752	0.0186	9.0	7,238.00	804.22
16d	2,967,564	**3,958,349**	-0.3339	20.0	7,277.4	363.87

Figure 11: Percent of Feasible Solutions per Code Type



2,833 seconds for CPLEX and 6.06 seconds for IFNET, a ratio of 488.6. Hence, in aggregate, IFNET found comparable solutions nearly 500 times faster than CPLEX and was more successful in identifying feasible solutions.

4.5 Analysis of Results

A statistical analysis of the test results determines which factors, if any, affect solution time and solution quality. A second analysis identifies those factors related to differences in solution values returned by the two codes.

4.5.1 Analysis of Solution Time

An analysis of variance identifies those factors that affect the solution time. The factors considered are the code type (IFNET or CPLEX) the total number of arcs, the total number of nodes, the percent of arcs with conditional lower bounds, the number of source nodes, and the number of sink nodes.

All 64 observations for each code type are evaluated using the ANOVA procedure with SAS version 9.1 for UNIX with the significance level of 5%. The results show that there is a significant difference in the amount of time each code takes to solve the problems with a mean time of 3009.9 seconds for CPLEX and a mean time of 6.1 seconds for IFNET.

There is no significant difference in the solve time due to levels of the number of arcs. All other factors showed significant effects for solution time including interactions of code type with total nodes, code type with percent conditional lower bound arcs, code type with number of sink nodes, percent conditional lower bound arcs with number of source nodes, code type with the number of source nodes, and a third order interaction of the number of source nodes, number of sink nodes, and the total number of nodes.

4.5.2 Analysis of Solution Quality Due to Code Type

An analysis of variance was performed to determine which factors affect the solution quality. The factors considered are the code type (IFNET or CPLEX) the total number of arcs, the total number of nodes, the percent of arcs with conditional lower bounds, the number of source nodes, and the number of sink nodes. It is expected that the solutions will vary due to problem-size differences. The important factors for this analysis is to determine if the solutions vary due to the code type or any interaction effects with the code type.

Only the 53 observations for which CPLEX returns a feasible solution are used for this analysis. The analysis of variance indicates that there is no significant difference between the CPLEX solution values and the IFNET solution values. A Tukey grouping analysis shows the IFNET and CPLEX are in the same group. There are no significant interaction effects between code type and any other factor.

4.5.3 Analysis of Solution Quality Variation

Another analysis explores the effect of any factors on z_{Δ}, the difference between the solutions of the two codes taken as a percentage of the best solution. The factors considered are the total number of arcs, the total number of nodes, the percent of arcs with conditional lower bounds, the range of the conditional lower bound, the average supply per source node, the number of source nodes, and the number of sink nodes. Again only the 53 observations for which CPLEX returns a feasible solution are used.

This analysis shows that while the number of arcs, source nodes, and sink nodes has no statistically significant effect on solution quality metric z_{Δ}, all other factors do. A Tukey analysis of nodes identified a significant difference due to the total number of nodes, with the larger-node problems showing a higher variation.

4.6 Additional Test Sets

An additional problem set B was created to explore the limits of both codes. This test set consists of 16 test problems using the same parameters of tests 1d-16d but increases the percentage of arcs with conditional lower bounds from 50% to 75%. IFNET is able to solve every problem in this test set with an average solution time of 30.3 seconds. CPLEX is only able to solve three problems with an average solution time of 7,353.6 seconds (with a 7200-second time limit) and hence does not provide a sufficient number of data points for a statistical comparison.

Finally, problem set C probes the size limitations of the IFNET code: eight problems, with all combinations of 20,000 and 40,000 nodes, 500,000 and 1,000,000 arcs, and the percentage of arcs with conditional lower bounds at 25% and 50%. The total supply is fixed at 200,000, the number of source and sink nodes is fixed at 5% of the total nodes, and the range of the conditional lower bounds is

Table 10: Test Set C: Large Problem Results

Test #	Number of Nodes	Number of Arcs	% CLB Arcs	Final Objective Value	Time (Seconds)
1	20,000	500,000	25%	63,733,825	52.0
2	20,000	500,000	50%	63,760,333	36.0
3	20,000	1,000,000	25%	49,715,107	115.0
4	20,000	1,000,000	50%	49,729,141	68.0
5	40,000	500,000	25%	80,903,551	126.0
6	40,000	500,000	50%	80,966,762	140.0
7	40,000	1,000,000	25%	61,875,969	203.0
8	40,000	1,000,000	50%	61,923,193	151.0

set to be between 50 and 100 for all runs. IFNET found a feasible solution for all problems with an average time of 68 seconds for problems with 20,000 nodes and an average of 155 seconds for problems with 40,000 nodes. The results of these test runs are shown in Table 10. CPLEX did not find a valid solution for any of these test problems in 14,400 seconds.

5 Conclusions

This research has produced a new, fast, and effective heuristic and optimization code, IFNET, for solving interval-flow transshipment problems. Comparisons between IFNET and CPLEX, a state-of-the-art commercial optimizer show that IFNET is extremely fast and provides solutions comparable quality. On problem sets as large as 10,000 nodes and 100,000 arcs, IFNET finds high-quality feasible solution to all problems, including challenging instances with conditional lower bound arc densities up to 75%—for which CPLEX solves less than 20% of the instances.

On problem sizes beyond the capability of CPLEX with up to 40,000 nodes and 1 million arcs, the IFNET code quickly produced high-quality feasible solutions. Hence this new technology renders readily solvable the large-scale instances of this important problem class that are encountered practice.

References

[1] M. Amini and R. Barr. Network reoptimization algorithms: A statistically designed comparison. *ORSA Journal on Computing*, 5(4):385–409, 1993.

[2] A. Apte and R. Barr. Solving the interval-flow transportation problem: Exact and heuristic methods. *Technical Report 04-EMIS-01, SMU CSE Department*, 1999.

[3] Egon Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[4] Egon Balas. Disjunctive programming. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 years of integer programming 1958-2008 the early years and state-of-the-art surveys*, chapter 10, pages 283–340. Springer, Berlin London, 2010.

[5] R. Barr, F. Glover, and D. Klingman. Enhancements of spanning tree labelling procedures for network optimization. *INFOR*, 17:16–29, 1979.

[6] R. Barr and B. Hickman. Parallel simplex for large pure network problems: Computation testing and sources of speedup. *Operations Research*, 42:65–80, 1993.

[7] Richard S. Barr, Robert H. Jones, and Anthony Klinkert. An efficient optimization approach to designing large-scale hierarchical smart-grid data networks. Emis technical report, Southern Methodist University, Lyle School of Engineering, Dallas, TX 75275, May 2012.

[8] G. Bradley, G. Brown, and G. Graves. Design and implementation of large scale primal transshipment algorithms. *Management Science*, 24(1):1–34, 1977.

[9] Carlos Daganzo and Karen Smilowitz. Bounds and approximations for the transportation problem of linear programming and other scalable network problems. *Transportation Science*, 38:343–356, 2004.

[10] R. Dial. Network optimized road pricing: Part II: Algorithms and examples. *Operations Research*, 26:538–550, 1999.

[11] F. Glover, D. Karney, D. Klingman, and A. Napier. A computation study on the start procedures, basis change criteria, and solution algorithm for transportation problems. *Management Science*, 20(5):793–813, 1974.

[12] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[13] John Jarvis, Ronald Rardin, V. Unger, Richard Moore, and Charles Schimpeler. Optimal design of regional wastewater systems: A fixed-charge network flow model. *Operations Research*, 26:538–550, 1978.

[14] D. Klingman, A. Napier, and J. Stutz. NETGEN - a program for generating large scale (un)capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–822, 1974.

[15] J. Mulvey. Pivot strategies for primal simplex network codes. *Association for Computing Machinery Journal*, 25:266–270, 1978.

[16] Yves Pochet and Mathieu VanVyve. A general heuristic for production planning problems. *INFORMS Journal of Computing*, 16:316–327, 2004.

[17] Ronald Rardin. *Optimization in Operations Research.* Prentice Hall, Upper Saddle River, New Jersey, 2000.

[18] Nils Rudi, Sandeep Kapur, and David Pyke. A two-location inventory model with transshipment and local decision making. *Managment Science*, 47:1668–1680, 2001.

[19] Hanif D. Sherali and C. M. Shetty. *Optimization wiht Disjunctive Constraints*, volume 181 of *Lecture Notes in Economics and Mathematical Systems.* Springer-Verlag, Berlin, 1980.

[20] Warren Walker. A heuristic adjacent extreme point algorithm for the fixed charge problem. *Management Science*, 22:587–596, 1976.