Received: 21 May 2020 | Revised: 22 December 2020 | Accepted: 23 December 2020
DOI: 10.1002/net.22020

SPECIAL ISSUE ARTICLE

WILEY

An extreme-point tabu-search algorithm for fixed-charge network problems

Richard S. Barr[1] | **Fred Glover**[2] | **Toby Huskinson**[3] | **Gary Kochenberger**[4]

[1]Department of Engineering Management, Information and Systems, Lyle School of Engineering, Southern Methodist University, Dallas, Texas, USA
[2]ECEE, College of Engineering and Applied Science, University of Colorado - Boulder, Boulder, Colorado, USA
[3]Computer Science Department, Lyle School of Engineering, Southern Methodist University, Dallas, Texas, USA
[4]Business School, University of Colorado at Denver, Denver, Colorado, USA

Correspondence
Richard S. Barr, Department of Engineering Management, Information and Systems, Lyle School of Engineering, Southern Methodist University, Dallas, TX, USA.
Email: barr@smu.edu

Abstract
We propose a new algorithm for fixed-charge network flow problems based on ghost image (GI) processes as proposed in Glover (1994) and adapted to fixed-charge transportation problems in Glover et al. (2005). Our GI algorithm iteratively modifies an idealized representation of the problem embodied in a parametric GI, enabling all steps to be performed with a primal network flow algorithm operating on the parametric GI. Computational testing is carried out on well-known problems from the literature plus a new set of large-scale fixed-charge transportation and transshipment network instances. We also provide comparisons against CPLEX 12.8 and demonstrate that the new GI algorithm with tabu search (TS) is effective on large problem instances, finding solutions with statistically equivalent objective values at least 700 times faster. The attractive outcomes produced by the current GI/TS implementation provide a significant advance in our ability to solve fixed-cost network problems efficiently and invites its use for larger instances from a variety of application domains.

KEYWORDS
combinatorial optimization, discrete optimization, fixed-charge networks, heuristics, mixed-integer optimization, network optimization, nonconvex optimization, tabu search

1 | PROBLEM DEFINITION AND BACKGROUND

We define the network fixed-charge problem as

$$\begin{aligned} \textit{NetFC:} \qquad \text{Minimize } x_o[FC] &= cx + F(x) \\ \text{subject to: } Ax &= b \\ U \geq x &\geq 0, \end{aligned}$$

where x is the vector given by $x = \left(x_j : j \in N = \{1, \ldots, n\}\right)$ and the matrix A is a node-arc incidence matrix, so that the equation $Ax = b$ constitutes a classical network representation of the flow equations defining a pure network problem. The variables x_j correspond to integer flows on the network arcs with simple upper bounds U_j and the real vector c is the cost per unit of flow. The fixed-charge function is given by $F(x) = \sum(F_j y_j : j \in N)$, where each fixed-charge coefficient F_j is nonnegative and the y_j variables take on binary values that satisfy $y_j = 1$ if $x_j > 0$ and $y_j = 0$ otherwise. $F(x)$ may be equivalently written as $F(x) = \sum\left(F_j y_j : j \in N(FC)\right)$, where $N(FC) = \{j \in N : F_j > 0\}$ and we call $N(FC)$ the set of (effective) fixed-charge coefficients.

Applications of the problem *NetFC*arise in many areas, including facility location, network design, logistics and supply chain, and specific problems, such as lot-sizing, course scheduling, and others. Location problems include the uncapacitated and capacitated facility or plant location problems as described in Fernández and Landete [13] and Eiselt et al. [12]. Daskin [11] provides in-depth coverage of the area and an extensive list of application papers is available in Eiselt et al. [12]. Network design applications arise in telecommunications [14, 27], including related location problems [15], regional wastewater system design [21], and electrical smartgrid data network design, including equipment placement, described in Barr et al. [9].

NetFC problems also have useful applications in supply chain optimization [2], logistics [3], vanpool assignment [22], and distribution networks [24]. In addition, they emerge in multilevel lot-sizing within an MRP [28] and scheduling training courses [18]. See other applications enumerated in Nicholson and Zhang [26].

In the following, we make extensive reference to concepts for primal simplex algorithms for pure networks, including basis structure, basis equivalent paths, and pricing and pivoting operations. (For useful background information, see, for example, Ahuja et al. [1], Bazaraa et al. [10], and Murty [25].)

The remainder of this article is organized as follows. Section 4 introduces our ghost image (GI) approach for the network fixed-charge problem, *NetFC*, and gives a pseudocode for its main algorithm and associated routines, followed by an explanation of the procedure. The design for testing our algorithm and the computational results, together with a comparison involving outcomes obtained by applying the CPLEX MIP code [15], are presented in Section 7. As shown, the outcomes demonstrate significant advantages for our algorithm both in solution time and solution quality in solving large and challenging *NetFC*problems. Finally, Section 7 concludes the article, highlighting the implications of the computational results and identifying directions for future research.

2 | THE GI APPROACH

The general form of the GI approach derives from a collection of problem-solving principles detailed in Glover [16]. The GI terminology refers to idealized representations that may be viewed as a generalization of certain relaxation/restriction procedures of mathematical optimization and that incorporate aspects of penalty-based neural models.

Our focus on applying the GI framework to fixed-charge network problems builds on the work of Glover et al. [17] that studies an earlier version of the approach applied to the special case of fixed-charge transportation networks. We have extended the underlying adaptive memory framework and integrated it with a more general network optimization approach that solves problems beyond the transportation setting.

Within the pure network setting of *NetFC*, our method exploits the problem structure by introducing a nonnegative penalty vector $p = (p_j : j \in N)$ and an associated penalized cost vector given by $c(p) = (c_j + p_j : j \in N)$. The penalty vector p is determined by a self-adjusting parameterization to give the following parametric network linear programming relaxation of the fixed-charge problem

$$\begin{aligned} LP(p): \qquad \text{Minimize } x_o(p) &= c(p)x \\ \text{subject to: } Ax &= b \\ U \geq x &\geq 0. \end{aligned}$$

The parameterization defining p_j occurs by setting $p_j = F_j/v_j$, where v_j denotes a quantity that is systematically updated throughout the algorithm. Hence p_j allocates the fraction $1/v_j$ of the fixed cost F_j to the total cost of x_j. We apply the convention that a denominator v_j close to 0 (smaller than a chosen ε value) translates into setting $p_j = M$ provided $F_j > 0$, where M is a large positive number, and similarly a denominator v_j that exceeds M translates into setting $p_j = 0$. However, we will in several instances identify the p_j values directly without bothering to make reference to v_j. (Note, if $F_j = 0$ then automatically $p_j = 0$, regardless of the value of v_j, since $F_j = 0$ expresses the fact that x_j is not a fixed-charge variable. We also interpret the value of x_j to be 0 if this value is less than ε.)

In the case $p = 0$ (where all $p_j = 0$), we have $c(p) = c$, and obtain the simple linear programming relaxation

$$\begin{aligned} LP: \qquad \text{Minimize } x_o &= cx \\ \text{subject to: } Ax &= b \\ U \geq x &\geq 0. \end{aligned}$$

The method sketched in Glover [16] begins by solving LP, and then solves a succession of problems $LP(p)$ produced by progressively modifying and updating p_j in alternation with applying an improvement method for enhancing the solution to $LP(p)$, utilizing adaptive memory strategies from tabu search (TS) [19].

An outline of this approach can be described as follows. Each solution obtained throughout these steps is evaluated as a candidate for the best solution x^* currently found.

Step 0: Solve LP, yielding an optimal linear programming solution as a first candidate for x^*, and set $v = U$, to yield $p_j = F_j/v_j$ for $j \in N$.
Step 1: Solve $LP(p)$, yielding a solution $x = x'$.
Step 2: Starting from x', use restriction to obtain a refined solution and apply the TS improvement method to obtain a further refined solution $x = x''$.
Step 3: Update v as a function of its current value and x''. If a maximum allowed iteration is not reached, return to Step 2. Otherwise, terminate the algorithm with the best solution x^* at hand.

In our adaptation of the GI method to the present context, for simplicity we use the convention of identifying the value of the (nonlinear) fixed-charge objective function x_o[FC] for a given trial solution vector x (e.g., $x = x', x''$, and so forth) as x_o (hence, $x'_o = x_o$[FC:x'], $x''_o = x_o$[FC:x''], and so forth). It is important to keep in mind that in such cases x_o includes reference to the fixed-charge component of the objective function, with the sole exception of explicitly referring to the problem LP.

The values U_o and U_j^o defined below are used as *proxy* bounds for x_j that will be introduced to replace the original bound U_j in certain calculations of the algorithm. Apart from trial solution vectors, we maintain a locally optimal solution vector x^* and an overall ("global") best solution vector x^G, that is, $x_o^G(= x_o$[FC:x^G]) is the minimum of the x_o^* $(= x_o$[FCx^*]) values.

We first give a pseudocode for the main routines of our GI/TS method embodied in our FixNetGI code and then describe the rationale that explains the key steps.

2.1 | GI/TS algorithm

The algorithm requires setting the following user input parameters:

Search limits:
1. MaxIter: maximum inside loop iterations per invocation
2. MaxPass: number of diversification invocations required to terminate algorithm
3. MaxInsideImprove: number of consecutive nonimproving inside loop iterations that will trigger an exit from the inside loop
4. BadLuck: number of consecutive x^*-improvement failures that will trigger a diversification
5. OutOfLuck: number of consecutive nonimproving outside loop iterations that will trigger an exit from the outside loop

Updating v:
6. Alpha(i), i= 1 to 3: weighting factors, summing to 1, for updating v_j values. Weights: Alpha(1) for current x_j^*, Alpha(2) for current v_j value, and Alpha(3) for the historical mean$_j$ plus U_j^o as adjusted by Beta
7. Beta: weight for historical average associated with Alpha(3) and v_j update
8. MaxSol: when updating v_j, the maximum number of previous x_j^* values used to calculate mean$_j$ for the Alpha(3) term

Tabu control:
9. TabuTenure: pivots required before a leaving arc can reenter the network tree (LP basis)

Duplicate solutions:
10. LimMatch: limits the number of times a solution duplication occurs before triggering diversification
11. sLim: number of solutions saved for duplicate-solution checking
12. ZeroRefresh: number of diversifications performed that will trigger refreshing the duplicate-check solution list with all counts equal 0

The GI/TS algorithm, as defined in Algorithms 1 and 2, is supported by several subsidiary procedures to update v, control the descent and tabu phases, perform moves/pivots, check for duplicate solutions, and diversify the search.[1] These components are defined and discussed separately.

2.2 | Discussion of the GI/TS main routine

In the initialization step, Step 0, the original linear programming relaxation LP is solved, and its solution is saved as the first locally optimal solution x^*. In addition, to initiate alternative formulas for updating the parameter vector v, the constant U_o is initialized to be the largest x_j value obtained in solving LP. In addition, the solution value for each variable x_j is recorded in U_j^o.

[1]The algorithms use the notation "++" to denote incrementing the associated variable by 1. For example, $++x$ is equivalent to $x = x + 1$; when used in a test, such as $++x > y$, variable x is incremented prior to making the comparison with y.

Algorithm 1. Procedure GI/TS Main Routine: Steps 0 and 1 (of 4)

—Step 0: Solve LP and create initial v, p, and locally best solution x^*
Initialize parameters: JIter= 0, v_{iter} = MaxIter/4, Pass = 0, LastInsideImprove = 0, Zero(s) = (0, … , 0) for $s = 1, \dots$, sLim (i.e., Zero(s,j) = 0 for $j = 1, \dots, n$), nMatch = 0, Recover = 0, DoTabu = True, NumSol = 0, NoLuck = 0, BigM = large positive number, AscentTenure = DescentTenure = TabuTenure
Solve *LP*, save the solution as the first locally best solution x^* and identify the fixed-charge objective function value $x_o^* = x_o[FC : x^*]$
Save the scalar U_o as the largest flow value $x_j, j \in N(FC)$ in the solution to *LP*
Save individual values $U_j^o(\leq U_j) = x_j$ as the maximum flow (so far) for each arc $j \in N(FC)$
Set $v_j = U_j$ so that initially $p_j = F_j/U_j$, mean$_j = U_j$ for all $j \in N(FC)$
—Step 1: Create and solve LP(p) to get first test solution x'
Solve $LP(p)$ by reoptimization to get x' and identify the fixed-charge objective function value $x_o' = x_o[FC : x']$. NumSol = 1
Update $U_j^o = \max\{U_j^o, x'_j\}$, for each $j \in N(FC)$
if $x_o' < x_o^*$ **then**
    $x_o^* = x_o', x^* = x'$, set Descent = True and perform V_UPDATE
end if
Create the n-vector ZeroØ, where ZeroØ(j) = 1 if $x'_j = 0$ and $F_j > 0$ ($j \in N(FC)$), else ZeroØ(j) = 0
Set First = 1, Zero(1) = SumZeroØ= ZeroØ, and OutsideOK = True
Perform STEPS23 // *Remainder of the algorithm, Steps 2 and 3*
STOP

In Step 1, the problem *LP*(p) is solved for the first time by reoptimizing the solution obtained in Step 0 for the modified objective function of *LP*(p), to obtain a *LP* optimum solution, x'. The fixed-charge objective function value $x_o' = x_o$[FC:x'] for x' is calculated and x' replaces the locally best solution x^* if $x_o' < x_o^* (= x_o$[FC:x^*]). We continue to update the values U_j^o designated to maintain the maximum value attained by x_j for the first v_{iter} iterations.

2.3 | Discussion of the supporting procedures

The method contains several supporting procedures. The first is V_UPDATE (Algorithm 3), which updates the v_j values as a foundation for subsequently determining the p_j values that define the problem *LP*(p) and, if appropriate, the U_j^o values and the identity of the best solution found so far. This procedure is accessed by Algorithms 1 and 2 of the Main Routine and also by other supporting routines.

The DESCEND routine (Algorithm 4) is the first supporting procedure invoked by the main routine, to implement the choice of x_{j^*} as the incoming pivot variable and the associated x_{k^*} as the leaving variable. If the algorithm is in a descent phase (Descent = True and TabuTenure = DescentTenure), and if the value x_{oj^*} continues the descent ($x_{oj^*} < 0$), then the routine simply performs the PIVOTJSTAR (Algorithm 5) procedure which pivots in x_{j^*} and removes x_{k^*} from the basis tree, to produce the updated solution x'' and its fixed-charge objective x_o'', and updates U_j^o for variables along the basis exchange path. Once the descent ends, Descent is set to False, TabuTenure is set to AscentTenure, and a check is performed to see if the solution x'' (before updating by the basis exchange of x_{j^*} and x_{k^*}) improves on x^* ($x_o'' < x_{o*}$). In this case, x^* is updated as customary and the routine performs V_UPDATE (Algorithm 3), which updates the v_j values as a foundation for subsequently determining the p_j values that define the problem *LP*(P) and likewise performs PIVOTJSTAR.

On the other hand, when the DESCEND routine is invoked in the situation where Descent = False, the PIVOTJSTAR routine is immediately performed and if $x_o'' < x_{o*}$, then x^* is updated as before. (The value x_{oj^*} can be improving after the initial descent has concluded. Instead of bouncing in and out of successive descent and ascent phases, once the initial descent has concluded, all subsequent steps are treated as an "ascent tabu phase." However, TabuTenure is set to DescentTenure whenever an improving step occurs, and to AssetTenure otherwise.) Finally, Tabu(k^*) = InsideIter + TabuTenure for the variable x_{k^*} that leaves the basis tree and becomes nonbasic. These background observations lay a foundation for understanding the thrust of the main routines and will be supplemented by additional comments below explaining the DUPCHECK and DIVERSIFY routines.

In the preceding steps of the Main Routines, to investigate the potential for further improvement to the current solution x', the objective function coefficients of the variables with nonzero and zero values are set to their variable costs c_j and c_j+ BigM, respectively, in Step 2-Phase I (as a result of setting $p_j = 0$ and p_j = BigM in these two cases). This results in the specified form of *LP*(p), which is then solved by postoptimization, yielding x''. The main purpose of setting the cost of variables with the zero values in the trial solution to BigM is to maintain their values at zero during the current postoptimization process, and these variables alternatively could simply be handled by temporarily setting their upper bounds to 0 during this step.

Algorithm 2. Procedure STEPS23

— Main Routine continued, Steps 2 and 3
Set OutsideOK = True
while OutsideOK **do**
—(Execute the outside loop)
—Step 2: Improve the current solution x', move to local optimum x'', and then to TS improvement
—Phase I: Refine x' by LP Restriction
Set p: p_j = BigM if ZeroØ(j) = 1, else $p_j = 0$
Solve $LP(p)$ by reoptimization to get x'' (and x''_o)
If $x''_o < x^*_o$, then set Descent = True *(Recording of $x^* = x''$ will be handled later)*
If JIter $<v_{\text{iter}}$, update $U^o_j = \max\{U^o_j, x''_j\}$, for each $j \in N(FC)$
—Phase II:
Initialize parameters: Set InsideIter = TSImprove = DescentImprove = LastInsideImprove = 0, Descent = True, Improve = False, TabuTenure = DescentTenure
Set Tabu(j) = 0 for each $j \in N$
Set Aspire = $\min\{x''_o, x^*_o\}$. InsideOK = True
while InsideIter < MaxIter and InsideOK **do**
—(Execute inside loop)
++InsideIter, $j^* = k *= 0$
for all NB arcs $j \in N$ **do**
Compute x_{oj}, the change in the objective function $x''_o(= x_o[FC : x''])$ if x_j is pivoted into the basis (and one or more variables x_k are driven to their lower or upper bounds to become candidates to leave the basis). Restrict consideration to $j \in N$ satisfying Tabu(j) < InsideIter or satisfying the aspiration criterion of x_{oj}< Aspire$-x''_o$
Save the best arc $j^* = \arg\min(x_{oj}$: for j subject to the restriction above), and identify a leaving arc k^*. ($k^* = j^*$ if there is a "bound flip" where x^*_j leaves the basis at its opposite bound)
end for
Perform DESCEND to carry out the pivot and associated update for the choice of j^* and k^*
if (InsideIter − LastInsideImprove > MaxInsideImprove) **then**
InsideOK = False *(Exit the Inside Loop)*
end if
end while// *for the inside loop*
if ++JIter > MaxIter **then**
OutsideOK = False *(Exit the Outside Loop)*
end if
if Improve **then**
NoLuck = 0
else
if ++NoLuck = OutOfLuck **then**
OutsideOK = False, BREAK *(Exit the Outside Loop)*
else if NoLuck = BadLuck **then**
$v_j = \max\{U_o - v_j, 1\}$ for each $j \in N(FC)$ *(mini-diversification)*
If $x^*_o < {x_o}^G$ then update ${x_o}^G = x^*_o$ and $x^G = x^*$
x^*_o = BigM *(to assure LP(p) starts over to make a new local optimum x^*)*
end if
end if
—Create and solve LP(p) to get new test solution x' and check for duplications
Set $p_j = F_j/v_j$ for each $j \in N(FC)$
Solve $LP(p)$ by postoptimization to get x' and x'_o
Update $U_o = \max\{U_o, {x'}_j\}$ for each $j \in N(FC)$
if $x'_o < x^*_o$ **then**
Update $x^*_o = x'_o$ and $x^* = x'$
Perform V_UPDATE
end if
Create the n-vector ZeroØ, where ZeroØ(j) = 1 if F_j>0 and ${x'}_j = 0$, else ZeroØ(j) = 0
Perform DUPCHECK // *which may include DIVERSIFY*
end while// *for the outside loop*
—Step 3: Conclusion after exit Outside Loop
if $x^*_o < {x_o}^G$ **then**
${x_o}^G = x^*_o$ and $x^G = x^*$ and set BestPass = Pass
end if
STOP

Remaining variables that were positive in the solution to the previous $LP(p)$ problem receive their original costs c_j so that the solution will be evaluated relative to the original variable costs. Following the calculation of the fixed-charge objective function value for the resulting solution x'', the current locally best solution x^* is replaced by x'' if this new solution turns out to be better. In addition, in Phase I the value U_j, identifying the maximum value for each x_j throughout the first v_{iter} iterations, is updated.

Next, the Inside Loop is initiated within Phase II that executes a tentative pivot exploration process, where each nonbasic variable $x_j, j \in N$, is considered as a potential entering variable, and the candidates for the leaving variable, x_k, are identified, to determine the change x_{oj} in the fixed-charge objective function that would result if x_j were selected to enter the basis tree. The process is guided by a simple TS approach, where attention is restricted to $j \in N$ satisfying Tabu(j) < InsideIter or satisfying the aspiration criterion x_{oj} < Aspire $- x''_o$, conditions that are irrelevant initially but that become relevant based on updates in the DESCEND routine.

Algorithm 3. Procedure V_UPDATE

Input: $x^*, x_o^*, x_o^G, U_o, v$, Beta, JIter, MaxSol, Mean_j, NumSol
Output: v, x^G, x_o^G, Mean_j, NumSol

— *Update v*
++NumSol
$Y = \min\{\text{NumSol}, \text{MaxSol}\}, X = 1/Y$
for all arcs $j \in N(FC)$ **do**
    $\text{Mean}_j = (X)x_j^* + (1 - X)\text{Mean}_j$
    $\text{UMean} = \text{Beta}(\text{Mean}_j) + (1 - \text{Beta})U_o$
    $v_j = \text{Alpha}(1)\cdot x_j^* + \text{Alpha}(2)\cdot v_j + \text{Alpha}(3)\cdot \text{UMean}$
end for
if $x_o^* < x_o^G$ **then**
    $x_o^G = x_o^*,\ x^G = x^*$
end if
RETURN

At the completion of the tentative pivot explorations within the main algorithm, the variable x_j^* that yields the greatest reduction in the fixed-charge objective function is selected for pivoting to bring it into the basis. To further improve the current solution, the process returns to the tentative pivot exploration phase, using the current basis representation.

The Inside Loop ends once the current iteration, InsideIter, exceeds the maximum allowed number of iterations, MaxInsideImprove, beyond the last improvement of the locally best solution x^*. At the conclusion of the Inside Loop the Outside Loop continues by setting the counter NoLuck to 0 if the Inside Loop had succeeded in improving the locally best solution x^*. Otherwise NoLuck is incremented and if NoLuck = OutOfLuck the Outside Loop terminates to record the final global best solution x^G at Step 3. Barring this, if NoLuck = BadLuck, a "mini-diversification" step is initiated. Phase II proceeds to generate the current p vector based on the vector v, and then solves $LP(p)$ by postoptimization to obtain x'. If the fixed-charge objective function value $x_o' = x_o[\text{FC}{:}x']$ improves on x_o^* then x^* is updated and the V_UPDATE routine is executed. Finally, the DUPCHECK routine is executed, as elaborated in the following section, which may involve executing the DIVERSIFY procedure, to lay the foundation for the next iteration of the Outside Loop.

2.4 Supporting procedures

We first give the pseudocode for the supporting procedures (shown as Algorithms 4–7) used within the main routine, in the order in which they first appear in the main routine and in other supporting procedures.

Having discussed V_UPDATE and PIVOTJSTAR in the explanation of DESCEND earlier, it remains to discuss the supporting procedure DUPCHECK (Algorithm 6) and the DIVERSIFY procedure (Algorithm 7) that is invoked within it. The DUPCHECK routine is designed to check whether there are any duplications among the most recent ZeroØ vectors stored in Zero(s) for $s = 1$ to sLim. Since each ZeroØ vector identifies the variables x_j that equal 0 in a given solution (by setting ZeroØ(j) = 1), and setting these variables to 0 automatically determines the network solution that sets remaining variables to 1, a duplication in these vectors implies that the associated fixed-charge solutions are duplicated. DUPCHECK carries out a check for duplications (matches) by recording Zero(s) as a wraparound list, where the most recent ZeroØ vector is stored in Zero(First) and Zero(Last) is the ZeroØ vector recorded sLim iterations ago. The Zero(s) array starts from s = First until reaching s = sLim, and then continues at $s = 1$ until reaching s = First − 1. Then the new (now most recent) ZeroØ vector is recorded by writing over the oldest one in the location s = First − 1 and then First is updated by setting First = First − 1. (Special case: If First = 1 then the location First − 1 is sLim.) This device avoids having to write the vectors into a temporary array and then write them back into Zero(s) to allow Zero(s) to always go from $s = 1$ to sLim.

If the number of matches nMatch is found to exceed the limit LimMatch, the DIVERSIFY routine is executed that updates x^G if the current x^* improves upon it and if the DIVERSIFY routine has been invoked MaxPass times the algorithm stops. Otherwise the diversification proceeds by generating new f_j values based on the formula f_j = SumZeroØ(j)/Max, where SumZeroØ(j) counts the number of times $x_j = 0$ in a solution that produced a ZeroØ vector in the DUPCHECK routine and Max is the maximum of these SumZeroØ(j) values. The new v_j values are then determined by setting $v_j = \lceil f_j \cdot U_j \rceil$ if SumZeroØ(j) > Max/2 and otherwise setting $v_j = \max\{\lceil f_j \cdot U_j^o \rceil, 1\}$.

From this, the p_j values are determined by the usual formula $p_j = F_j/v_j$ as a basis for creating the problem $LP(p)$, which is then solved by postoptimization to obtain a solution x'. The locally optimal solution x^* starts again "from scratch" by setting $x^* = x'$, and the bounds U_j^o are updated in the customary way, along with establishing the ZeroØ vector as in the first step of the

Algorithm 4. DESCEND Algorithm

Input: x_o^*, x_o'', AllTSImprove, AscentTenure, Descent, DescentImprove, DescentTenure, DoTabu, LastInsideImprove, Improve, InsideIter, JIter, TSImprove
Output: x_o^*, x_o'', AllTSImprove, Aspire, Improve, InsideOK, Tabu(), TabuTenure, TSImprove

— *Continue pivoting to local optimum or execute ascent tabu phase*
if Descent = TRUE **then**
  if $x_{oj}^*<0$ **then** // *the Descent Phase continues to improve*
    Perform PIVOTJSTAR // *to pivot in j^* and remove k^* from the basis*
    Update x_o and set Aspire = $\min\{x_o^*, x_o''\}$
    ++DescentImprove
  else
    Descent = FALSE // *happens the first time that leave Descent Phase*
    TabuTenure = AscentTenure
    if $x_o<x_o^*$ **then**
      Improve = TRUE
      LastInsideImprove = InsideIter −1
      Update $x_o^* = x_o$ and $x^* = x$
      Perform V_UPDATE
    end if
    if DoTabu = FALSE **then**
      InsideOK = FALSE, Return // *Exit Inside loop*
    end if
    Perform PIVOTJSTAR
  end if
else // *Descent = FALSE and we are not in the TS phase*
  Perform PIVOTJSTAR
  if $x_{oj}^*<0$ **then**
    TabuTenure = DescentTenure
    if $x''<x_o^*$ **then**
      Improve = TRUE
      LastInsideImprove = InsideIter
      Update $x_o^* = x_o''$ and $x^* = x''$
      ++TSImprove and ++AllTSImprove
      Aspire = x_o^*
      Perform V_UPDATE
    end if
  end if
end if
Update Tabu(k^*) = InsideIter + TabuTenure
RETURN

main algorithm. Finally, the V_UPDATE routine is executed, and the arrays associated with ZeroØ are likewise reinitialized, to conclude the DIVERSIFY procedure.

In the event that Match is not True in the DUPCHECK procedure (and hence nMatch is not checked for exceeding LimMatch, and DIVERSIFY is not executed), then the DUPCHECK procedure updates values for tracking the algorithm's performance, assures that nMatch = 0, and updates the Zero(s) array in accordance with the explanation above.

In conjunction with the main routine, these supporting procedures complete the GI/TS algorithm. (Note that the algorithm contains no random components.)

3 | GI/TS COMPUTATIONAL TESTING

An implementation of the above GI/TS algorithm, our code FixNetGI, was built using the alternating-path primal network simplex methods and data structures described in References [4-6]. This solver is implemented in Fortran, compiled with gfortran

Algorithm 5. Procedure PIVOTJSTAR

Input: j^*, k^*, x, U^o
Output: x'', x''_o, U^o

— Pivot in arc j^*, remove k^* to create x''
Pivot in j^* and remove k^* from the basis tree (or perform a bound flip), yielding a new x'' and updating x''_o
As x'' is created, set $U^o_j = \max\{U^o_j, x''_j\}$ along the basis equivalent path
RETURN

Algorithm 6. Procedure DUPCHECK

Input: First, CheckDupnMatch, MaxRecover, Recover, sLim, sMax, SumZeroØ, Zero(), ZeroØ
Output: Last, MaxRecover, nMatch, sMax, SumZeroØ, Recover

—If Zero() has duplicate ZeroØ vectors, perform diversification
Set s = First and Match = False // *Set True if some Zero(s) = ZeroØ*
for CheckDup = 1 to sLim and Match = False **do**
  if Zero(CheckDup) = ZeroØ **then**
    Match = True // *Exit loop*
  else
    If ++s > sLim, then s = 1
  end if
end for
if Match = True **then**
  if ++nMatch > LimMatch **then**
    sMax = max{sMax, CheckDup} // *Record how far we had to go to find a match*
    Execute DIVERSITY
    nMatch = 0
  end if
else
  if nMatch > 0 **then**
    ++Recover, MaxRecover = max{Recover, MaxRecover}, nMatch = 0
  end if
  SumZeroØ = SumZeroØ + ZeroØ
  if First > 1 **then**
    Last = First − 1
  else
    Last = sLim
  end if
  Zero(Last) = ZeroØ and First = Last // *Replace Zero(Last)*
end if
RETURN

-O3, and tested under the Centos 6.10 version of the Linux operating system at Southern Methodist University. The test hardware is a Dell R720 with a Dual Six Core Intel Xeon @ 3.5 GHz with 252 GB of RAM[2] available to the runs, which are executed in single-thread mode.

To assess the performance of FixNetGI, computational comparisons in terms of solution quality and speed are made with the IBM commercial optimization software CPLEX 12.8 [20], running with default parameters except for specifying single-threaded execution mode and a time limit per problem. Since CPLEX is a general-purpose optimizer for linear and mixed-integer problems, the special-purpose heuristic approach of FixNetGI gives it major advantages. This comparison, however, is valuable because: no comparable solver for *NetFC* is available, CPLEX is widely used and respected by practitioners and researchers, and the comparison will indicate the heuristic's efficiency and solution quality for use on real-world industry problems of this type.

[2] FixNetGI memory requirements for a problem with n nodes and a arcs: $10n + 12a + a\cdot$sLim integer variables and $4a + n$ double-precision variables.

Algorithm 7. Procedure DIVERSIFY

Input: U^o, x_o^*, x_o^G, x^*, MaxPass, Pass, sLim, SumZeroØ(), ZeroRefresh
Output: $p, U^o, v, x^G, x_o^G, x', x_o', x^*, x_o^*$, BestPass, First, Zero(), ZeroØ
— Diversify search after reaching local optimum
if $x_o^* < x_o^G$ **then**
  $x^G = x^*$ and $x_o^G = x_o^*$ and set BestPass = Pass
end if
if Pass = MaxPass **then**
  STOP
end if
++Pass
Let Max = $\max\{$SumZeroØ(j), over $j \in N(FC)\}$
for all $j \in N(FC)$ **do**
  Let f_j =SumZeroØ(j)/Max
  if SumZeroØ(j)> Max/2 **then**
    $v_j = \lceil f_j U_j \rceil$
  else
    $v_j = \max\{\lceil f_j U_j^o \rceil, 1\}$
  end if
  $p_j = F_j / v_j$
end for
— Create and solve LP(p) to get new "first" test solution x'
Solve $LP(p)$ by postoptimization to get x' and x'_o
Begin x^* again from scratch to set $x^* = x'$ and $x_o^* = x_o'$
Update $U_j^o = \max\{U_j^o, x'_j\}$ for each $j \in N(FC)$
Create the n-vector ZeroØ, where ZeroØ(j) = 1 if F_j>0 and $x'_j = 0$, else ZeroØ(j) = 0
Perform V_UPDATE
Set First = 1 and Zero(1) = ZeroØ
Set Zero(s) = (0, …, 0) for s = 2 to sLim
if Pass is a multiple of ZeroRefresh **then**
  Also reinitialize SumZeroØ = (0, …, 0), but otherwise let SumZeroØ continue to accumulate
end if
RETURN

To test the effectiveness of the new solution approach, two problem test sets are used for benchmarking. The first is a collection of known problems from the literature and the second is a new suite of larger problems generated to explore the effects of problem characteristics on performance.

Since there are over a dozen tuning parameters for the heuristic, we performed preliminary testing to identify a single set of parameters to use for all computational results reported herein. Randomly selected values from assigned ranges were run on the test sets, giving varied results, but providing guidance as to what value ranges seemed appropriate. The following parameter settings are employed for all runs reported: MaxIter = 50, MaxPass = 10, MaxInsideImprove = 40, BadLuck = 5, OutOfLuck = 20, Alpha(1) = 0.3, Alpha(2) = 0.45, Alpha(3) = 0.25, Beta = 0.4, MaxSolLimit = 1000, TabuTenure = 10, LimMatch = 10, sLim = 10, and ZeroRefresh = 30.

3.1 | Test Set 1: Description

This first set of studied problems is drawn from the comprehensive FCTP testbed of Sun et al. [29] with a variety of problem dimensions and characteristics. The problems were originally created with a version of the well-known NETGEN random problem generator [7, 23], modified to include fixed costs on arcs.

These Test Set 1 problems have seven problem dimensions, eight fixed-cost ranges (or types, labeled A-H), and 17 randomly generated instances of each combination. See Table 1 for definitions of these characteristics.

Each test problem is a totally dense capacitated fixed-charge transportation problem with randomly distributed supplies and demands per Table 1(A) and with each arc randomly assigned a discrete variable cost between 3 and 8 plus a fixed cost in the associated range from Table 1(B).

TABLE 1 Test Set 1 problem characteristics: (A) dimensions, (B) fixed-cost range [29]

(A)		(B)	
Problem dimensions	**Total supply**	**Fixed-charge type**	**Fixed-charge range**
10 × 10	10 000	A	[50, 200]
10 × 20	15 000	B	[100, 400]
15 × 15	15 000	C	[200, 800]
10 × 30	15 000	D	[400, 1600]
50 × 50	50 000	E	[800, 3200]
30 × 100	30 000	F	[1600, 6400]
50 × 100	50 000	G	[3200, 12 800]
		H	[6400, 25 600]

TABLE 2 Test Set 1 solution results for small problems, type A

		CPLEX 12.8			FixNetGI			
Dimension	**Prob ID**	**Best Z**		**Time (s)**	**Best Z**	**Time (s)**	**Z-Ratio**	**Time-X**
10 × 10	N104	40 255	*	1.49	40 258	0.01	1.0001	114.62
10 × 10	N107	42 026	*	1.16	42 029	0.01	1.0001	116.00
10 × 20	N304	56 361	*	0.74	56 366	0.02	1.0001	32.17
10 × 20	N307	49 737	*	1.61	49 742	0.03	1.0001	59.63
15 × 15	N204	54 497	*	1.48	54 547	0.03	1.0009	49.33
15 × 15	N207	53 591	*	1.26	53 601	0.03	1.0002	43.45
10 × 30	N504	56 883	*	3.2	57 137	0.04	1.0045	78.05
10 × 30	N507	52 898	*	4.72	52 998	0.04	1.0019	134.86
50 × 50	N1004	162 863		7200.03	163 764	1.64	1.0055	4395.62
50 × 50	N1007	161 186		7200.00	162 386	0.56	1.0074	12 834.22
30 × 100	N2004	103 163		7200.00	104 204	0.57	1.0101	12 543.55
30 × 100	N2007	103 402		7200.00	104 340	0.55	1.0091	13 162.71
Average:		78 072		2401.31	78 448	0.29	1.0033	3630.35

*Solved to optimality.

A subset of the 896 original testbed problems was selected for computational experiments with the FixNetGI code, following the choices of Glover et al. [17]. For the six smallest problem sizes, two instances of type A were used for this experimentation. For the largest and most difficult 50 × 100 size, all 15 instances of each fixed-charge type (A-H) were included, for a total of 132 problems. Hence the focus is on mixed-integer programs with 50 000 binary variables.

3.2 | Test Set 1: Computational results and analysis

Table 2 describes the solution results for the 12 smaller problems tested. Shown are the dimensions of the transportation problem, the problem identifier, the best solution value found (Z^*) and CPU solution time for CPLEX 12.8 (run with a 7200-s time limit) and the FixNetGI code, the ratio of the two solvers' best solution values (Z-ratio = FixNetGI's x_o^G/CPLEX's z^*) and the CPLEX time as a multiple of the FixNetGI solution time (Time-X).

With these smaller problems, the heuristic's x_o^G solution values are within 0.1% of the CPLEX best, on average, and were identified an average of three orders of magnitude faster. Optimal solution values from CPLEX are indicated by "*" when solved exactly.

The bulk of the testing was focused on the more-difficult totally dense fixed-charge transportation problems with 50 source and 100 sink nodes, 50 000 arcs, supply of 50 000, and all fixed-charge ranges as described in Table 1(B). Table 3 summarizes the results from solving 15 problem instances from each of the eight fixed-charge ranges (A-H). Detailed computational results from these 120 problems are found in Tables 4–11.

The results on the larger problems underscore the effectiveness of the GI/TS algorithm. In every case, CPLEX did not run to completion and exited at the 7200-s time limit, while FixNetGI used an average of 1.11 s of CPU time. Although FixNetGI's solution values averaged 9% higher, these were identified 6000 times faster.

To evaluate these solvers' abilities to handle even more challenging problems, as found in industrial applications, a new problem set was created. The problems are not only larger, but the suite is structured to facilitate statistical analysis of problem characteristics.

TABLE 3 Test Set 1: Summary of difficult, large 50 × 100 problems, averages of 15 problems per fixed-charge type

Fixed-charge		CPLEX 12.8		FixNetGI			
Type	Range	Best Z	Time (s)	Best Z	Time (s)	Z-Ratio	Time-X[a]
A	50–200	165 809	7200.01	167 499	1.09	1.010	6589
B	100–400	175 337	7200.00	178 795	1.09	1.020	6614
C	200–800	193 422	7200.00	200 498	1.22	1.037	5917
D	400–1600	227 260	7200.00	241 310	1.09	1.062	6625
E	800–3200	289 470	7200.01	316 637	1.08	1.094	6675
F	1600–6400	405 351	7200.00	459 073	1.08	1.133	6674
G	3200–12 800	624 726	7200.00	731 128	1.19	1.170	6303
H	6400–25 600	1 046 011	7200.01	1 258 395	1.08	1.203	6664
Average:		390 923	7200.01	444 167	1.11	1.091	6508

[a] All CPLEX run times are 7200 s.

TABLE 4 Test Set 1: Solution results for larger, difficult problems, type A fixed costs in range [50, 200]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3001	50 × 100	A	165 214	166 974	1.12	1.011	6446
N3002	50 × 100	A	166 266	168 050	1.11	1.011	6516
N3003	50 × 100	A	167 095	168 503	1.16	1.008	6212
N3004	50 × 100	A	165 793	167 406	1.11	1.010	6516
N3005	50 × 100	A	166 360	168 106	1.07	1.010	6754
N3006	50 × 100	A	164 614	166 146	1.06	1.009	6818
N3007	50 × 100	A	166 007	167 552	1.11	1.009	6516
N3008	50 × 100	A	164 273	165 943	1.09	1.010	6618
N3009	50 × 100	A	165 641	167 421	1.07	1.011	6761
N300A	50 × 100	A	166 124	167 635	1.04	1.009	6930
N300B	50 × 100	A	167 103	168 913	1.05	1.011	6870
N300C	50 × 100	A	163 857	165 929	1.09	1.013	6624
N300D	50 × 100	A	164 909	166 494	1.10	1.010	6534
N300E	50 × 100	A	168 075	169 908	1.17	1.011	6138
Average:			165 809	167 499	1.09	1.010	6589

[a] All CPLEX run times are 7200 s.

3.3 | Test Set 2: Overview and experimental design

To explore still larger problems and the possible effects of problem structure on solution time and quality, an experimental design using randomly generated test problems was established. For this, the NETGEN problem generator [23], modified to include fixed charges, created a new structured suite of transportation and transshipment problems with up to 33 times as many nodes, 100 000 binary variables, and a variety of problem characteristics.

Test Set 2 consists of 96 problems, each generated with a different seed value, and with problem characteristics varied to enable a full-factorial experimental design. All combinations of five factors are used: number of problem nodes (500, 1000, 3000, and 5000), percentage of source and sink nodes (30%/70% for transportation, and 20%/20% for transshipment), number of arcs (10 000, 50 000, and 100 000), total supply (100 000 and 500 000), and fixed-cost range (20–200 and 1600–6400). All arcs have a fixed cost, a variable cost between 3 and 8, and an arc capacity from 200 to 1500 units. Transshipment sources and sinks are not used.

Tables 12 and 13 display Test Set 2's problem characteristics and solution results from the FixNetGI code and CPLEX 12.8, run with a 1-h time limit and a single CPU thread. Problem characteristics shown are problem identifier and the number of nodes, sources and sinks, arcs, total supply, and fixed-cost range. Solution results are: the best solution value found (Best Z) for each application, the ratio of these solution values for FixNetGI to CPLEX (Z-ratio), the solution time using FixNetGI, and the CPLEX time (3600 s in all instances) as a multiple of the FixNetGI solution time (CPLEX Time-X).

Summary performance statistics by problem size and structure are given in Table 14. In terms of solution quality between the two solvers, The FixNetGI solution values average 1.2% larger than CPLEX, but for 13 of the 96 problems FixNetGI solutions

TABLE 5 Test Set 1: Solution results for larger, difficult problems, type B fixed costs in range [100, 400]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3100	50 × 100	B	176 223	179 323	1.04	1.018	6943
N3101	50 × 100	B	174 779	178 546	1.07	1.022	6704
N3102	50 × 100	B	175 859	179 340	1.08	1.020	6667
N3103	50 × 100	B	176 296	179 287	1.15	1.017	6245
N3104	50 × 100	B	176 175	179 947	1.15	1.021	6283
N3105	50 × 100	B	175 673	179 081	1.06	1.019	6825
N3106	50 × 100	B	174 171	177 536	1.10	1.019	6545
N3107	50 × 100	B	175 253	178 562	1.13	1.019	6400
N3108	50 × 100	B	173 440	177 091	1.12	1.021	6457
N3109	50 × 100	B	174 661	178 350	1.06	1.021	6825
N310A	50 × 100	B	176 295	179 663	1.08	1.019	6691
N310B	50 × 100	B	176 731	180 122	1.06	1.019	6825
N310C	50 × 100	B	173 012	176 917	1.08	1.023	6698
N310D	50 × 100	B	174 555	177 726	1.07	1.018	6748
N310E	50 × 100	B	176 933	180 436	1.13	1.020	6349
Average:			175 274	178 757	1.09	1.020	6590

[a]All CPLEX run times are 7200 s.

TABLE 6 Test Set 1: Solution results for larger, difficult problems, type C fixed costs in range [200, 800]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3200	50 × 100	C	194 225	201 498	1.05	1.037	6844
N3201	50 × 100	C	193 288	200 823	1.12	1.039	6440
N3202	50 × 100	C	194 189	202 126	1.08	1.041	6660
N3203	50 × 100	C	193 755	200 250	1.12	1.034	6434
N3204	50 × 100	C	195 218	202 696	1.12	1.038	6406
N3205	50 × 100	C	193 750	200 086	1.06	1.033	6805
N3206	50 × 100	C	192 095	199 228	1.08	1.037	6679
N3207	50 × 100	C	192 863	199 989	1.06	1.037	6786
N3208	50 × 100	C	191 262	197 823	1.10	1.034	6545
N3209	50 × 100	C	192 371	199 614	1.06	1.038	6773
N320A	50 × 100	C	195 345	201 847	1.08	1.033	6679
N320B	50 × 100	C	195 428	202 049	1.06	1.034	6786
N320C	50 × 100	C	190 533	197 625	1.98	1.037	3640
N320D	50 × 100	C	192 668	199 815	2.15	1.037	3347
N320E	50 × 100	C	194 341	202 005	1.13	1.039	6377
Average:			193 365	200 427	1.23	1.037	6169

[a]All CPLEX run times are 7200 s.

are superior (Z-ratio less than 1), including some larger instances where CPLEX's Best Z is 30 times larger. Based on average Z-ratio, the heuristic's solution quality tends to be superior for transportation problems when compared with transshipment problems with the same number of nodes.

In terms of solution speed, CPLEX runs to the 1-h time limit in all cases. FixNetGI averages 10.1 s per problem, or 700 times faster than the 3600-s time limit for CPLEX, as shown in the CPLEX Time-X column of Table 14. These multiples are better for the smaller problems, but all multiples would be much larger if CPLEX had been allowed to run to optimality.

3.4 | Test Set 2: Computational results and statistical analysis

The structure of the test set enables rigorous statistical analysis of the relative performance of CPLEX and FixNetGI solvers in terms of solution values and solution time, and the effect of the five factors described above. SAS 9.2's analysis of variance procedure (ANOVA) and comparisons of means using Tukey's significant difference (TSD) test are employed to determine whether

TABLE 7 Test Set 1: Solution results for larger, difficult problems, type D fixed costs in range [400, 1600]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3300	50 × 100	D	228 374	241 643	1.05	1.058	6857
N3301	50 × 100	D	227 575	242 211	1.11	1.064	6498
N3302	50 × 100	D	228 110	242 684	1.10	1.064	6575
N3303	50 × 100	D	225 815	239 882	1.12	1.062	6440
N3304	50 × 100	D	229 561	244 426	1.16	1.065	6218
N3305	50 × 100	D	227 701	241 937	1.07	1.063	6710
N3306	50 × 100	D	226 219	239 843	1.08	1.060	6679
N3307	50 × 100	D	225 348	239 331	1.09	1.062	6636
N3308	50 × 100	D	224 414	236 798	1.10	1.055	6551
N3309	50 × 100	D	226 652	241 535	1.09	1.066	6630
N330A	50 × 100	D	231 382	244 641	1.05	1.057	6844
N330B	50 × 100	D	230 094	244 703	1.04	1.063	6916
N330C	50 × 100	D	224 210	238 289	1.06	1.063	6812
N330D	50 × 100	D	226 083	241 055	1.07	1.066	6729
N330E	50 × 100	D	227 364	240 667	1.13	1.059	6360
Average:			227 181	241 286	1.09	1.062	6614

[a]All CPLEX run times are 7200 s.

TABLE 8 Test Set 1: Solution results for larger, difficult problems, type E fixed costs in range [800, 3200]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3400	50 × 100	E	291 035	316 495	1.03	1.087	6970
N3401	50 × 100	E	289 261	316 734	1.07	1.095	6754
N3402	50 × 100	E	290 616	319 367	1.08	1.099	6667
N3403	50 × 100	E	284 639	310 945	1.12	1.092	6434
N3404	50 × 100	E	292 426	321 563	1.13	1.100	6389
N3405	50 × 100	E	290 940	318 012	1.05	1.093	6870
N3406	50 × 100	E	288 448	314 592	1.08	1.091	6698
N3407	50 × 100	E	284 681	311 924	1.09	1.096	6599
N3408	50 × 100	E	285 990	312 083	1.08	1.091	6654
N3409	50 × 100	E	289 127	316 983	1.07	1.096	6710
N340A	50 × 100	E	296 495	324 751	1.07	1.095	6754
N340B	50 × 100	E	293 248	320 955	1.06	1.094	6786
N340C	50 × 100	E	287 021	315 303	1.07	1.099	6716
N340D	50 × 100	E	288 295	313 506	1.08	1.087	6661
N340E	50 × 100	E	289 837	316 340	1.12	1.091	6457
Average:			289 359	316 647	1.08	1.094	6654

[a]All CPLEX run times are 7200 s.

the average results differed by solution method and whether factors affected the average results. The TSD procedure compares and ranks solver performance under the effect of different single-factor levels and treatment combinations. Specifically, we test hypotheses that the mean solution times and solution values are the same for both solvers and under different factor levels.

Based on the problem solution times and values in Tables 12 and 13, ANOVA shows a statistically significant difference in mean solution times between the CPLEX and FixNetGI codes. Hence, as expected, the mean solution speeds of the two solvers are statistically different, with FixNetGI being the faster. Statistical differences in time are also found between the four levels of problem node count, the two fixed-charge ranges, transportation and transshipment network structures, the three levels of number of problem arcs, and two levels of total supply and demand. Hence, all hypotheses of equivalent means are rejected when runtime is the performance metric.

However, when comparing solvers based on problem solution values (Z), the TSD test finds no statistically significant difference between the solvers. Therefore, while the mean Z-ratio for FixNetGI is slightly higher than CPLEX's, ANOVA

TABLE 9 Test Set 1: Solution results for larger, difficult problems, type F fixed costs in range [1600, 6400]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3500	50 × 100	F	406 610	462 061	1.05	1.136	6890
N3501	50 × 100	F	403 755	460 160	1.08	1.140	6667
N3502	50 × 100	F	405 202	459 936	1.09	1.135	6581
N3503	50 × 100	F	394 992	445 519	1.11	1.128	6475
N3504	50 × 100	F	409 471	464 457	1.09	1.134	6630
N3505	50 × 100	F	407 823	462 557	1.04	1.134	6923
N3506	50 × 100	F	403 233	450 885	1.07	1.118	6704
N3507	50 × 100	F	396 770	452 211	1.10	1.140	6534
N3508	50 × 100	F	402 621	457 526	1.09	1.136	6606
N3509	50 × 100	F	405 749	460 973	1.08	1.136	6642
N350A	50 × 100	F	415 374	464 597	1.06	1.119	6792
N350B	50 × 100	F	409 530	462 858	1.10	1.130	6575
N350C	50 × 100	F	405 979	459 980	1.07	1.133	6729
N350D	50 × 100	F	405 994	459 980	1.07	1.133	6735
N350E	50 × 100	F	407 160	462 399	1.09	1.136	6624
Average:			405 261	458 860	1.08	1.132	6658

[a]All CPLEX run times are 7200 s.

TABLE 10 Test Set 1: Solution results for larger, difficult problems, type G fixed costs in range [3200, 12 800]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3600	50 × 100	G	628 353	728 685	1.05	1.160	6851
N3601	50 × 100	G	623 633	728 390	1.07	1.168	6748
N3602	50 × 100	G	622 435	739 308	1.07	1.188	6742
N3603	50 × 100	G	606 551	706 872	1.11	1.165	6463
N3604	50 × 100	G	629 427	733 056	1.83	1.165	3934
N3605	50 × 100	G	627 022	729 120	2.07	1.163	3483
N3606	50 × 100	G	623 664	726 111	1.10	1.164	6569
N3607	50 × 100	G	609 916	718 671	1.11	1.178	6516
N3608	50 × 100	G	621 534	724 269	1.09	1.165	6630
N3609	50 × 100	G	623 355	738 275	1.06	1.184	6792
N360A	50 × 100	G	638 942	735 655	1.07	1.151	6735
N360B	50 × 100	G	632 751	744 229	1.07	1.176	6761
N360C	50 × 100	G	627 701	741 241	1.06	1.181	6812
N360D	50 × 100	G	627 689	741 241	1.06	1.181	6805
N360E	50 × 100	G	627 919	731 792	1.08	1.165	6698
Average:			624 467	731 302	1.20	1.171	6263

[a]All CPLEX run times are 7200 s.

shows that the mean solution values are not statistically different and the hypothesis of equality of mean solution values is not rejected. The two fixed-charge ranges do produce statistically different average solution values, as expected, but transportation and transshipment problems do not demonstrate statistically different values, nor do the numbers of problem arcs. Problems with 5000 nodes had mean solution values that are statistically different from those with 500 and 1000 nodes, but not those with 3000 nodes.

This combination of hypothesis outcomes validates the effectiveness and speed of the GI/TS algorithm as implemented in FixNetGI for these larger and more challenging problem types. With solution times three orders of magnitude faster than CPLEX while producing comparable objective function values, this approach advances the state-of-the-art for fixed-charge network problems and renders solvable large practical instances from industrial settings.

TABLE 11 Test Set 1: Solution results for larger, difficult problems, type H fixed costs in range [6400, 25 600]

PROB	Size	Prob type FC range	CPLEX 12.8 Best Z	FixNetGI Best Z	FixNetGI Time (s)	Z-Ratio	Time-X[a]
N3700	50 × 100	H	1 054 655	1 266 006	1.07	1.200	6754
N3701	50 × 100	H	1 041 146	1 263 578	1.07	1.214	6704
N3702	50 × 100	H	1 040 325	1 252 861	1.09	1.204	6636
N3703	50 × 100	H	1 018 972	1 239 035	1.10	1.216	6522
N3704	50 × 100	H	1 050 443	1 263 694	1.09	1.203	6593
N3705	50 × 100	H	1 053 995	1 263 791	1.07	1.199	6704
N3706	50 × 100	H	1 049 237	1 260 282	1.08	1.201	6661
N3707	50 × 100	H	1 022 451	1 229 135	1.10	1.202	6563
N3708	50 × 100	H	1 040 737	1 255 743	1.10	1.207	6528
N3709	50 × 100	H	1 041 100	1 255 976	1.08	1.206	6667
N370A	50 × 100	H	1 067 181	1 281 905	1.08	1.201	6685
N370B	50 × 100	H	1 061 167	1 280 281	1.08	1.206	6685
N370C	50 × 100	H	1 052 506	1 260 941	1.07	1.198	6761
N370D	50 × 100	H	1 052 254	1 260 941	1.07	1.198	6761
N370E	50 × 100	H	1 044 003	1 241 756	1.07	1.189	6735
Average:			1 045 394	1 257 851	1.08	1.203	6657

[a]All CPLEX run times are 7200 s.

3.5 | Test Set 2: Analysis of early CPLEX solutions

The CPLEX software also uses heuristics to identify promising solutions early in its search process before applying branching and cutting methods that lead to optimality. The termination criteria include finding a solution whose value is within a specified distance from optimality and reaching a user-defined time limit. It is possible to identify an optimal or near-optimal solution early in the process but spend significant time proving optimality or making incremental improvements.

To assess CPLEX's early progress, some insight can be found through a retrospective analysis of its logs from Test Set 2's 96 problems when run with a 3600-s time limit. We collected the following information: the initial integer feasible solution value (InitialZ), the first integer solution value for which elapsed time is shown (EarlyZ), and the time that EarlyZ was reported. These can then be compared with CPLEX's best solution value found in 1 h (BestZ) and FixNetGI's best solution value (GI Z) and runtime (GI time). Table 15 displays averages of these values for groups of 12 problems organized by number of nodes and transportation/transshipment structure.

The table shows by group, the number of problem nodes, numbers of problem sources and sinks, CPLEX's average initial integer solution value, ratio of InitialZ to BestZ, EarlyZ, ratio of EarlyZ to BestZ, time to EarlyZ, and the ratios of EarlyZ time to GI time and EarlyZ to GI Z. Also included is the average time to best solution for FixNetGI. The means of these averages are also given over all eight problem groups and show that the mean InitialZ is 1505 times larger than CPLEX's final solution value, mean EarlyZ is 1% larger than BestZ, and an average of 1115 s are required to identify EarlyZ.

In addition, CPLEX requires an average of 100 times longer to determine EarlyZ than FixNetGI requires to reach its final solution, and those CPLEX EarlyZ values are 65% larger than FixNetGI's final solution value. FixNetGI, which does not have a timeout stopping capability, identifies its best solution (x^G and x_o^G) in an average of 2.67 s (out of its average 10.13 s total runtime). Unfortunately, neither code knows at the time-to-best whether or not a better solution will be discovered later.

4 | CONCLUSIONS AND FUTURE DIRECTIONS

Statistical testing reveals that the FixNetGI code is not only dramatically faster than CPLEX in identifying its best solutions, but its mean solution quality is statistically equivalent to that of CPLEX. This implementation of the GI/TS algorithm makes it appropriate for applications requiring high-quality results quickly, as in time-critical logistics, military response, airline rescheduling, telecommunications and content-delivery network reconfiguration for demand fluctuations, and other near-real-time decision-making situations.

There are a variety of opportunities to improve the GI/TS algorithm in the future. The tabu-search procedure currently employed in the method is exceedingly simple, and a more advanced version may well enhance overall performance. Another conspicuous opportunity for future improvement will be to determine better parameters settings (e.g., based on problem size and network class). A related possibility for investigation is to shortcut the Inside Loop operation and solve $LP(p)$ more often,

TABLE 12 Test Set 2, 500- and 1000-node problem characteristics and solution results for FixNetGI and CPLEX 12.8

Prob	Nodes	Sources/ Sinks	Arcs (000s)	Supply (000s)	FC Range	FixNetGI Best Z	CPLEX Best Z	Z-Ratio	FixNetGI time (s)	CPLEX Time-X
1001	500	150/350	10	100	[20,200]	356 689	355 891	1.002	2.28	1582
1002	500	150/350	10	100	[1600,6400]	1 450 668	1 458 839	0.994	1.29	2793
1003	500	150/350	10	500	[20,200]	1 615 340	1 614 341	1.001	3.35	1075
1004	500	150/350	10	500	[1600,6400]	3 026 670	3 019 022	1.003	1.24	2903
1005	500	150/350	50	100	[20,200]	317 018	317 199	0.999	14.81	243
1006	500	150/350	50	100	[1600,6400]	1 233 074	1 228 705	1.004	6.30	572
1007	500	150/350	50	500	[20,200]	1 519 582	1 519 662	1.000	16.93	213
1008	500	150/350	50	500	[1600,6400]	2 475 879	2 472 508	1.001	7.64	471
1009	500	150/350	100	100	[20,200]	315 383	315 917	0.998	16.01	225
1010	500	150/350	100	100	[1600,6400]	1 242 415	1 230 644	1.010	5.78	623
1011	500	150/350	100	500	[20,200]	1 515 707	1 516 089	1.000	16.91	213
1012	500	150/350	100	500	[1600,6400]	2 507 125	2 493 600	1.005	6.11	590
1013	500	100/100	10	100	[20,200]	506 218	505 593	1.001	3.58	1006
1014	500	100/100	10	100	[1600,6400]	1 493 392	1 237 146	1.207	2.10	1713
1015	500	100/100	10	500	[20,200]	2 417 010	2 416 865	1.000	2.89	1245
1016	500	100/100	10	500	[1600,6400]	3 161 702	3 149 330	1.004	2.55	1410
1017	500	100/100	50	100	[20,200]	363 544	362 896	1.002	9.23	390
1018	500	100/100	50	100	[1600,6400]	1 193 942	916 022	1.303	3.94	913
1019	500	100/100	50	500	[20,200]	1 724 593	1 724 192	1.000	8.79	410
1020	500	100/100	50	500	[1600,6400]	2 472 404	2 363 545	1.046	4.14	869
1021	500	100/100	100	100	[20,200]	344 606	344 442	1.000	16.84	214
1022	500	100/100	100	100	[1600,6400]	946 404	821 025	1.153	6.28	574
1023	500	100/100	100	500	[20,200]	1 579 353	1 578 955	1.000	17.13	210
1024	500	100/100	100	500	[1600,6400]	2 120 325	2 106 602	1.007	9.52	378
1025	1000	300/700	10	100	[20,200]	423 114	419 652	1.008	2.61	1379
1026	1000	300/700	10	100	[1600,6400]	2 817 946	2 792 776	1.009	2.60	1387
1027	1000	300/700	10	500	[20,200]	1 848 984	1 847 206	1.001	2.35	1535
1028	1000	300/700	10	500	[1600,6400]	4 564 825	4 472 742	1.021	1.61	2232
1029	1000	300/700	50	100	[20,200]	359 472	358 373	1.003	7.32	492
1030	1000	300/700	50	100	[1600,6400]	2 615 272	2 607 964	1.003	8.08	446
1031	1000	300/700	50	500	[20,200]	1 582 610	1 581 089	1.001	8.58	420
1032	1000	300/700	50	500	[1600,6400]	3 803 147	3 773 611	1.008	7.76	464
1033	1000	300/700	100	100	[20,200]	338 193	337 842	1.001	15.25	236
1034	1000	300/700	100	100	[1600,6400]	2 168 455	2 144 094	1.011	13.77	261
1035	1000	300/700	100	500	[20,200]	1 558 965	1 557 745	1.001	16.83	214
1036	1000	300/700	100	500	[1600,6400]	3 592 581	3 568 389	1.007	16.32	221
1037	1000	200/200	10	100	[20,200]	655 125	652 786	1.004	4.99	722
1038	1000	200/200	10	100	[1600,6400]	2 798 754	2 202 916	1.270	2.56	1408
1039	1000	200/200	10	500	[20,200]	3 067 512	3 067 129	1.000	4.03	894
1040	1000	200/200	10	500	[1600,6400]	5 135 864	4 863 736	1.056	1.45	2488
1041	1000	200/200	50	100	[20,200]	424 763	421 677	1.007	7.58	475
1042	1000	200/200	50	100	[1600,6400]	2 004 296	1 580 543	1.268	5.41	666
1043	1000	200/200	50	500	[20,200]	1 903 514	1 903 031	1.000	13.61	265
1044	1000	200/200	50	500	[1600,6400]	3 439 539	3 318 684	1.036	5.84	616
1045	1000	200/200	100	100	[20,200]	385 023	383 094	1.005	18.33	196
1046	1000	200/200	100	100	[1600,6400]	1 840 723	1 406 015	1.309	9.36	385
1047	1000	200/200	100	500	[20,200]	1 677 809	1 677 451	1.000	22.68	159
1048	1000	200/200	100	500	[1600,6400]	3 197 222	2 914 185	1.097	8.86	406

TABLE 13 Test Set 2, 3000- and 5000-node problem characteristics and solution results for FixNetGI and CPLEX 12.8

Prob	Nodes	Sources/ Sinks	Arcs (000s)	Supply (000s)	FC Range	FixNetGI Best Z	CPLEX Best Z	Z-Ratio	FixNetGI time (s)	CPLEX Time-X
1049	3000	900/2100	10	100	[20,200]	659 133	650 375	1.013	2.76	1306
1050	3000	900/2100	10	100	[1600,6400]	7 733 243	7 642 712	1.012	1.98	1815
1051	3000	900/2100	10	500	[20,200]	2 396 668	2 391 344	1.002	2.39	1504
1052	3000	900/2100	10	500	[1600,6400]	10 099 152	10 064 444	1.003	2.12	1700
1053	3000	900/2100	50	100	[20,200]	498 714	494 887	1.008	12.13	297
1054	3000	900/2100	50	100	[1600,6400]	5 664 575	5 611 541	1.009	12.85	280
1055	3000	900/2100	50	500	[20,200]	1 818 914	1 816 890	1.001	14.22	253
1056	3000	900/2100	50	500	[1600,6400]	8 778 672	8 729 810	1.006	13.03	276
1057	3000	900/2100	100	100	[20,200]	455 864	454 198	1.004	22.16	162
1058	3000	900/2100	100	100	[1600,6400]	5 119 067	5 126 635	0.999	21.11	171
1059	3000	900/2100	100	500	[20,200]	1 715 184	1 713 425	1.001	23.56	153
1060	3000	900/2100	100	500	[1600,6400]	7 109 451	7 110 977	1.000	25.10	143
1061	3000	600/600	10	100	[20,200]	1 180 615	1 159 167	1.019	3.26	1103
1062	3000	600/600	10	100	[1600,6400]	8 011 095	7 545 095	1.062	2.18	1651
1063	3000	600/600	10	500	[20,200]	5 031 102	5 019 882	1.002	5.01	718
1064	3000	600/600	10	500	[1600,6400]	12 953 363	11 923 212	1.086	2.42	1490
1065	3000	600/600	50	100	[20,200]	692 841	675 280	1.026	9.76	369
1066	3000	600/600	50	100	[1600,6400]	6 398 952	4 697 047	1.362	9.15	393
1067	3000	600/600	50	500	[20,200]	2 716 655	2 703 913	1.005	10.36	347
1068	3000	600/600	50	500	[1600,6400]	8 666 228	7 987 438	1.085	9.69	371
1069	3000	600/600	100	100	[20,200]	562 672	545 123	1.032	15.31	235
1070	3000	600/600	100	100	[1600,6400]	5 849 454	5 230 491	1.118	15.90	226
1071	3000	600/600	100	500	[20,200]	2 287 102	2 277 315	1.004	16.73	215
1072	3000	600/600	100	500	[1600,6400]	7 638 972	7 031 009	1.086	16.26	221
1073	5000	1500/3500	10	100	[20,200]	878 096	871 688	1.007	2.76	1306
1074	5000	1500/3500	10	100	[1600,6400]	14 241 804	14 008 932	1.017	1.98	1815
1075	5000	1500/3500	10	500	[20,200]	2 806 918	2 796 959	1.004	2.39	1504
1076	5000	1500/3500	10	500	[1600,6400]	16 539 549	16 487 661	1.003	2.12	1700
1077	5000	1500/3500	50	100	[20,200]	646 918	648 049	0.998	12.13	297
1078	5000	1500/3500	50	100	[1600,6400]	10 034 153	10 419 983	0.963	12.85	280
1079	5000	1500/3500	50	500	[20,200]	2 119 350	6 903 430	0.307	14.22	253
1080	5000	1500/3500	50	500	[1600,6400]	12 157 360	12 408 107	0.980	13.03	276
1081	5000	1500/3500	100	100	[20,200]	578 204	573 823	1.008	22.16	162
1082	5000	1500/3500	100	100	[1600,6400]	8 781 707	8 697 678	1.010	21.11	171
1083	5000	1500/3500	100	500	[20,200]	1 927 148	1 921 606	1.003	23.56	153
1084	5000	1500/3500	100	500	[1600,6400]	10 903 122	296 140 690	0.037	25.10	143
1085	5000	1000/1000	10	100	[20,200]	1 617 523	1 594 130	1.015	3.56	1010
1086	5000	1000/1000	10	100	[1600,6400]	15 691 467	14 233 263	1.102	2.25	1599
1087	5000	1000/1000	10	500	[20,200]	6 619 232	6 607 723	1.002	7.93	454
1088	5000	1000/1000	10	500	[1600,6400]	21 746 227	20 078 528	1.083	3.01	1198
1089	5000	1000/1000	50	100	[20,200]	894 573	857 541	1.043	11.92	302
1090	5000	1000/1000	50	100	[1600,6400]	10 733 033	8 240 724	1.302	12.21	295
1091	5000	1000/1000	50	500	[20,200]	3 358 382	3 338 499	1.006	13.31	270
1092	5000	1000/1000	50	500	[1600,6400]	14 089 181	11 755 011	1.199	13.66	264
1093	5000	1000/1000	100	100	[20,200]	771 060	726 754	1.061	21.66	166
1094	5000	1000/1000	100	100	[1600,6400]	8 494 313	7 072 083	1.201	20.45	176
1095	5000	1000/1000	100	500	[20,200]	2 700 873	2 684 237	1.006	22.83	158
1096	5000	1000/1000	100	500	[1600,6400]	10 864 655	406 850 056	0.027	23.60	153

TABLE 14 Problem group and overall average Z-ratio, FixNetGI time, CPLEX time multiple

Group	Z-Ratio	FixNetGI time (s)	CPLEX Time-X
500-node transportation	1.001	8.221	958.4
500-node transshipment	1.060	7.250	777.7
1000-node transportation	1.006	8.589	773.8
1000-node transshipment	1.088	8.724	723.3
3000-node transportation	1.005	12.783	671.7
3000-node transshipment	1.074	9.670	611.8
5000-node transportation	0.861	12.783	671.7
5000-node transshipment	1.004	13.033	503.7
All	1.012	10.132	711.5

TABLE 15 Test Set 2, CPLEX average values for initial/early solutions, solution times, and comparisons with FixNetGI

Nodes	InitialZ	InitialZ: BestZ	EarlyZ	EarlylZ: BestZ	Time to EarlyZ (s)	EarlyZ time: GI time	EarlyZ: GI Z	FixNetGI time to best (s)
500-node transportation	78 613 523	50.0	1 471 696	1.006	46.1	9.56	1.00	0.20
500-node transshipment	12 397 367 792	9909.1	1 493 072	1.022	52.3	9.87	0.97	2.91
1000-node transportation	111 459 663	43.9	2 131 361	1.003	245.6	28.01	1.00	0.15
1000-node transshipment	2 967 146 722	1984.6	2 104 874	1.034	178.3	26.17	0.96	3.87
3000-node transportation	86 955 933	16.5	4 324 120	1.002	1974.3	195.58	1.00	0.16
3000-node transshipment	114 422 473	22.0	4 755 095	1.004	1656.5	155.13	0.94	5.58
5000-node transportation	83 376 753	8.2	31 014 631	1.002	2508.6	193.75	3.37	0.33
5000-node transshipment	113 820 110	11.9	40 432 228	1.007	2257.3	187.39	3.97	8.15
Average:	1 994 145 371	1505.8	10 965 885	1.010	1114.9	100.68	1.65	2.67

with the option of updating the solution each time by solving the restricted *LP* problem. Within the DUPCHECK procedure, the trade-offs between the sLim and the LimMatch values likewise invite examination, as do the values of the "alpha parameters" in V_UPDATE.

There is also great potential for enhancements to FixNetGI. Profiles of FixNetGI runs on Test Set 2 show that 88% of the run time is spent evaluating the nonbasic arcs in lines 18–21 of Procedure STEPS23. For a given nonbasic, the basis equivalent path is first evaluated to perform the ratio test; if the min ratio is positive, the path is retraced to determine x_{oj}. (Runtime for each operation averages 72% and 16%, respectively.) On average, the nonbasics were degenerate 79% of the time and, therefore, nonimproving. Rather than following the algorithm's steepest descent rule, candidate lists could help focus the effort on the improving arcs and arc subsets in the Inside Loop.

Moreover, this evaluation process is highly parallelizable and could take advantage of multiprocessing, much as codes like CPLEX do. A parallel version of FixNetGI (per Barr and Hickman [8]) would enable equitable comparisons with commercial optimizers that have come to rely on multithreading for speed improvements and would be a valuable future study.

The attractive outcomes produced by the current version of GI/TS embodied in FixNetGI provides a significant advance in our ability to solve fixed-cost network problems efficiently and motivates a study devoted to the solution of practical problems in multiple areas.

ACKNOWLEDGMENT

We are indebted to Doug Shier and Cole Smith for valuable and perceptive observations that have improved the quality of this article.

DATA AVAILABILITY STATEMENT

The test problems and related data used to produce the findings of this study are available from coauthor R. Barr upon request.

ORCID

Richard S. Barr https://orcid.org/0000-0002-1925-6642
Fred Glover https://orcid.org/0000-0001-6945-0438

REFERENCES

[1] R. Ahuja, T. Magnanti, and J. Orlin, *Network flows*, Prentice Hall, Upper Saddle River, NJ, 1993.

[2] M. Alizadeh, *Facility location in supply chain*, in *Facility location in supply chain facility location: Concepts, models, algorithms and case studies*, R. Z. Farahani and M. Hekmatfar, Eds., Springer-Verlag, Berlin, Germany, 2009, 473–504.
[3] S. A. Alumur, B. Y. Kara, and M. T. Melo, *Location and logistics*, in *Location science*, G. Laporte, S. Nickel, and F. Saldanha da Gama, Eds., Springer, New York, NY, 2015, 419–441.
[4] R. Barr, F. Glover, and D. Klingman, *Enhancements of spanning tree labeling procedures for network optimization*, INFOR. Inf. Syst. Oper. Res. **17** (1979), 16–34.
[5] R. S. Barr, J. Elam, F. Glover, and D. Klingman, *A network augmenting path basis algorithm for transshipment problems*, in *Extremal methods and systems analysis*, A. V. Fiacco and K. O. Kortanek, Eds., Springer-Verlag, New York, NY, 1980, 250–274.
[6] R. S. Barr, F. Glover, and D. Klingman, *The generalized alternating path algorithm for transportation problems*, Eur. J. Oper. Res. **2** (1978), 137–144.
[7] R. S. Barr, F. Glover, and D. Klingman, *A new optimization method for large scale fixed charge transportation problems*, Oper. Res. **29** (1981), 448–463.
[8] R. S. Barr and B. Hickman, *Parallel simplex for large pure network problems: Computational testing and sources of speedup*, Oper. Res. **42** (1994), 65–80.
[9] R. S. Barr, R. Jones, and A. Klinkert, *An efficient optimization approach to designing large-scale hierarchical smart-grid data networks, technical report*, Southern Methodist University, Department of Engineering Management, Information, and Systems, Dallas, TX, 2019.
[10] M. Bazaraa, J. Jarvis, and H. Sherali, *Linear programming and network flows*, 4th ed., Hoboken, NJ: Wiley Online Library, 2010.
[11] M. Daskin, *Network and discrete location: Models, algorithms, and applications*, 2nd ed., Wiley, New York, NY, 2013.
[12] H. A. Eiselt, V. Marianov, and J. Bhadury, *Location analysis in practice*, in *Applications of location analysis*, H. Eiselt and V. Marianov, Eds., Springer, New York, NY, 2015.
[13] E. Fernández and M. Landete, *Fixed-charge facility location problems*, in *Location science*, G. Laporte, S. Nickel, and F. Saldanha da Gama, Eds., Springer, New York, NY, 2015, 47–77. https://www.springer.com/us/book/9783030321765
[14] A. Forsgren and M. Prytz, *Telecommunications network design*, in *Handbook of optimization in telecommunications*, M. G. C. Resende and F. M. Pardalos, Eds., Springer, New York, NY, 2006, 269–290.
[15] B. Fortz, *Location problems in telecommunications*, in *Location science*, G. Laporte, S. Nickel, and F. Saldanha da Gama, Eds., Springer, New York, NY, 2015, 537–554.
[16] F. Glover, *Optimization by ghost image processes in neural networks*, Comput. Oper. Res. **21** (1994), 801–822.
[17] F. Glover, M. Amini, and G. Kochenberger, *Parametric ghost image processes for fixed-charge problems: A study of transportation networks*, J. Heuristics **11** (2005), 307–336.
[18] F. Glover, D. Klingman, and N. V. Phillips, *Network models in optimization and their applications in practice*, Vol **36**, Wiley-Interscience, New York, NY, 1992.
[19] F. Glover and M. Laguna, *Tabu search*, Kluwer Academic Publishers, Boston, MA, 1997.
[20] IBM CPLEX optimizer 2019, Available at https://www.ibm.com/analytics/cplex-optimizer.
[21] J. J. Jarvis, R. L. Rardin, V. E. Unger, R. W. Moore, and C. C. Schimpeler, *Optimal design of regional wastewater systems: A fixed-charge network flow model*, Oper. Res. **26** (1978), 538–550.
[22] L. Kaan and E. V. Olinick, *The vanpool assignment problem: Optimization models and solution algorithms*, Comput. Ind. Eng. **66** (2013), 24–40.
[23] D. Klingman, H. A. Napier, and J. Stutz, *NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems*, Manag. Sci. **20** (1974), 814–821.
[24] G. R. Mateus and Z. K. G. Patrocinio Jr., *Optimization issues in distribution network design*, in *Handbook of optimization in telecommunications*, M. G. C. Resende and F. M. Pardalos, Eds., Springer, New York, NY, 2006, 341–366.
[25] K. G. Murty, *Network programming*, Prentice-Hall, Inc, Upper Saddle River, NJ, 1992.
[26] C. D. Nicholson and W. Zhang, *Optimal network flow: A predictive analytics perspective on the fixed-charge network flow problem*, Comput. Ind. Eng. **99** (2016), 260–268.
[27] M. Pioro and D. Medhi, *Routing, flow, and capacity design in communication and computer networks*, Elsevier, San Francisco, CA, 2004.
[28] E. Steinberg and H. A. Napier, *Optimal multi-level lot sizing for requirements planning systems*, Manag. Sci. **26** (1980), 1258–1271.
[29] M. Sun, J. E. Aronson, P. G. McKeown, and D. Drinka, *A tabu search heuristic procedure for the fixed charge transportation problem*, Eur. J. Oper. Res. **106** (1998), 441–456.

How to cite this article: Barr RS, Glover F, Huskinson T, Kochenberger G. An extreme-point tabu-search algorithm for fixed-charge network problems. *Networks.* 2021;1–19. https://doi.org/10.1002/net.22020