Combining Exact and Heuristic Approaches for the Capacitated Fixed Charge Network Flow Problem

Mike Hewitt, George L. Nemhauser, Martin W.P. Savelsbergh

Milton H. Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205, U.S.A.

Abstract

We develop a solution approach for the fixed charge network flow problem (FCNF) that produces *provably* high-quality solutions *quickly.* The solution approach combines mathematical programming algorithms with heuristic search techniques. To obtain high-quality solutions it relies on neighborhood search with neighborhoods that involve solving carefully chosen integer programs derived from the arc-based formulation of FCNF. To obtain lower bounds, the linear programming relaxation of the path-based formulation is used and strengthened with cuts discovered during the neighborhood search. The solution approach incorporates randomization to diversify the search and learning to intensify the search. Computational experiments demonstrate the efficacy of the proposed approach.

1 Introduction

The Fixed Charge Network Flow (FCNF) problem is a classic discrete optimization problem in which a set of commodities has to be routed through a directed network. Each commodity has an origin, a destination, and a quantity. Each network arc has a capacity. There is a fixed cost associated with using an arc and a variable cost that depends on the quantity routed along the arc. The objective is to minimize the total cost. Two versions of the problem are considered: commodities have to be routed along a single path and commodities can be routed along multiple paths. Many real-life instances of FCNF (or of instances of models that contain FCNF as a substructure) are very large (see for example Powell and Sheffi (1989)) - so much so that sometimes even the linear programming relaxation of the natural arc-based integer programming formulation can be intractable. In such situations, we can use a path-based integer programming formulation, but that necessitates the use of column generation techniques. Furthermore, the linear programming relaxation of the arc-formulation and the path-formulation have the same optimal value which is known to be weak. Hence, it may not just be difficult to find feasible solutions, it may also be challenging to determine the quality of these solutions. As a result, even though much research has been devoted to FCNF, exact methods are only capable of handling small instances, far smaller than many realistic-sized instance. Fortunately, in today's dynamic business environment getting high-quality solutions in a short amount of time is usually

more important than getting provably optimal solutions. Hence the focus of our research is to develop a solution approach that produces *provably* high-quality solutions *quickly.*

Our solution approach relies heavily on linear and integer programming to take advantage of the power of commercially available linear and integer programming solvers. Algorithmically, we combine mathematical programming techniques with heuristic search techniques. More specifically, we develop

- a primal local search algorithm using neighborhoods that involve solving carefully chosen integer programs, and
- a scheme to generate dual bounds that involves strengthening the linear programming relaxation via cuts discovered while solving these integer programs.

Our approach tightly integrates the use of the arc-based formulation of FCNF and the path-based formulation of FCNF. It also incorporates randomization to diversify the search and learning to intensify the search.

The resulting solution approach is very effective. For instances with 500 nodes, with 2000, 2500 and 3000 arcs, and with 50, 100, 150, and 200 commodities, we compared the quality of the solution produced by our solution approach with the best solution found by CPLEX after 15 minutes of computation and after 12 hours of computation. On average, the solution we found in less than 15 minutes is 35% better than CPLEX' best solution after 15 minutes and 20% better than CPLEX' best solution after 12 hours. Furthermore, we find a better solution than CPLEX' best solution after 15 minutes within 1 minute, and CPLEX' best solution after 12 hours within 3 minutes. On these instances the approach produces dual bounds that are 25% stronger than the LP relaxation. We also compared the quality of the solutions produced by our solution approach with the quality of the solutions produced by a recent implementation of the tabu search algorithm of Ghamlouch et al. (2003). For nearly all instances in their test set, our solution is better than the solution of the tabu search algorithm and this solution is found much faster.

The key characteristics of our solution approach for FCNF, which differentiate it from existing heuristic approaches, are:

- it uses exact methods to find improving solutions,
- it generates both a primal solution and a dual bound at each iteration, and
- it uses both the arc and path formulations of FCNF to guide the search.

The remainder of the paper is organized as follows. In Section 2, we briefly review some relevant literature. In Section 3, we present the arc formulation and the path formulation of FCNF. In Section 4, we introduce the main ideas and a high-level overview of our solution approach. In Section 4.1, we review the components of the solution approach that are geared towards improving a known solution. In Section 4.2, we focus on the techniques

used to strengthen the lower bound. In Section 4.3, we outline the scheme for finding an initial solution. Finally, in Section 5 we present the results of an extensive computational study.

2 Literature

Metaheuristics have been developed that find good primal solutions to instances of FCNF. A tabu search algorithm using pivot-like moves in the space of path-flow variables is proposed in Crainic et al. (2000). The scheme has been parallelized in Crainic and Gendreau (2002). A tabu search algorithm using cycles that allow the re-routing of multiple commodities is given in Ghamlouch et al. (2003). This cycle-based neighborhood is incorporated within a path-relinking algorithm in Ghamlouch et al. (2004).

Our heuristic solves carefully chosen integer programs to improve an existing solution. As such, it considers exponential-sized neighborhoods similar to very large-scale neighborhood search (VLSN) (Ahuja et al. (2002)). However, in contrast to VLSN, no polynomial-time algorithm exists for searching these neighborhoods. General integer programming heuristics such as local branching (Fischetti and Lodi (2003)) and relaxation-induced neighborhood search (Danna et al. (2005)) also are integer-programming based local search algorithms, but they are different from ours.

The dual side of the problem is studied in Crainic et al. (2001), using various lagrangean relaxations and solution methods. Although lagrangean-based heuristics, such as the one proposed in Holmberg and Yuan (2000) generate dual bounds, these bounds are no better than the value of the LP Relaxation. To the best of our knowledge, no heuristic produces dual bounds that are stronger than the LP relaxation.

Combining exact and heuristic search techniques, a key characteristic of our approach, has received quite a bit of attention in recent years, see for example De Franceschi et al. (2006), Schmid et al. (2008), Archetti et al. (2008), and Savelsbergh and Song (2008).

3 Formulations

Before describing the main ideas of the proposed solution approach, we present the arc-based formulation and the path-based formulation for FCNF. Let $D = (N, A)$ be a network with node set N and directed arc set A. Let K denote the set of commodities, each of which has a single source $s(k)$, a single sink $t(k)$, and a quantity d^k that must be routed from source to sink. Let f_{ij} denote the fixed cost for using arc (i, j), c_{ij} denote the variable cost for routing one unit of flow along arc (i, j) and u_{ij} denote the capacity of arc (i, j). We use variables x_{ij}^k to indicate the fraction of commodity k routed along arc (i, j) and binary variables y_{ij} to indicate whether arc (i, j) is used or not. The arc-based formulation

of FCNF (Arc-FCNF) is:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}(d^k x_{ij}^k) + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

subject to

$$\sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(j,i) \in A} x_{j,i}^k = \delta_i^k \quad \forall i \in N,\ \forall k \in K, \tag{1}$$

$$\sum_{k \in K} d^k x_{ij}^k \leq u_{ij} y_{ij} \ \ \forall (i,j) \in A, \tag{2}$$

$$y_{ij} \in \{0,1\} \ \ \forall (i,j) \in A. \tag{3}$$

If a commodity's demand must follow a single path we have

$$x_{ij}^k \in \{0,1\} \quad \forall k \in K,\ \forall (i,j) \in A, \tag{4}$$

otherwise, we have

$$0 \leq x_{ij}^k \leq 1 \quad \forall k \in K,\ \forall (i,j) \in A. \tag{5}$$

The objective is to minimize the sum of fixed and variable costs. Constraints (1) ensure flow balance, where δ_i^k indicates whether node i is a source ($\delta_i^k = 1$), a sink ($\delta_i^k = -1$) or an intermediate node ($\delta_i^k = 0$) for commodity k. Constraints (2) are the coupling constraints that ensure that an arc is used if and only if its fixed charge is paid and that the total flow on the arc does not exceed its capacity. It is well-known that Arc-FCNF has a weak LP relaxation and can be strengthened by disaggregating the coupling constraints to

$$x_{ij}^k \leq y_{ij} \quad \forall k \in K,\ \forall (i,j) \in A. \tag{6}$$

The resulting formulation has a tighter LP relaxation, but comes at the expense of many more constraints.

Next, we consider the path-based formulation of FCNF. Let variable x_p^k denote the fraction of commodity k that uses path p and let $P(k)$ denote the set of feasible paths for commodity k. We will consider instances where $P(k)$ is not known in full. Hence let $\bar{P}(k) \subseteq P(k)$ denote a subset of all feasible paths for commodity k. The path-based formulation of FCNF (Path-FCNF) is:

$$\min \sum_{k \in K} \sum_{p \in P(k)} (\sum_{(i,j) \in p} c_{ij}) d^k x_p^k + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

subject to

$$\sum_{p \in P(k)} x_p^k = 1 \quad \forall k \in K, \tag{7}$$

$$\sum_{k \in K} \sum_{p \in P(k):(i,j) \in p} d^k x_p^k \leq u_{ij} y_{ij} \ \ \forall (i,j) \in A, \tag{8}$$

$$y_{ij} \in \{0,1\} \ \ \forall (i,j) \in A. \tag{9}$$

We again have either $x_p^k \in \{0,1\}$ or $0 \leq x_p^k \leq 1$ depending on whether a commodity's demand may be split across multiple paths.

As before, the objective is to minimize the sum of fixed and variable costs. Constraints (7) ensure that every commodity is routed through the network and the coupling constraints (8) ensure that fixed charges are paid and that arc capacities are respected. Here too we can disaggregate the coupling constraints:

$$\sum_{p \in P(k):(i,j) \in p} x_p^k \leq y_{ij} \ \ \forall k \in K, \ \ \forall (i,j) \in A. \tag{10}$$

4 Solution Approach

At the heart of the primal side of our solution approach is a neighborhood search procedure. Consider the arc-based formulation of FCNF. By selecting a suitably small subset of the variables (and assuming all other variables are fixed) a tractable integer program can be defined and solved using an IP solver. Hopefully, a better solution to the whole problem is obtained. The process can be repeated multiple times by choosing different subsets of variables. A pseudo-code describing this process is given in Algorithm 1.

Algorithm 1 Neighborhood Search

```
while the search time has not exceeded a prespecified limit T do
  Choose a subset of variables V
  Solve the IP defined by variables in V
  if an improved solution is found then
    Update the global solution
  end if
end while
```

Note that the neighborhood in Algorithm 1 is defined by an integer program and searched using an integer programming solver. The key to making this neighborhood search scheme work is in the choice of the subsets of variables V.

This approach can be used on extremely large instances because the algorithm never requires the full instance to be in memory. To evaluate the quality of the solution produced by the neighborhood search we find lower bounds on the value of the primal solution using a path-based formulation since solving the LP relaxation of the arc-based formulation may require too much memory. The number of variables in the path-based formulation is huge, but variables can be considered implicitly rather than explicitly. The LP relaxation is

solved over a subset of the variables and a pricing problem is solved to determine whether there is a need to expand the set of variables or not. The pricing problem is a shortest path problem and relies on the dual values associated with the constraints of the path-based formulation. A well-known observation related to this column generation process is that a valid lower bound can be obtained at every pricing iteration. Specifically, let z^*_{LP} be the value of the solution to the LP and let $\bar{c}^k_p$ be the reduced cost of an optimal path p for the pricing problem for commodity k. Then $z^*_{LP} + \sum_{k \in K} \bar{c}^k_p$ is a lower bound for the value of the LP when all columns are considered.

Note that any solution of Arc-FCNF can be converted to a solution of Path-FCNF and vice versa. As a result, a solution to the IP based on a subset of arc variables V can be converted to a set of variables for Path-FCNF. It is highly likely that several of these variables do not appear in the path formulation yet. In this way, the neighborhood search generates variables for Path-FCNF. Similarly, a solution to Path-FCNF can be converted to a solution of Arc-FCNF and can thus be used to guide the choice of subset V in the neighborhood search.

4.1 Neighborhood Search

During the neighborhood search, we solve smaller integer programs defined by a subset of arcs in the network or a subset of commodities in the instance. We define a $SOLVER$ to be one of the two small integer programs together with a selection method for the necessary subset. A high-level implementation of a solver is given in Algorithm 2. A high-

Algorithm 2 Solver Template

Require: A feasible solution F to the full FCNF instance
Require: The set of all feasible paths P found so far
  Select Subset
  Solve Subproblem
  Construct Feasible Solution $F^{'}$ to full FCNF instance
  Determine new paths $P^{'}$ found
  Determine new cuts $C^{'}$ found in course of solving subproblem
  return (F', P', C')

level implementation of the neighborhood search is given in Algorithm 3. We currently use a stopping criterion of whether the search time has exceeded a prespecified limit T. Given that we have multiple solvers to choose from, we need a scheme for choosing a solver at each iteration. We use a scheme similar to the *roulette wheel* approach of Ropke and Pisinger (2006) (see also Pisinger and Ropke (2007)), i.e., we randomly pick solvers with the probability of choosing a solver favoring ones that have recently given improvement. Specifically, if C_t is the cost of the feasible solution F_t provided to solver S at iteration t and $C^{'}_t$ is the cost of the feasible solution $F^{'}_t$ found by S, then we calculate the improvement

Algorithm 3 Neighborhood Search

Find an initial feasible solution
Set best_solution = initial feasible solution
Set $PATHS$ = paths from initial feasible solution
Set $SOLVERS$ = set of solvers we wish to consider using
Set $CUTS = \emptyset$
while not done **do**
  Solve path formulation LP only considering $PATHS$ but with $CUTS$ to get dual bound
  Select solver S from $SOLVERS$
  if solution found by S better than best_solution **then**
    Set best_solution = solution found by S
  end if
  Add new paths p from S to $PATHS$
  Take new cuts (π, π_o) from S, lift and add to $CUTS$
end while

for solver S in that iteration as $D_t^S = C_t - C_t'$ and in iteration T, we give S the score

$$E_S = \sum_{t=1}^{T-1} \frac{1}{2^{T-t}} D_t^S.$$

The larger the value E_S is, the more effective solver S has been in improving solutions in recent iterations. Assuming m solvers, ordered such that $E_{S_i} \geq E_{S_{i+1}}$ for $i = 1, ..., m-1$, we assign selection probabilities

$$\pi_i = \frac{m - (i-1)}{\sum_{j=1}^{m} m - (j-1)}, \; i = 1, ..., m.$$

We do not use a simple proportional scheme so that solver S_i can be chosen even when $E_{S_i} = 0$, and to ensure that the selection is not biased too greatly in favor of solvers that have been effective in the last iterations.

4.1.1 Arc Subset Solves

Given a subset of arcs $A' \subseteq A$, the IP defined by A' is Arc-FCNF with A' replacing A. Observe that if A' contains the arcs associated with the current best solution, then we can seed the arc subset IP with that solution and ensure that we only find solutions at least as good as the current best. Therefore, our schemes for choosing a subset A' start from the subset of arcs associated with the current best solution. Also, our schemes do not choose arcs directly but choose paths, thus ensuring that every arc chosen exists in some feasible

solution. In fact, the schemes choose paths from among the paths that are part of the path-based formulation that is used to compute a dual bound.

We introduce randomization to diversify our choices of paths as described in Algorithm 4.

Algorithm 4 *RandomizedPathSelection*$(n,\ PATHS,\ s)$

Require: Desired number of arcs n, set of paths $PATHS$, scores s_p for each $p \in PATHS$
  Assign probability π_p to each $p \in PATHS$ such that $s_p \geq s_q \rightarrow \pi_p \geq \pi_q$
  $ARCS = \emptyset$
  while $n \geq 0$ **do**
    Randomly choose path p based on probabilities π_p
    $n = n - |Arcs(p)|$
    $ARCS = ARCS \cup Arcs(p)$
  end while
  return $ARCS$

Scheme 1. This scheme selects paths that often appear in improving solutions, but are not in our current best solution. Scheme 1 is given in Algorithm 5.

Algorithm 5 Arc selection – Scheme 1

Require: Desired number of arcs n
  set $ARCS =$ arcs in best solution found so far
  for all $p \in PATHS$, p not in best solution **do**
    Set $s_p = \#$ of times path p appears in the set of paths P' returned by a solver
  end for
  $ARCS = ARCS \cup RandomizedPathSelection(n - |ARCS|,\ PATHS,\ s)$
  return $ARCS$

Scheme 2. This scheme selects paths that appear in the optimal solution to the LP relaxation of the path-based formulation. Let $x_p^{k,LP}$ represent the value of variable x_p^k in the most recent solution to the LP relaxation of our path-based formulation. Scheme 2 is given in Algorithm 6.

Algorithm 6 Arc selection – Scheme 2

Require: Desired number of arcs n
  Set $ARCS =$ arcs in best solution found so far
  Set $s_p = x_p^{k,LP}\ \ \forall p \in PATHS$
  $ARCS = ARCS \cup RandomizedPathSelection(n - |ARCS|,\ PATHS,\ s)$
  return $ARCS$

Scheme 3. This scheme selects the path produced for each commodity by the pricing problem. If these paths do not generate enough arcs, we revert to Scheme 2. Scheme 3 is given in Algorithm 7.

Algorithm 7 Arc selection – Scheme 3

Require: Desired number of arcs n
  Set $ARCS =$ arcs in best solution found so far
  Set $CG\_PATHS = \emptyset$
  for all k **do**
    Solve pricing problem for commodity k
    if new path $p_k^{'}$ **then**
      Add $p_k^{'}$ to CG_PATHS with score $s_{p_k^{'}} = |\bar{c}_{p_k^{'}}|$, the absolute value of its reduced cost
    end if
  end for
  $ARCS = ARCS \cup RandomizedPathSelection(n - |ARCS|,\ CG\_PATHS,\ s)$
  if $|ARCS| < n$ **then**
    Set $s_p = x_p^{k,LP}\ \ \forall p \in PATHS$
    $ARCS = ARCS \cup RandomizedPathSelection(n - |ARCS|,\ PATHS,\ s)$
  end if

4.1.2 Commodity Subset Solves

Given the current solution $\bar{x}$ and a subset of commodities J, let $A^{'} = \{(i,j) \in A \ \mid \ \bar{x}_{ij}^k = 0\ \forall k \notin J\}$, i.e., the set of arcs that is *not* used by commodities *not* in J. Let

$$\tilde{f}_{ij} = \begin{cases} f_{ij} & (i,j) \in A^{'} \\ 0 & (i,j) \notin A^{'} \end{cases}$$

and $\tilde{u}_{ij} = u_{ij} - \sum_{k \notin J} d^k \bar{x}_{ij}^k$. Then we solve an Arc-FCNF only considering commodities in J and with $\tilde{f}_{ij}$ instead of f_{ij} and $\tilde{u}_{ij}$ instead of u_{ij}. That is, if there exists a commodity not in J that already uses an arc, we allow the commodities in J to use that arc for free (in terms of the associated fixed cost charge), but we make sure that the available capacity on the arc reflects the flow already on it.

Constructing a feasible solution to the full FCNF instance starting from the solution to the IP requires updating the paths used by the commodities in J and the appropriate y_{ij} variables for $(i,j) \in A^{'}$. Note that since we do not have variables y_{ij} for $(i,j) \in A \setminus A^{'}$ our subproblem cannot "turn off" an arc used by a commodity not in J. We can again seed the IP with the paths used by the commodities in J in the current best solution.

Again, we introduce randomization to diversify our choices of commodities as described in Algorithm 8.

Algorithm 8 *RandomizedCommoditySelection*$(n,\ PATHS,\ s)$

Require: A set of commodities K, set S of scores s_k for each commodity k, number n of commodities to choose

Assign probability π_k to each $k \in K$ such that $s_k \geq s_l \rightarrow \pi_k \geq \pi_l$

Randomly draw n commodities from K based on probabilities π

Scheme 1. This scheme selects commodities whose paths in the current best solution use arcs (i, j) for which the reduced cost of variable y_{ij}, denoted by $\bar{f}_{ij}$, in the most recent LP solution are far from 0. In an LP solution complementary slackness implies $y_{ij} = 1$ only if $\bar{f}_{ij} = 0$. Thus we re-route commodities away from arcs that are far away from the complementary slackness condition. Scheme 1 is given in Algorithm 9.

Algorithm 9 Commodity selection – Scheme 1

Require: Desired number of commodities n

For arc (i, j), let $\bar{f}_{ij}$ be the reduced cost of variable y_{ij} in the most recent LP solution

Sort arcs in descending order of $\bar{f}_{ij}$

for all (i, j) in sorted list **do**

  for all $k \in K$ **do**

    if commodity k uses arc (i, j) in current best solution **then**

      Select commodity k

      if selected n commodities **then**

        break

      end if

    end if

  end for

end for

Scheme 2. Suppose in the current best solution $(\bar{x}, \bar{y})$, there is a node v with three commodities $C1, C2, C3$ entering on a single arc, but leaving on three different arcs (see Figure **??**). When the arcs leaving node v are not fully used, these three commodities are good candidates for re-optimization. (Of course, a similar situation occurs when we have multiple commodities entering on different incoming arcs, but leaving on a single outgoing arc.) Assume we have a pre-defined utilization threshold $UTIL\_THRESHOLD$, where we calculate the utilization of an arc for the current best solution $(\bar{x}, \bar{y})$ as

$$UTIL_{ij} = \frac{\sum_{k \in K} \bar{d}^k x_{ij}^k}{u_{ij}}.$$

Then we can score each node based on the number of under-utilized inbound and outbound arcs in the current best solution. Given those scores, we generate probabilities for selecting

a node v and hence for selecting a set of commodities. Scheme 2 is given in Algorithm 10.

Algorithm 10 Commodity selection – Scheme 2

```
for all nodes v do
  Set node_score_v = 0
  for all inbound arcs (w, v) do
    Calculate UTIL_{w,v}
    if 0 < UTIL_{w,v} < UTIL_THRESHOLD then
      Set node_score_v = node_score_v + 1
    end if
  end for
  for all outbound arcs (v, w) do
    Calculate UTIL_{v,w}
    if 0 < UTIL_{v,w} < UTIL_THRESHOLD then
      Set node_score_v = node_score_v + 1
    end if
  end for
end for
Generate node probabilities π_v such that node_score_v ≥ node_score_w → π_v ≥ π_w
Randomly choose a node v
Choose commodities k such that in solution (x̄, ȳ), ∃w such that x̄^k_{v,w} > 0 or x̄^k_{w,v} > 0
```

Scheme 3. Since we want to find sets of commodities that are likely to share arcs, we search for commodities whose paths in the current best solution are close together. Given a randomly chosen node v, we perform breadth-first search with v as root. As we discover nodes visited by commodities k in our current best solution, we mark the commodities. Scheme 3 is given in Algorithm 11.

Scheme 4. Suppose the network is very large and there are two commodities k and l that have only one feasible path from their source to their sink, say p_k and p_l. In that case, it is unlikely that these paths have a common arc hence re-optimizing commodities k and l together is not likely to lead to improvements. Conversely, if k and l have many paths, then they are likely to share arcs and thus are good candidates to optimize together. During the neighborhood search a collection of feasible paths $\bar{P}(k)$ is generated for each commodity k. We use these paths in Scheme 4, which is given in Algorithm 12.

Scheme 5. So far the selection schemes have been geared towards improving the current best solution. The purpose of this scheme is to provide diversification for our arc subset solvers. If there is a commodity k for which we have only generated a single path so far, then an arc subset solver will have little flexibility for re-routing it. Therefore, Scheme 5,

Algorithm 11 Commodity selection – Scheme 3

Require: Desired number of commodities n
  Set $COMMODITIES = \emptyset$
  Randomly choose a node v
  nodes.push(v)
  while nodes $\neq \emptyset$ and $|COMMODITIES| < n$ **do**
    v=nodes.pop()
    for all k such that $x^k_{w,v} > 0$ or $x^k_{v,w} > 0$ for some w **do**
      $COMMODITIES = COMMODITIES \cup \{k\}$
      if $|COMMODITIES| = n$ **then**
        break
      end if
    end for
    for all inbound arcs (w, v) and outbound arcs (v, w) **do**
      nodes.push(w)
    end for
  end while

Algorithm 12 Commodity selection – Scheme 4

Require: Desired number of commodities n
  Let $\bar{P}(k)$ be the set of paths generated for commodity k so far
  Set $s_k = |\bar{P}(k)|$
  return $RandomizedCommoditySelection(n,\ K,\ s)$

given in Algorithm 13, selects commodities for which we have generated very few paths.

Algorithm 13 Commodity selection – Scheme 5

Require: Desired number of commodities n
  Let $\bar{P}(k)$ be the set of paths generated for commodity k so far
  Set $s_k = -1 * |\bar{P}(k)|$
  return $RandomizedCommoditySelection(n,\ K,\ s)$

Scheme 6. The final scheme is specifically designed for instances of FCNF in which a commodity must be routed along a single path. The scheme selects commodities whose demand is split across many paths in the most recent solution to the LP relaxation of the path-based formulation. In a sense, we are "repairing" commodities that violate the single path constraint in the solution to the LP relaxation of the path-based formulation. Scheme 6 is given in Algorithm 14.

Algorithm 14 Commodity selection – Scheme 6

Require: $x_p^{k,LP}$ - the value of variable x_p^k in the solution of the LP relaxation to the path-based formulation

Require: Desired number of commodities n

Set $P'(k) = \{p \in P(k) : 0 < x_p^{k,LP} < 1\}$

Set $s_k = |P'(k)|$

return $RandomizedCommoditySelection(n,\ K,\ s)$

4.2 Lower Bounds

At each iteration our solution approach produces both an upper and a lower bound. The lower bound is provided by the value of the LP relaxation of the path-based formulation. However, since the pricing problem is a shortest path problem, the bound is no better than the LP relaxation of the arc-based formulation, which is known to be weak. We attempt to strengthen the bound by adding valid inequalities before solving the path-based formulation. We first note that we can translate inequalities between the arc and path formulations using the transformation $x_{ij}^k = \sum_{p \in \bar{P}(k):(i,j) \in p} x_p^k$, where $\bar{P}(k)$ is the set of paths generated for commodity k thus far. Since we want valid inequalities that are violated by the optimal solution to the LP relaxation of the path-based formulation given the sets $\bar{P}(k)$, a desirable property of a valid inequality (π, π_0) is to have $\pi_{ij}^k > 0$ when $x_{ij}^k > 0$ in the optimal LP solution. We estimate the likelihood of x_{ij}^k being greater than 0 in the optimal LP solution by considering the ratio

$$r_{ij}^k = \frac{|\{p \in \bar{P}(k) : (i,j) \in p\}|}{|\bar{P}(k)|}$$

which calculates the percentage of paths we have generated (and hence will consider in the LP) for commodity k that use arc (i, j).

We have focused on two techniques for strengthening the path-formulation LP

- Judiciously adding disaggregate coupling inequalities, and
- Lifting cuts found while solving small IPs during the neighborhood search.

We could further strengthen the path-formulation bound by generating valid inequalities after solving the LP relaxation which the current fractional solution does not satisfy. However, identifying the separating inequality and re-solving the LP relaxation would would take time away from the search for primal solutions. This trade-off could be avoided in a parallel implementation by having separate processes work on the lower and upper bounds.

4.2.1 Disaggregate Inequalities

It is well-known that the disaggregate coupling inequalities (10) strengthen the LP bound, yet due to the great number of them it is typically not possible to add them all. In addition, while adding the disaggregate coupling inequalities strengthens the LP bound they may also increase the solve time for the LP relaxation. Of course adding the inequality (10) for commodity k and arc (i, j) only strengthens the LP bound if in fact

$$x_{ij}^k = \sum_{p \in \bar{P}(k):(i,j) \in p} x_p^k > 0$$

in the optimal LP solution. Hence we determine the arcs (i, j) and commodities k for which the disaggregated coupling constraints will be added using the ratio r_{ij}^k. A threshold value $T \in [0, 1]$ is specified and the inequality is added for commodity k and arc (i, j) only when $r_{ij}^k \geq T$.

4.2.2 Lifted Cover Inequalities

We have investigated re-using cuts (π, π_0) found while solving small IPs during the neighborhood search. These cuts are lifted to make them globally valid and to strengthen them. As the inequalities are found while solving small IPs during the neighborhood search, the computational effort is only in the lifting process. After lifting, these inequalities are added to the path formulation to strengthen the bounds its LP relaxation provides.

We have focused our efforts on the single-path variant of FCNF and investigated lifting cover inequalities based on arc capacities. A *cover* with respect to an arc (i, j) is a set $C \subseteq K$ such that $\sum_{k \in C} d^k > u_{ij}$. Since the variables x_{ij}^k are binary for the single-path variant we have the *cover inequality* $\sum_{k \in C} x_{i,j}^k \leq |C| - 1$.

We use the procedure suggested by Gu et al. (1998) for separating cover inequalities. Note that since the cover inequality is generated during the solution of a commodity subset IP some variables are fixed to either 0 or 1. As suggested by Gu et al. (1998), we first uplift variables fixed to 0 and then down-lift those fixed to 1.

It is well known that the strength of the lifted cover inequality depends on the order in which the variables fixed to 0 (or 1) are lifted. Therefore we first lift variables x_{ij}^k whose value is likely to be positive in the optimal LP solution. Hence, the lifting order is quite simple: lift in decreasing order of r_{ij}^k. The complete procedure is outlined in Algorithm 15.

Currently, we limit ourselves to cover inequalities that are violated in the small IP. Given that we are especially interested in strengthening the LP relaxation of the path-based formulation, this methodology can be extended to search for cover inequalities that are *not* violated in the small IP, but are likely to be violated when all commodities are considered.

Since our lifting order depends on the sets $\bar{P}(k)$ which may change at each iteration, we could re-lift the cover inequalities as the path sets change and get different valid inequalities.

Algorithm 15 Cover Inequality Lifting

Require: Cover Inequality (π, π_0) for arc (i, j) from commodity subset IP
Require: Set of commodities K^0 such that when generating (π, π_0), x_{ij}^k fixed to 0
Require: Set of commodities K^1 such that when generating (π, π_0), x_{ij}^k fixed to 1
Calculate $r_{ij}^k = \frac{|\{p \in \bar{P}(k):(i,j) \subseteq p\}|}{|\bar{P}(k)|}, \ \ \forall k \in K^0 \cup K^1$
Uplift variables $x_{ij}^k, k \in K^0$ in descending order of r_{ij}^k
Downlift variables $x_{ij}^k, k \in K^1$ in descending order of r_{ij}^k
return new cut $(\pi^{'}, \pi_0^{'})$

However, we do not re-lift because of our time-constrained setting and the computational effort lifting requires.

4.3 Initial Feasible Solution

Our scheme for generating an initial feasible solution has two phases. In Phase I we find a path for each commodity by solving a shortest path problem where arc costs reflect a linearized fixed charge and are updated using a slope-scaling approach. Heuristics based on updating the linearization of the fixed charge are given in Balinski (1961), Crainic et al. (2004), and Kim and Pardalos (1999). The solution to the LP relaxation of Arc-FCNF has $y_{ij} = \frac{1}{u_{ij}} \sum_k d^k x_{ij}^k$, hence we have $f_{ij} y_{ij} = \frac{f_{ij}}{u_{ij}} \sum_k d^k x_{ij}^k = \sum_k f_{ij} \frac{d^k}{u_{ij}} x_{ij}^k$. This suggests that we can use a variable cost $\tilde{c}_{ij} = c_{ij} d^k + f_{ij} \frac{d^k}{u_{ij}}$ for each arc (i, j) and then solve a shortest path problem using these arc costs for each commodity k. Instead, we recognize that by assigning paths to commodities sequentially we remove the capacity of some arcs. Hence we update the variable cost on an arc (i, j) to reflect how much of its capacity remains. The details are given in Algorithm 16.

If we have a feasible solution after Phase I, we stop. Since we do not explicitly enforce the capacity constraints on arcs during Phase I, we may not end with a feasible solution. In this case, we go on to Phase II, where for each arc whose capacity is exceeded, we find a set of commodities to re-route away from the arc in an attempt to "fix the infeasibility." The details are given in Algorithm 17. Note that this method cannot guarantee that a feasible solution is found when one exists, nor can it detect whether an instance is infeasible.

Algorithm 16 Phase I algorithm

Sort commodities in descending order of demand d^k
Set $\tilde{u}_{ij} = u_{ij}$
for $k = 1$ to K **do**
  Set $\tilde{c}_{ij} = c_{ij} + f_{ij} * \frac{d^k}{\tilde{u}_{ij}}$
  Solve a shortest path problem for commodity k w.r.t. arc costs $\tilde{c}_{ij}$; let x^k_{ij} indicate when arc (i, j) used by k
  Set $\tilde{u}_{ij} = max(1, \tilde{u}_{ij} - x^k_{ij} * d^k)$
end for

Algorithm 17 Phase II algorithm

while $\exists (i, j) : \sum_k x^k_{ij} * d^k > u_{ij}$ **do**
  Randomly choose such an (i, j)
  Sort commodities using the arc $(x^k_{ij} = 1)$ in ascending order of demand d^k
  Find the smallest l such that $\sum_{k=l}^{K} x^k_{ij} * d^k \leq u_{ij}$
  Solve our commodity subset IP for those first $l - 1$ commodities
end while

5 Computational Results

There are many questions to answer regarding the primal and dual-side performance of our approach, which we refer to as IP Search. Specifically:

- Primal-Side
  - How competitive is IP Search with existing metaheuristics and commercial IP solvers?
  - Is there value in using both commodity subset and arc subset IPs?
  - How successful is the method for finding an initial feasible solution?
  - Do the solvers contribute equally to finding good solutions?
  - Does the biased selection of solvers lead to better solutions?
- Dual-Side
  - Does IP Search produce a stronger dual bound than the optimal solution to the LP relaxation of the arc-based formulation?
  - Is the dual bound produced by IP Search competitive with the root node bound produced by a commercial IP solver using an arc-based formulation?

– Do the disaggregate inequalities strengthen the dual bound?

– Do the lifted of cover inequalities strengthen the dual bound?

IP Search is implemented in C++ and uses CPLEX 9.1 to solve the subset IPs. Other than those taken from the literature, all the results are obtained from experiments performed on a machine with 8 Intel Xeon CPUs running at 2.66 GHz and 8 GB RAM. Whenever CPLEX 9.1 is used as a benchmark, it was run with an emphasis on integer feasibility, with the disaggregate constraints $x_{ij}^k \leq y_{ij}$ added as user cuts, and with an optimality tolerance set to 5%. All computation times reported are in seconds.

5.1 Instance Generation

A network generator was used to create instances for experimentation. The generator creates instances without parallel arcs. The inputs to the generator are:

- The number of nodes, arcs, and commodities in the network
- A range $[c_l, c_u]$ for arc variable costs
- A range $[b_l, b_u]$ for commodity quantities
- A range $[1, k_u]$ of how many commodities can fit on an arc
- A ratio R of arc fixed cost to variable cost

The generator creates an arc (i, j) by randomly picking nodes i and j. The variable cost c_{ij} for arc (i, j) is uniformly drawn from $[c_l, c_u]$. The fixed cost f_{ij} for arc (i, j) is uniformly drawn from $[Rc_l, Rc_u]$. The capacity u_{ij} for an arc is set to kb, where k is uniformly drawn from $[1, k_u]$ and b is uniformly drawn from $[b_l, b_u]$. Commodities k are created by randomly picking pairs of nodes $(o(k), d(k))$ such that a path exists between the two nodes. The quantity b_k for commodity k is uniformly drawn from $[b_l, b_u]$. Finally all commodity and arc data is rounded down to integer values.

We use the same instance classifications as Ghamlouch et al. (2003). An instance is classified as F if the ratio of fixed to variable cost for an arc is high and V otherwise. An instance is classified as T if the approximate number of commodities that can be accommodated on an arc is low and L otherwise. Hence we have 4 classes of instances $(F, T), (F, L), (V, T), (V, L)$.

5.2 Calibration

Since we solve IPs as subroutines in IP Search, their size is important. If the IP is too large we may not find a good improving solution in the time alloted to the IP solver. On the other hand, if the IP is too small it simply may not yield a large enough neighborhood to contain a good improving solution. We determine the arc and commodity subset sizes for a specific

instance size by measuring the performance of the heuristic as the subset size grows. For example, to determine the number of commodities in our subset when running IP Search on instances with 500 nodes, 2500 arcs and 150 commodities we randomly generate 5 instances of that size. We then run the heuristic on each of the 5 instances multiple times, varying the number of commodities selected. Table 1 shows the results for instances of class (F,T) with 500 nodes and 2500 arcs. We report the number of commodities in the instance (K), the number of commodities in a subset (KS), the average percentage improvement (Avg. Imp.), and the average solution time in seconds (Avg. Time).

Table 1: Calibrating instances with 500 nodes and 2500 arcs

K	KS	Avg. Imp.	Avg. Time
50	5	.003	1.75
50	10	.008	6.4
50	15	.013	22.2
50	20	.001	49.8
100	10	.009	4.6
100	15	.013	7.4
100	20	.019	13.2
100	25	.014	22.6
100	30	.026	38.8
150	10	.006	4.8
150	15	.023	7.52
150	20	.006	8.6
150	25	.015	11
150	30	.017	10.8
150	35	.014	18
200	10	.009	6.6
200	15	.007	5.2
200	20	.011	7.8
200	25	.012	11.2
200	30	.018	13
200	35	.010	10.2
200	40	.02	18.4
200	45	.012	17.6

We observe that choosing 20% of the total number of commodities in the instance is a reasonable rule-of-thumb. For instances with 2000 and 3000 arcs the same phenomenon was observed. It is not surprising that the size of the commodity subset grows with the number of commodities in the instance. The more commodities outside of the subset, the more likely an arc (i, j) is used by a commodity outside of the subset, which implies there will be no fixed charge for arc (i, j). Hence the larger the number of commodities in the instance the more the commodity subset IPs become multi-commodity flow problems without fixed charges.

5.3 Upper Bound

5.3.1 Comparison to Tabu Cycle and Path Relink

To examine how IP Search compares with existing metaheuristics, we compared it to a cycle-based tabu-search algorithm (Ghamlouch et al. (2003)) and a cycle-based path-relinking approach (Ghamlouch et al. (2004)) on the instances used in Ghamlouch et al. (2004); except for the 6 instances identified as "easy" (solved in a few seconds by a commercial IP solver). Note that the comparison is only for the variant in which commodities can be routed along multiple paths (the variant handled by the metaheuristics). The results are shown in Table 2, where we report the instance, the value of the solution found by CPLEX in 12 hours (CPLEX), the value of the solution obtained by the tabu search algorithm (Tabu Cycle), the value of the solution obtained by the path relinking algorithm (Path Relink), the value of the solution obtained by IP Search (IP Search), the percentage difference between the value of the solution found by IP Search and the value of the solution found by CPLEX in 12 hours (CPLEX gap), the percentage difference between the value of the solution found by IP Search and the best of the values of the solutions found by the tabu search and path relinking algorithms (Best gap), the optimality gap (Opt gap), and for each of the different algorithms the time to reach the best solution (TTB). When we report a percentage difference (or gap) between IP Search and "X", it is computed as $100\frac{\text{IP Search}-\text{X}}{\text{IP Search}}$. To compute the the optimality gap, we use the dual bound produced by CPLEX in 12 hours. Note that we imposed a time limit of 15 minutes (900 seconds) on IP Search.

We first observe that IP Search finds a better solution than the metaheuristics in all but 2 of the 37 instances. Since different hardware platforms were used to produce the metaheuristic and IP Search results a comparison of computing times cannot be exact. However, the time to the best solution for instance (30,700,400,F,L) indicates that the computing time is orders of magnitudes longer for the metaheuristics than for IP Search. For the instance (100,400,10,F,T) the percentage difference between the value of the best of the metaheuristics solutions and the value of the solution produced by IP Search is less than 1. Hence, we can comfortably conclude that IP Search is superior to these metaheuristics. We next observe that IP Search is producing solutions that are very competitive with those produced by CPLEX in 12 hours. Even though the instances are relatively small, we see that CPLEX often needs more than 5 hours to find its best solution. Finally, we observe that the optimality gap for the solution produced by IP Search is small, within 5% for 24 of the 37 instances and only 3.96% on average.

Table 2: Comparison with Tabu Search and Path Relinking

	Solution Value							TTB			
Problem	CPLEX	Tabu Cycle	Path Relink	IP Search	CPLEX Gap	Best Gap	Opt Gap	CPLEX	Tabu Cycle	Path Relink	IP Search
100,400,10,V,L	28,423	28,786	28,485	28,423	0.00	-0.22	0.00	3	252	89	35
100,400,10,F,L	23,949	24,022	24,022	23,949	0.00	-0.30	0.00	3,354	196	82	9
100,400,10,F,T	**63,764**	**67,184**	**65,278**	**65,885**	**3.22**	**0.92**	**13.97**	**34,301**	**451**	**209**	**813**
100,400,30,V,T	384,802	385,508	384,926	384,836	0.01	-0.02	0.60	1,290	1,199	492	330
100,400,30,F,L	49,018	51,831	51,325	49,694	1.36	-3.28	3.44	22,457	717	314	886
100,400,30,F,T	138,948	147,193	141,359	141,365	1.71	0.00	11.65	36,001	1,300	480	888
20,230,40,V,L	423,848	430,628	424,385	424,385	0.13	0	0.13	1	214	148	4
20,230,40,V,T	371,475	372,522	371,811	371,779	0.08	-0.01	0.08	1	241	156	41
20,230,40,F,T	643,036	652,775	645,548	643,187	0.02	-0.37	0.02	13	259	172	45
20,230,200,V,L	94,213	100,001	100,404	95,097	0.93	-5.16	6.21	6,118	2,585	2,494	822
20,230,200,F,L	137,642	148,066	147,988	141,253	2.56	-4.77	8.72	8,079	3,142	2,878	691
20,230,200,V,T	97,914	106,868	104,689	99,410	1.5	-5.31	4.9	7,097	2,730	2,210	821
20,230,200,F,T	132,081	147,212	147,554	140,273	5.84	-4.95	8.06	17,397	3,634	3,386	156
20,300,40,V,L	429,398	432,007	429,398	429,398	0	0	0	1	305	224	19
20,300,40,F,L	586,077	602,180	590,427	586,077	0	-0.74	0	5	336	228	29
20,300,40,V,T	464,509	466,115	464,509	464,509	0	0	0	2	379	247	24
20,300,40,F,T	604,198	615,426	609,990	604,198	0	-0.96	0	1	350	214	68
20,300,200,V,L	73,335	81,367	78,184	75,319	2.63	-3.8	3.75	36,364	4,086	3,566	802
20,300,200,F,L	111,436	122,262	123,484	117,543	5.2	-4.01	7.18	42,220	4,210	4,012	686
20,300,200,V,T	74,991	80,344	78,866	76,198	1.58	-3.5	3.44	6,490	4,204	3,924	388
20,300,200,F,T	103,967	113,947	113,584	110,344	5.78	-2.94	7.08	42,585	4,855	3,857	396
30,520,100,V,L	53,958	56,603	54,904	54,113	0.29	-1.46	0.82	436	2,261	1,194	218
30,520,100,F,L	90,488	103,657	102,054	94,388	4.13	-8.12	7.78	39,409	2,684	1,459	226
30,520,100,V,T	51,475	54,454	53,017	52,174	1.34	-1.62	1.34	38,260	2,716	1,513	455
30,520,100,F,T	94,199	105,130	106,130	98,883	4.74	-6.32	5.97	39,302	2,892	1,522	815
30,520,400,V,L	111,844	122,673	119,416	114,042	1.93	-4.71	2.85	36,436	55,771	27,477	394
30,520,400,F,L	145,430	164,140	163,112	154,218	5.7	-5.77	5.79	30,624	40,070	36,669	750
30,520,400,V,T	114,133	122,655	120,170	114,922	0.69	-4.57	0.97	31,588	4,678	23,089	621
30,520,400,F,T	149,009	169,508	163,675	154,606	3.62	-5.87	3.72	31,127	49,886	52,173	466
30,700,100,V,L	47,603	50,041	48,723	47,612	0.02	-2.33	0.02	205	2,959	1,860	32
30,700,100,F,L	58,767	64,581	63,091	60,700	3.18	-3.94	6.22	18,900	3,182	1,837	741
30,700,100,V,T	45,199	48,176	47,209	46,046	1.84	-2.53	2.21	34,111	3,746	1,894	371
30,700,100,F,T	53,815	57,628	56,575	55,609	3.23	-1.74	4.78	39,396	3,547	1,706	387
30,700,400,V,L	96,063	107,727	105,116	98,718	2.69	-6.48	2.93	40,901	38,857	22,314	222
30,700,400,F,L	**129,094**	**150,256**	**145,026**	**152,576**	**15.39**	**4.95**	**15.42**	**10,231**	**68,214**	**75,664**	**860**
30,700,400,V,T	93,626	101,749	101,212	96,168	2.64	-5.24	2.81	37,661	51,764	24,288	365
30,700,400,F,T	126,779	144,852	141,013	131,629	3.68	-7.13	3.79	5,464	79,053	44,936	225
Average					2.37	-2.76	3.96	18860.30	12106.08	9431.81	408.14

Table 3: Primal-side Comparison with CPLEX

Problem	CPLEX-15M	CPLEX-12H	IP Search	CPLEX-15M Gap	CPLEX-12H Gap	Opt Gap
500,2000,50,F,L	5,301,081	5,301,081	3,910,120	-35.57	-35.57	55.94
500,2000,50,F,T	X	7,927,065	5,249,040	N/A	-51.02	31.14
500,2000,100,F,L	8,944,724	8,299,799	6,764,310	-32.23	-22.70	63.52
500,2000,100,F,T	10,199,000	8,306,181	7,718,750	-32.13	-7.61	31.84
500,2000,150,F,L	10,996,000	10,080,000	8,618,060	-27.59	-16.96	63.78
500,2000,150,F,T	12,115,000	10,770,000	9,448,890	-28.22	-13.98	65.32
500,2000,200,F,L	13,808,000	12,824,000	10,333,200	-33.63	-24.10	64.37
500,2000,200,F,T	X	X	12,425,600	N/A	N/A	49.63
500,2500,50,F,L	4,611,275	4,611,275	3,841,350	-20.04	-20.04	70.88
500,2500,50,F,T	5,779,926	5,084,529	4,666,740	-23.85	-8.95	32.56
500,2500,100,F,L	9,351,042	9,251,042	6,875,420	-36.01	-34.55	71.97
500,2500,100,F,T	9,724,997	7,995,284	7,235,520	-34.41	-10.50	46.31
500,2500,150,F,L	13,660,000	12,497,000	9,730,100	-40.39	-28.44	77.45
500,2500,150,F,T	11,385,000	10,683,000	7,934,360	-43.49	-34.64	72.22
500,2500,200,F,L	15,539,000	13,468,000	11,261,300	-37.99	-19.60	75.02
500,2500,200,F,T	18,906,000	14,948,000	12,825,300	-47.41	-16.55	58.30
500,3000,50,F,L	5,098,318	5,098,318	3,596,980	-41.74	-41.74	76.27
500,3000,50,F,T	5,615,096	4,866,768	4,504,260	-24.66	-8.05	41.85
500,3000,100,F,L	8,721,798	8,721,798	6,577,980	-32.59	-32.59	75.06
500,3000,100,F,T	10,119,000	8,330,109	7,517,970	-34.60	-10.80	49.67
500,3000,150,F,L	12,628,000	12,623,000	9,214,960	-37.04	-36.98	80.31
500,3000,150,F,T	12,615,000	10,147,000	9,186,840	-37.32	-10.45	55.71
500,3000,200,F,L	15,039,000	13,441,000	10,853,400	-38.56	-23.84	80.03
500,3000,200,F,T	17,883,000	13,674,000	11,578,000	-54.46	-18.10	58.35
Average				-35.18	-22.95	60.31

5.3.2 Comparison to Commercial IP Solver

Next we focus on the performance of IP Search on large instances for the variant in which each commodity has to be routed along a single path by comparing the value of the solution produced to the value of the solution produced by CPLEX in 15 minutes (CPLEX-15M) and in 12 hours (CPLEX-12H). We consider 24 instances of the classes (F,T) and (F,L), which we refer to as the GT instances. The results are given in Table 3. An "X" in a column indicates that no feasible solution was found. As before, the value reported for IP Search is the value of the solution produced within 15 minutes.

We observe that in *every* instance IP Search finds a better solution in 15 minutes than CPLEX does in 12 hours. We also see that the improvement over the solution found by CPLEX in 15 minutes is significant, often greater than 30%. Even the improvement over the solution found by CPLEX in 12 hours is impressive, often greater than 20%. Unfortunately, little can be said with confidence regarding the true optimality gap of the solutions produced by IP Search since the dual bounds produced by CPLEX change very little over the course of the execution and are likely to be weak. In fact, for many of the loosely capacitated instances, CPLEX did not find a significantly better primal solution in 12 hours than it did in 15 minutes. This highlights the difficulty that an LP-based branch-and-bound algorithm can have in finding good primal solutions when the dual bounds are

weak.

In Table 4 we see that IP Search not only outperforms CPLEX, but it does so in very little time, needing on average less than 3 minutes to produce a solution that is better than the best solution produced by CPLEX in 12 hours. The time limit of 15 minutes on IP

Table 4: Heuristic Time To Beat CPLEX

Problem	CPLEX 15 M Primal	CPLEX 12 H Primal
500,2000,50,F,L	10	10
500,2000,50,F,T	16	16
500,2000,100,F,L	98	98
500,2000,100,F,T	49	343
500,2000,150,F,L	46	119
500,2000,150,F,T	45	294
500,2000,200,F,L	44	280
500,2000,200,F,T	19	19
500,2500,50,F,L	82	82
500,2500,50,F,T	10	23
500,2500,100,F,L	32	32
500,2500,100,F,T	77	226
500,2500,150,F,L	24	47
500,2500,150,F,T	50	115
500,2500,200,F,L	57	289
500,2500,200,F,T	37	142
500,3000,50,F,L	17	17
500,3000,50,F,T	5	296
500,3000,100,F,L	27	27
500,3000,100,F,T	120	365
500,3000,150,F,L	22	22
500,3000,150,F,T	30	338
500,3000,200,F,L	50	75
500,3000,200,F,T	10	104
Average	41	141

Search is, of course, self-imposed. Table 5 reports the results for IP Search after 15, 30, and 60 minutes as well as the percentage improvement over the initial solution found. We see that the greatest percentage improvement occurs in the first 15 minutes, but that IP Search continues to find improving solutions when given more time. The column labeled "Times Phase II" gives the number of times Phase II of the scheme for finding an initial feasible solution is executed. Not surprisingly, Phase II is only executed for tightly-capacitated instances. In fact, for the majority of instances Phase II is not needed at all.

Table 5: IP Search Given More Time

Problem	Times Phase II	Init. Feas. Soln	15 Minute Soln	30 Minute Soln	60 Minute Soln	Imp. After 15 M	Imp. After 30 M	Imp. After 60M
500,2000,50,F,L	0	6,175,840	3,910,120	3,907,440	3,823,610	-36.69	-36.73	-38.09
500,2000,50,F,T	5	6,314,980	5,249,040	5,112,490	4,949,780	-16.88	-19.04	-21.62
500,2000,100,F,L	0	10,244,000	6,764,310	6,655,370	6,453,880	-33.97	-35.03	-37.00
500,2000,100,F,T	4	10,125,700	7,718,750	7,632,220	7,619,670	-23.77	-24.63	-24.75
500,2000,150,F,L	0	13,317,800	8,618,060	8,279,270	8,081,600	-35.29	-37.83	-39.32
500,2000,150,F,T	0	14,557,800	9,448,890	9,250,760	8,807,650	-35.09	-36.45	-39.50
500,2000,200,F,L	0	17,411,800	10,333,200	10,138,900	9,828,350	-40.65	-41.77	-43.55
500,2000,200,F,T	0	17,543,000	12,425,600	12,117,800	11,893,100	-29.17	-30.93	-32.21
500,2500,50,F,L	0	5,712,380	3,841,350	3,780,160	3,612,030	-32.75	-33.83	-36.77
500,2500,50,F,T	1	5,898,160	4,666,740	4,661,160	4,600,200	-20.88	-20.97	-22.01
500,2500,100,F,L	0	10,640,700	6,875,420	6,621,140	6,400,140	-35.39	-37.78	-39.85
500,2500,100,F,T	1	10,077,400	7,235,520	7,065,020	6,953,660	-28.20	-29.89	-31.00
500,2500,150,F,L	0	14,834,200	9,730,100	9,475,760	9,089,920	-34.41	-36.12	-38.72
500,2500,150,F,T	0	13,365,000	7,934,360	7,673,410	7,571,640	-40.63	-42.59	-43.35
500,2500,200,F,L	0	17,446,900	11,261,300	10,623,700	10,099,200	-35.45	-39.11	-42.11
500,2500,200,F,T	1	18,628,100	12,825,300	11,843,200	11,452,900	-31.15	-36.42	-38.52
500,3000,50,F,L	0	5,687,990	3,596,980	3,519,400	3,457,280	-36.76	-38.13	-39.22
500,3000,50,F,T	0	5,553,000	4,504,260	4,405,120	4,262,350	-18.89	-20.67	-23.24
500,3000,100,F,L	0	10,383,600	6,577,980	6,202,380	6,015,950	-36.65	-40.27	-42.06
500,3000,100,F,T	2	10,350,000	7,517,970	7,444,950	7,186,810	-27.36	-28.07	-30.56
500,3000,150,F,L	0	13,707,300	9,214,960	9,126,970	8,919,720	-32.77	-33.42	-34.93
500,3000,150,F,T	0	12,958,300	9,186,840	8,898,880	8,709,390	-29.10	-31.33	-32.79
500,3000,200,F,L	0	16,869,900	10,853,400	10,419,400	10,040,000	-35.66	-38.24	-40.49
500,3000,200,F,T	1	17,071,600	11,578,000	10,750,200	10,390,700	-32.18	-37.03	-39.13
Average						-31.66	-33.59	-35.45

We observed in Table 3 that IP Search significantly outperforms CPLEX on large instances when each commodity has to be routed along a single path. Next we investigate what happens when we have small instances but require a commodity to be routed along a single path. The results are given in Table 6. As before, IP Search is limited to 15 minutes. When CPLEX obtains a better solution than IP Search, we report how long it took CPLEX to find the first solution that is better than the solution found by IP Search (CPLEX Time Beat). We observe that in only 2 of the 37 instances does CPLEX find a better solution in 15 minutes than IP Search. In those 2 instances the difference in solution value between IP Search and CPLEX is less than .15%. On the other hand, IP Search often finds solutions that are 5% better than what CPLEX finds in 15 minutes. The optimality gap information reveals that the solutions produced by IP Search are of high quality, on average within 4% of optimality. When CPLEX does find a better solution, it often requires more than 3 hours to do so. We also see that IP Search produces solutions within 15 minutes that are competitive with what CPLEX produces in 12 hours. CPLEX's best improvement over IP Search occurs for instance (100,400,30,F,L) where it finds a solution that is only 4.52% better than IP Search.

Table 6: Primal-side Comparison with CPLEX - Metaheuristic Instances - Single Path

Problem	CPLEX-15M	CPLEX-12H	IP Search	CPLEX-15M Gap	CPLEX-12H Gap	Opt Gap	CPLEX Time Beat
100,400,10,V,L	31,785	31,785	31,785	0	0	0	-
100,400,10,F,L	24,104	24,104	24,104	0	0	0	-
100,400,10,F,T	328,177	328,177	328,177	0	0	0	-
100,400,30,V,T	750,420	750,420	750,420	0	0	0	-
100,400,30,F,L	55,665	50,391	52,779	-5.47	4.52	6.90	10,998
100,400,30,F,T	188,113	188,113	188,113	0	0	0	-
20,230,40,V,L	423,933	423,933	424,671	0.17	0.17	0.17	1
20,230,40,V,T	398,870	398,870	398,870	0	0	0	-
20,230,40,F,T	668,699	668,699	668,699	0	0	0	-
20,230,200,V,L	102,118	95,126	96,456	-5.87	1.38	5.18	2,595
20,230,200,F,L	160,535	141,197	141,700	-13.29	0.35	6.37	-
20,230,200,V,T	107,145	100910	100,884	-6.21	-0.03	6.4	24,469
20,230,200,F,T	157,874	143781	141,734	-11.39	-1.44	8.98	-
20,300,40,V,L	430,253	430,253	430,253	0	0	0	-
20,300,40,F,L	597,059	597,059	597,059	0	0	0	-
20,300,40,V,T	501,766	501,766	501,766	0	0	0	-
20,300,40,F,T	643,395	643,395	643,395	0	0	0	-
20,300,200,V,L	81,101	78262	76,946	-5.4	-1.71	5.7	-
20,300,200,F,L	128,871	120301	119,590	-7.76	-0.59	8.14	-
20,300,200,V,T	78,900	77303	77,858	-1.34	0.71	4.99	17,403
20,300,200,F,T	120,997	110897	110,609	-9.39	-0.26	6.1	20,911
30,520,100,V,L	54,394	54,387	54,454	0.11	0.12	0.12	308
30,520,100,F,L	103,771	98797	97,199	-6.76	-1.64	6.34	-
30,520,100,V,T	53,812	53,812	53,812	0	0	0	-
30,520,100,F,T	102,186	99204	100,871	-1.3	1.65	4.35	8,677
30,520,400,V,L	132,799	120781	117,078	-13.43	-3.16	5.25	-
30,520,400,F,L	164,469	159146	157,682	-4.3	-0.93	7.92	-
30,520,400,V,T	132,935	120326	118,214	-12.45	-1.79	3.63	-
30,520,400,F,T	166,561	158396	161,577	-3.08	1.97	7.87	22,897
30,700,100,V,L	47,883	47,883	47,883	0	0	0	-
30,700,100,F,L	63,275	60856	61,254	-3.3	0.65	3.43	12,109
30,700,100,V,T	47,864	47,670	47,736	-0.27	0.14	0.14	1,236
30,700,100,F,T	57,218	56686	57,125	-0.16	0.77	0.77	2,019
30,700,400,V,L	103,839	100332	102,216	-1.59	1.84	6.28	12,599
30,700,400,F,L	677,227	151043	144,077	-370.05	-4.83	10.48	-
30,700,400,V,T	100,937	99972	99,118	-1.84	-0.86	5.52	-
30,700,400,F,T	142,604	137149	138,787	-2.75	1.18	8.68	11,605
Average				-13.17	-0.05	3.51	10,559.07

5.3.3 Value of Subset Solvers

Table 7 shows the performance of standard IP Search, IP Search using only arc subset IPs, and IP Search using only commodity subset IPs. The percentage difference, or gap, information for only using subset type X is calculated as $100\frac{\text{X–Arc \& Commodity}}{\text{Arc \& Commodity}}$. We observe that using each subset IP type by itself is already effective, sometimes even more effective than using them in combination, but that on average the combination yields better solutions.

Table 7: Value of subset IP types

Problem	Arc + Commodity	Arc	Arc Gap	Commodity	Commodity Gap
500,2000,50,F,L	3,910,120	4,256,180	8.85	4,066,470	4.00
500,2000,50,F,T	5,249,040	5,682,850	8.26	5,330,480	1.55
500,2000,100,F,L	6,764,310	7,425,740	9.78	6,735,860	-0.42
500,2000,100,F,T	7,718,750	8,549,650	10.76	8,089,280	4.80
500,2000,150,F,L	8,618,060	8,988,500	4.30	9,207,030	6.83
500,2000,150,F,T	9,448,890	9,257,380	-2.03	9,587,780	1.47
500,2000,200,F,L	10,333,200	10,880,100	5.29	11,229,800	8.68
500,2000,200,F,T	12,425,600	13,030,400	4.87	12,581,300	1.25
500,2500,50,F,L	3,841,350	4,286,400	11.59	3,960,530	3.10
500,2500,50,F,T	4,666,740	4,939,120	5.84	4,635,180	-0.68
500,2500,100,F,L	6,875,420	7,136,500	3.80	6,909,580	0.50
500,2500,100,F,T	7,235,520	7,796,070	7.75	7,859,290	8.62
500,2500,150,F,L	9,730,100	10,013,900	2.92	9,570,420	-1.64
500,2500,150,F,T	7,934,360	8,604,130	8.44	8,497,240	7.09
500,2500,200,F,L	11,261,300	11,188,600	-0.65	11,714,800	4.03
500,2500,200,F,T	12,825,300	12,816,900	-0.07	12,947,200	0.95
500,3000,50,F,L	3,596,980	4,390,700	22.07	3,759,470	4.52
500,3000,50,F,T	4,504,260	4,855,100	7.79	4,380,330	-2.75
500,3000,100,F,L	6,577,980	7,366,460	11.99	6,468,730	-1.66
500,3000,100,F,T	7,517,970	8,211,270	9.22	8,303,140	10.44
500,3000,150,F,L	9,214,960	10,222,900	10.94	9,479,810	2.87
500,3000,150,F,T	9,186,840	9,204,010	0.19	9,206,770	0.22
500,3000,200,F,L	10,853,400	11,495,100	5.91	11,013,700	1.48
500,3000,200,F,T	11,578,000	11,838,700	2.25	11,606,100	0.24
Average			6.67		2.73

To gain a greater understanding of the efficacy of each selection scheme, we examine in detail the contribution of each on finding improving solutions. Table 8 gives the breakdown by scheme, averaged over the GT instances, after running IP Search for 60 minutes. We report the percentage of iterations a scheme was executed (% Called) and the percentage improvement that can be attributed to a scheme (% Improvement). The schemes are denoted by A-x for arc subset selection scheme x and C-x for commodity subset selection scheme x. We observe that the vast majority of solution improvement is due to the commodity subset IPs. This may simply be due to the biased random selection mechanism repeatedly selecting the commodity subset solvers because of their effectiveness. Commod-

Table 8: Selection Method Contribution

Scheme	% Called	% Total Improvement
A-1	6	3
A-2	4	2
A-3	9	8
C-1	14	13
C-2	10	4
C-3	13	9
C-4	14	16
C-5	14	28
C-6	16	17

ity selection scheme 5 results in the largest percentage improvement. Recall that this is the scheme that selects commodities for which few paths have been discovered and was designed to aid in diversification for the arc subset solvers. These results indicate the scheme is effective in finding improving solutions itself.

We also see that the biased randomized selection appears to be working properly as the most effective selection methods are called most frequently. To further evaluate the value of the biased random selection, we compare it to a simple round-robin approach for selecting solvers. Our simple round-robin scheme executes the solvers in the following order: A-1, C-1, C-2, A-2, C-3, C-4, A-3, C-5, and C-6. The results are given in Table 9 where we report the value of the solution obtained when using the biased random selection (Biased Random), the value of the solution obtained when using simple round-robin (Round Robin), and the percentage difference between the two, calculated as $100\frac{\text{Round Robin}-\text{Biased Random}}{\text{Round Robin}}$. We see that biased random selection does better than round-robin, but not significantly better. Of course, it may be that we have simply chosen an effective order for the round-robin scheme. The results indicate that IP Search is robust in the sense that the choice of subset type and the choice of subset selection scheme has a relatively minor impact on the overall effectiveness.

Table 9: Biased Random vs. Round Robin

Problem	Biased Random	Round Robin	Gap
500,2000,50,F,L	3,910,120	4,029,970	3.07
500,2000,50,F,T	5,249,040	5,260,950	0.23
500,2000,100,F,L	6,764,310	6,732,430	-0.47
500,2000,100,F,T	7,718,750	7,944,700	2.93
500,2000,150,F,L	8,618,060	9,134,460	5.99
500,2000,150,F,T	9,448,890	8,996,460	-4.79
500,2000,200,F,L	10,333,200	10,615,300	2.73
500,2000,200,F,T	12,425,600	12,447,300	0.17
500,2500,50,F,L	3,841,350	3,862,160	0.54
500,2500,50,F,T	4,666,740	4,701,030	0.73
500,2500,100,F,L	6,875,420	7,014,860	2.03
500,2500,100,F,T	7,235,520	7,254,500	0.26
500,2500,150,F,L	9,730,100	9,817,410	0.90
500,2500,150,F,T	7,934,360	8,770,120	10.53
500,2500,200,F,L	11,261,300	10,898,400	-3.22
500,2500,200,F,T	12,825,300	13,167,700	2.67
500,3000,50,F,L	3,596,980	3,681,570	2.35
500,3000,50,F,T	4,504,260	4,545,940	0.93
500,3000,100,F,L	6,577,980	6,520,750	-0.87
500,3000,100,F,T	7,517,970	8,389,710	11.60
500,3000,150,F,L	9,214,960	9,675,770	5.00
500,3000,150,F,T	9,186,840	9,522,640	3.66
500,3000,200,F,L	10,853,400	11,729,700	8.07
500,3000,200,F,T	11,578,000	12,053,100	4.10
Average			2.46

5.4 Lower Bound

We have conducted some computational experiments to analyze the quality of the lower bounds produced. Since we want to evaluate the value of the lifted cover inequalities, we only consider the single-path variant of FCNF. First we compare the lower bound IP Search produces with the value of the LP relaxation of the arc-based formulation and with the root node bound CPLEX produces. Since the disaggregate coupling constraints are added as user cuts, they are not present in the LP relaxation, but are reflected in the root node bound in addition to any other cuts generated by CPLEX.

The results are shown in Table 10, where we report the value of the LP-relaxation (LPR), the value of the root node bound (Root Node), the lower bound IP Search produces, and the percentage differences between the bound IP Search produces and the other two bounds (LP Gap and Root Gap). The percentage differences are calculated as $100\frac{\text{IP Search}-\text{LPR}}{\text{IP Search}}$ and $100\frac{\text{Root Node}-\text{IP Search}}{\text{Root Node}}$. Note that this means a positive percentage difference with the value of the LP Relaxation indicates that IP Search has given a stronger lower bound and a positive percentage difference with the value of the root node LP indicates that CPLEX has a stronger root node bound. We observe that IP Search

Table 10: Dual-side Comparison with CPLEX - GT Instances

Problem	LPR	Root Node	IP Search	LP Gap	Root Gap
500,2000,50,F,L	749,167	1,722,684	1,205,430	37.85	30.03
500,2000,50,F,T	2,289,020	3,614,601	2,717,780	15.78	24.81
500,2000,100,F,L	1,069,554	2,466,756	1,803,010	40.68	26.91
500,2000,100,F,T	3,401,818	5,257,456	3,687,050	7.74	29.87
500,2000,150,F,L	1,629,223	3,121,039	2,537,120	35.78	18.71
500,2000,150,F,T	1,761,529	3,276,022	2,792,120	36.91	14.77
500,2000,200,F,L	2,051,456	3,679,590	3,230,900	36.51	12.19
500,2000,200,F,T	4,198,214	6,259,068	5,464,430	23.17	12.70
500,2500,50,F,L	554,544	1,118,701	757,794	26.82	32.26
500,2500,50,F,T	1,970,332	3,144,840	2,162,030	8.87	31.25
500,2500,100,F,L	801,012	1,927,414	1,301,380	38.45	32.48
500,2500,100,F,T	2,325,097	3,884,996	2,747,240	15.37	29.29
500,2500,150,F,L	1,122,028	2,194,102	1,682,550	33.31	23.31
500,2500,150,F,T	1,119,283	2,192,494	1,721,910	35.00	21.46
500,2500,200,F,L	1,663,847	2,811,792	2,752,630	39.55	2.10
500,2500,200,F,T	3,576,177	5,347,065	4,397,850	18.68	17.75
500,3000,50,F,L	447,750	853,592	600,636	25.45	29.63
500,3000,50,F,T	1,629,701	2,618,366	1,755,360	7.16	32.96
500,3000,100,F,L	886,097	1,640,026	1,164,540	23.91	28.99
500,3000,100,F,T	2,222,056	3,783,508	2,499,770	11.11	33.93
500,3000,150,F,L	1,017,504	1,813,662	1,424,980	28.60	21.43
500,3000,150,F,T	2,551,015	4,068,577	2,992,860	14.76	26.44
500,3000,200,F,L	1,236,509	2,165,713	1,791,940	31.00	17.26
500,3000,200,F,T	3,299,636	4,822,044	3,819,270	13.61	20.80
Average				25.25	23.81

produces lower bounds which are much stronger than the value of the LP Relaxation. Furthermore, the difference is larger for loosely capacitated instances. On the other hand we

see that CPLEX produces a much stronger bound at the root node.

Next, we perform the same comparison for the Ghamlouch instances. The results are shown in Table 11. We observe that for these small instances, IP Search produces weaker lower bounds, often less than the value of the LP relaxation. The average values are strongly influenced by two instances, (100,400,10,F,T) and (100,400,30,V,T). When we ignore these instances, IP Search produces better bounds. At the same time, the difference between the lower bound of IP Search and the root node bound CPLEX produces has also narrowed.

Table 11: Dual-side Comparison with CPLEX - Metaheuristic Instances

Problem	LPR	Root Node	IP Search	LP Gap	Root Gap
100,400,10,V,L	31,167	31,785	27,343	-13.99	-16.25
100,400,10,F,L	11,719	22,197	12,342	5.05	-79.85
100,400,10,F,T	289,118	328,177	41,520	-596.33	-690.41
100,400,30,V,T	711,036	750,420	378,722	-87.75	-98.15
100,400,30,F,L	23,498	41,336	30,731	23.54	-34.51
100,400,30,F,T	148,330	188,113	106,334	-39.49	-76.91
20,230,40,V,L	378,623	422,778	363,348	-4.2	14.06
20,230,40,V,T	376,393	398,820	398,870	5.64	-0.01
20,230,40,F,T	609,411	668,699	618,803	1.52	7.46
20,230,200,V,L	68,805	88,873	71,296	3.49	19.78
20,230,200,F,L	98,342	128,039	102,808	4.34	19.71
20,230,200,V,T	73,975	94,211	78,952	6.3	16.2
20,230,200,F,T	100,728	128,768	100,370	-0.36	22.05
20,300,40,V,L	386,461	430,253	376,500	-2.65	12.49
20,300,40,F,L	492,690	583,079	553,120	10.93	5.14
20,300,40,V,T	473,755	501,233	394,472	-20.1	21.3
20,300,40,F,T	572,481	643,395	563,965	-1.51	12.35
20,300,200,V,L	58,934	72,358	59,772	1.4	17.39
20,300,200,F,L	87,877	109,033	86,353	-1.76	20.8
20,300,200,V,T	61,481	73,427	64,765	5.07	11.8
20,300,200,F,T	84,733	102,436	88,939	4.73	13.18
30,520,100,V,L	44,118	53,010	48,606	9.23	8.31
30,520,100,F,L	67,559	87,169	68,312	1.1	21.63
30,520,100,V,T	47,429	52,897	48,957	3.12	7.45
30,520,100,F,T	76,709	93,569	77,083	0.49	17.62
30,520,400,V,L	96,690	110,937	100,603	3.89	9.32
30,520,400,F,L	122,679	145,201	125,903	2.56	13.29
30,520,400,V,T	100,648	113,928	104,874	4.03	7.95
30,520,400,F,T	126,356	148,830	127,283	0.73	14.48
30,700,100,V,L	39,055	47,061	43,468	10.15	7.63
30,700,100,F,L	45,708	57,121	47,836	4.45	16.25
30,700,100,V,T	41,798	46,561	42,788	2.31	8.1
30,700,100,F,T	45,026	54,228	48,330	6.84	10.88
30,700,400,V,L	79,059	95,744	82,330	3.97	14.01
30,700,400,F,L	106,564	128,969	108,054	1.38	16.22
30,700,400,V,T	81,216	93,593	85,416	4.92	8.74
30,700,400,F,T	109,364	126,739	110,959	1.44	12.45
Average				-17.18	-15.89

Since both disaggregate coupling constraints and lifted cover inequalities are used to

strengthen the dual bound IP Search produces, we next analyze their respective contributions. Table 12 shows the value of the LP relaxation (LPR), the value of the LP relaxation with judiciously added disaggregated coupling constraints (LPR + disaggr), and the value of the LP relaxation with judiciously added disaggregated coupling constraints and lifted cover inequalities (LPR + disaggr + cover). We observe that the judiciously selected disaggregate coupling constraint contribute the most to the bound improvement. We also observe that with a threshold value $T = .25$ we typically only add 1% of the possible disaggregate coupling constraints. Not surprisingly, the lifted cover inequalities are most effective for tightly-capacitated instances.

Table 12: Disaggregate vs. Cover Inequalities

Problem	LPR	Only Disagg	Imp. Gap	Disagg & Cover	Imp. . Gap
500,2000,50,F,L	749,167	1,113,400	32.71	1,205,430	37.85
500,2000,50,F,T	2,289,020	2,617,900	12.56	2,717,780	15.78
500,2000,100,F,L	1,069,554	1,699,710	37.07	1,803,010	40.68
500,2000,100,F,T	3,401,818	3,518,480	3.32	3,687,050	7.74
500,2000,150,F,L	1,629,223	2,446,600	33.41	2,537,120	35.78
500,2000,150,F,T	1,761,529	2,524,590	30.23	2,792,120	36.91
500,2000,200,F,L	2,051,456	3,129,100	34.44	3,230,900	36.51
500,2000,200,F,T	4,198,214	5,335,330	21.31	5,464,430	23.17
500,2500,50,F,L	554,544	698,710	20.63	757,794	26.82
500,2500,50,F,T	1,970,332	2,098,610	6.11	2,162,030	8.87
500,2500,100,F,L	1,069,554	1,272,720	15.96	1,301,380	17.81
500,2500,100,F,T	2,325,097	2,700,930	13.91	2,747,240	15.37
500,2500,150,F,L	1,122,028	1,626,530	31.02	1,682,550	33.31
500,2500,150,F,T	1,119,283	1,642,180	31.84	1,721,910	35
500,2500,200,F,L	1,663,847	2,607,580	36.19	2,752,630	39.55
500,2500,200,F,T	3,576,177	4,294,910	16.73	4,397,850	18.68
500,3000,50,F,L	447,750	597,728	25.09	600,636	25.45
500,3000,50,F,T	1,629,701	1,674,410	2.67	1,755,360	7.16
500,3000,100,F,L	886,097	1,163,210	23.82	1,164,540	23.91
500,3000,100,F,T	2,222,056	2,430,220	8.57	2,499,770	11.11
500,3000,150,F,L	1,017,504	1,252,230	18.74	1,424,980	28.6
500,3000,150,F,T	2,551,015	2,894,492	11.87	2,992,860	14.76
500,3000,200,F,L	1,236,509	1,780,330	30.55	1,791,940	31
500,3000,200,F,T	3,299,636	3,720,500	11.31	3,819,270	13.61
Average			21.25		24.39

6 Conclusions

We have proposed a local search heuristic (IP Search) for the fixed charge network flow problem capable of producing high-quality feasible solutions, as well as lower bounds, in a short amount of time. Unlike traditional local search heuristics, we rely on exponential-sized neighborhoods which we search using integer programming technology. Since we use integer programming technology to search neighborhoods, the heuristic can easily handle the different variants of the problem, e.g., with arc capacities, without arc capacities,

commodities have to be routed along a single path, commodities can be routed along multiple paths, etc.

Computational experiments demonstrate that IP Search is superior to existing meta-heuristics and pure integer programming approaches when it comes to producing high-quality primal solutions quickly. Since IP Search produces a lower bound by strengthening the LP relaxation with a limited set of valid inequalities, the bound produced (like those produced by pure integer programming solves) is relatively weak.

IP Search can easily be parallelized, which will likely allow it to produce high-quality solutions even faster. This is at the top of our list for future research. IP Search makes extensive use of both the compact (arc-based) and extended (path-based) formulation of FCNF and we believe this is one reason for its success. Therefore we plan to investigate whether the general framework of IP Search will be successful on other hard discrete optimization problems.

References

Ahuja, R., Ergun, O., Orlin, J., and Punnen, A. (2002). A survery of very large scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102.

Archetti, C., Speranza, M. G., and Savelsbergh, M. W. P. (2008). An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science*, 42:22–31.

Balinski, M. (1961). Fixed cost transportation problem. *Nav. Res. Logistics Q.*, 8:41–54.

Crainic, T., Frangioni, A., and Gendron, B. (2001). Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112:73–99.

Crainic, T. and Gendreau, M. (2002). Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627.

Crainic, T., Gendreau, M., and Farvolden, J. (2000). A simplex-based tabu search method for capacitated network design. *INFORMS J. Comput.*, 12:223–236.

Crainic, T., Gendron, B., and Hernu, G. (2004). A slope scaling/lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. *Journal of Heuristics*, 10:525–545.

Danna, E., Rothberg, E., and Le Pape, C. (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90.

De Franceschi, R., Fischetti, M., and Toth, P. (2006). A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming B*, 105:471–499.

Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical Programming*, 98:23–47.

Ghamlouch, I., Crainic, T., and Gendreau, M. (2003). Cycle-based neighborhoods for fixed charge capacitated multicommodity network design. *Operations Research*, 51:655–667.

Ghamlouch, I., Crainic, T., and Gendreau, M. (2004). Path relinking, cycle-based neighborhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. (1998). Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal of Computing*, 10:427–437.

Holmberg, K. and Yuan, D. (2000). A lagrangean heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48:461–481.

Kim, D. and Pardalos, P. (1999). A solution to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters*, 24:195–203.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403–2435.

Powell, W. and Sheffi, Y. (1989). Design and implementation of an interactive optimization system for network design in the motor carrier industry. *Operations Research*, 37:12–29.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472.

Savelsbergh, M. and Song, J.-H. (2008). An optimization algorithm for inventory routing with continuous moves. *Computers and Operations Research*, 35:2266–2282.

Schmid, V., Doerner, K. F., Hartl, R. F., Savelsbergh, M. W. P., and Stoecher, W. (2008). A hybrid solution approach for ready-mixed concrete delivery. *Transportation Science*, To appear.