Branch-and-Price Guided Search for Integer Programs with an Application to the Multicommodity Fixed Charge Network Flow Problem

Mike Hewitt
Department of Industrial and Systems Engineering, Rochester Institute of Technology, Rochester, NY 14623, USA, mrheie@rit.edu,

George Nemhauser
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, george.nemhauser@isye.gatech.edu,

Martin W.P. Savelsbergh
School of Mathematical and Physical Sciences, University of Newcastle, Callaghan NSW 2308, Australia, martin.savelsbergh@newcastle.edu.au

We develop an exact algorithm for integer programs that uses restrictions of the problem to produce high-quality solutions quickly. Column generation is used both for generating these problem restrictions and for producing bounds on the value of the optimal solution. The performance of the algorithm is greatly enhanced by using structure, such as arises in network flow type applications, to help define the restrictions that are solved. In addition, local search around the current best solution is incorporated to enhance overall performance. The approach is parallelized and computational experiments on a classical problem in network design demonstrate its efficacy.

Key words: integer programmng; column generation; local search; multicommodity fixed-charge network flow
History:

---

1. Introduction

When integer programming (IP) models are used in operational situations there is a need to consider the tradeoff between the conflicting goals of solution quality and solution time, because for many problems solving realistic-size instances to a tight tolerance is still beyond the capability of state-of-the-art solvers. However, by reducing the size of the solution space, such as by fixing variables, good primal solutions frequently can be found quickly. Much like with valid inequalities for integer programs, methods for choosing which variables to fix can be categorized into those that do not rely on problem structure and those that do.

Techniques used within linear programming-based branch-and-bound algorithms such as local branching (Fischetti and Lodi (2003)) and relaxation induced neighborhood search (Danna et al. (2005)) fall into the first category. These techniques use information from the solution to the active linear program and the best known feasible solution to define a small integer program, which is then optimized. As these techniques do not rely on problem structure to choose which variables to fix, they can be applied to any integer program and are available in commercial solvers such as CPLEX and Gurobi.

However, the models solved in real-world settings often have or contain a specific structure that a local search scheme can exploit to define small integer programs whose solution is likely to produce other high-quality solutions. Hewitt et al. (2010) develop such a scheme for the multi-commodity fixed-charge network flow problem. Combining exact and heuristic search techniques by embedding the solution of small integer programs in a heuristic search scheme has received quite a lot of attention recently; see De Franceschi et al. (2006), Schmid et al. (2008), Archetti et al. (2008), Savelsbergh and Song (2008), and Song and Furman (2010) for examples. Only the approach of Hewitt et al. (2010) produces a dual bound and performance guarantee for the solutions produced. None of the approaches are guaranteed to converge to an optimal solution.

Methods based on Benders' decomposition, such as Logic-based Benders' decomposition (Hooker and Ottosson (2003)), also produce primal solutions by solving small integer programs. The methods iteratively solve a first and second stage problem, with the solution of the first stage problem, which is a relaxation of the original problem, inducing a restriction of the original problem that is solved in the second stage. The solution of the restriction defines constraints that are added to the relaxation solved in the first stage, and, potentially, a primal solution. Problem structure is used to determine the form of the first and second stage problems solved in such an approach.

In this paper, we introduce a new approach that retains the strengths of integer programming based heuristic search methods, i.e., problem structure can be exploited to produce high-quality solutions quickly, but remedies their main weakness, i.e., being unable to provide performance guarantees. Moreover, while the method benefits significantly from the use of structure, it does not formally require it to produce an optimal solution. The approach uses an extended formulation of the problem whose solution automatically and dynamically yields a small restricted integer program to be solved next. The extended formulation is solved with a branch-and-price algorithm, which, when run to completion, produces a prov-

able optimal solution. However, it gives dual bounds throughout the execution, so even if terminated prematurely, it provides a performance guarantee.

The extended formulation specifies the structure of the small restricted integer programs to be solved. In this sense, it is different from techniques such as local branching and RINS. However, specifying the structure of the small integer programs to be solved has to be done only once and doing so allows for the transfer of problem specific knowledge, which may significantly enhance the ability to find high-quality solutions quickly. Furthermore, because only the structure of the small integer programs has to be specified, applying the approach to new problems requires little development time. The approach is different from Benders-based methods in that it solves restrictions that are guaranteed to produce a feasible solution and the process of creating and solving restrictions is loosely coupled, and, thus, amenable to parallelization.

The extended formulation is very different from typical column generation formulations that employ structurally different objects from the corresponding compact formulation; for example paths rather than arcs. Our extended formulation keeps the original variables from the compact formulation and augments them with an exponential number of additional variables that are used to define restrictions to solve. A related idea can be found in Fukasawa et al. (2006), where an exponential number of variables representing *q-routes* are added (as well as the necessary constraints linking edges to *q-routes*) to an edge-based formulation of the capacitated vehicle routing problem. As Fukasawa et al. observed too, by preserving the original compact formulation, it is possible to enrich it by preprocessing, cutting planes, or any other techniques normally used in a branch-and-cut framework. However, while our formulation exhibits similarities in form to theirs, it differs substantially in its purpose; our formulation is augmented with additional variables to speed up the search for high-quality primal solutions, whereas their formulation is augmented with additional variables to strengthen the dual bound.

We apply the approach to the multi-commodity fixed-charge network flow problem (MCFCNF), and illustrate how an understanding of problem structure can be used to develop an extended formulation that yields a computationally effective procedure. We demonstrate the effectiveness of the approach by comparing its performance on instances of the MCFCNF against both an exact solver, CPLEX, and an integer programming based heuristic search method (Hewitt et al. (2010)). For the instances used in the computational study the primal solution found by the proposed approach in 30 minutes is often optimal, better than the one pro-

duced by the integer programming based heuristic search method in the same time frame, and better than the one CPLEX produced in 6 hours. At the same time, the dual bound produced by the approach in 15 minutes is often close to the one CPLEX produces in 6 hours.

The main contribution of this paper is a new algorithm for solving integer programs that has the primal-side capabilities of a problem-specific integer programming based heuristic search method and the dual-side capabilities of an exact solver. Furthermore, because only the structure of the restricted integer programs has to be specified, the approach can be applied to new problems in much less time than traditional integer programming based heuristic search methods.

The remainder of this paper is organized as follows. In Section 2, we present the extended formulation that enhances the search for optimal solutions through the solution of small restricted integer programs, and a branch-and-price approach for solving it. In Section 3, we discuss its application to the MCFCNF and present computational results for a straightforward implementation of the approach and an analysis of these results which points to various ways in which the implementation can be improved. In Section 4, we present a more sophisticated implementation, engineered to speed up the search for high quality primal solutions, which exploits the loose coupling of the core components and can be executed in a multi-core or multi-node processing environment. In Section 5, we show a vastly improved performance of the more sophisticated implementation on the instances of the MCFCNF. Lastly, in Section 6, we present conclusions and future research directions.

2. Modeling the Search for Optimal Solutions

Consider problem P given by

$$\begin{array}{lrcll} \min & cx & + & dy & \\ \text{s.t.} & Ax & + & By & = b \\ & & & x \text{ real}, & y \text{ integer}. \end{array} \tag{1}$$

Let V_P denote the optimal value of P, let $S_P = \{(x, y) |\ Ax + By = b, x \text{ real}, y \text{ integer}\}$ be the set of feasible solutions to P and LP denote its linear relaxation.

For a given integer matrix N and a given integer vector q, both of appropriate dimension,

we define *restriction* $P_N(q)$ of P as:

$$\begin{array}{lrcll} \min & cx & + & dy & \\ \text{s.t.} & Ax & + & By & = b \\ & & & Ny & \leq q \\ & & & x \text{ real}, & y \text{ integer} \end{array} \tag{2}$$

with optimal value $V_N(q)$. We suppose that this restriction can be solved much faster than P. If the y variables are binary, q is a binary vector, and N is a matrix with components $n_{ij} \in \{0, \pm 1\}$, then the restriction defined by $Ny \leq q$ can enforce fixing components of y, such as $y_i \leq 0$, or enforce bounds on combinations of components of y, such as $y_i + y_j \leq 1$, or relations between components of y, such as $y_i \leq y_j$; such restrictions are likely to significantly speed up the solution of the integer program.

Letting $R = \{r |\ r = Ny \text{ for some } (x, y) \in S_P\}$, we have $V_P \leq V_N(r)\ \forall\ r \in R$ and $V_P = V_N(r^*)$ with $r^* = Ny_P^*$ for an optimal solution (x_P^*, y_P^*) to P. Thus, a strategy for finding an optimal solution to P is searching over the set R for vectors r and solving $P_N(r)$. The major advantage of such a strategy is that it produces a feasible solution to P each time a restriction $P_N(q)$ is solved.

Ideally, we would only solve restrictions $P_N(q)$ whose optimal value $V_N(q)$ is close to the optimal value V_P. Consequently, we would need an oracle that considers all vectors $r \in R$, but returns only those with $V_N(r) \approx V_P$. The role of this oracle is thus similar to the role of the pricing problem in column generation: consider all columns, but return only columns with negative reduced costs (for a minimization problem).

Therefore, we next assume that we know the entire set R and build a model that extends the formulation of P so as to incorporate both choosing a vector r from R and solving the resulting restriction $P_N(r)$. Specifically, we define the master problem MP:

$$\begin{array}{lrcllll} \min & cx & + & dy & & & \\ \text{s.t.} & Ax & + & By & & & = b \\ & & & Ny & -Rz & & \leq 0 \\ & & & & 1z & & = 1 \\ & & & x \text{ real}, & y \text{ integer}, z \text{ binary}, & & \end{array} \tag{3}$$

where the binary variables z in MP represent the choice of vector r for which the restriction $P_N(r)$ should be solved, and let MLP denote its linear relaxation. In addition, by replacing $Ny - Rz \leq 0$ with $Ny - Rz = 0$, we obtain problem $MP_=$ with linear relaxation $MLP_=$. We observe that MP and $MP_=$ are equivalent reformulations of P, i.e., $V_{MP_=} = V_{MP} = V_P$, and that their linear relaxations are no weaker than LP, i.e., $V_{MLP_=} \geq V_{MLP} \geq V_{LP}$, where

V_X denotes the optimal value of the linear program X. Although the primary purpose of MP and $MP_=$ is to aid in the search for high-quality primal solutions, MLP may provide a tighter bound on V_P than LP, and $MP_=$ may have an even tighter linear relaxation, i.e., we can have $V_{LP} < V_{MLP} < V_{MLP,=}$; examples are given in the Online Supplement.

As the set R may have an exponential number of elements, column generation forms the basis of our solution approach. While $MLP_=$ may provide a tighter bound on V_P than MLP, it may be more difficult to solve. For example, we will see that when solving MLP for MCFCNF, the non-negative dual variables associated with $Ny - Rz \leq 0$ yield a relatively easy pricing problem, whereas the corresponding pricing problem for $MLP_=$ is much harder. Thus we restrict the presentation of our approach to MP, although all the steps can be easily adapted to $MP_=$.

Because MLP will be solved by column generation, MP will be solved by branch-and-price. Next, we define the main components of that branch-and-price algorithm, i.e., a restricted master problem, a pricing problem, and a branching scheme. With $\bar{R} \subseteq R$, the restricted master problem RMP is defined as:

$$\begin{array}{llllllll} \min & cx & + & dy & & & & \\ \text{s.t.} & Ax & + & By & & & = b & \\ & & & Ny & & -\bar{R}z & \leq 0 & (\pi) \\ & & & & & 1z & = 1 & (\alpha) \\ & & & & x \text{ real}, & y \text{ integer}, z \text{ binary}, & & \end{array} \tag{4}$$

with dual variables $\pi \leq 0$ and α unrestricted for its LP relaxation, $RMLP$.

To ensure that we solve MP and in turn find an optimal solution (x_P^*, y_P^*) to P, we need a mechanism for generating columns to add to $\bar{R}$ that will eventually produce $r^* = Ny_P^*$. To search for a column $r \in R \setminus \bar{R}$ with negative reduced cost, i.e. with $\pi r - \alpha < 0$, we define the pricing problem:

$$\begin{array}{llllllll} -\alpha + & \min & & & & \pi r & & \\ & \text{s.t.} & Ax & + & By & & = b & \\ & & & & Ny & -r & = 0 & \qquad G(\pi, \alpha) \\ & & & & x \text{ real}, & y \text{ integer}, & r \text{ integer}. & \end{array}$$

Note that valid inequalities $fx + gy \leq \gamma$ for P can be used to strengthen both $RMLP$ and $G(\pi, \alpha)$. Also, if we know a primal solution $(\bar{x}, \bar{y})$ of P with value $v = c\bar{x} + d\bar{y}$, then we can add the constraint $cx + dy \leq v$ to the pricing problem to reflect that we are only interested in restrictions whose optimal solution is better than the incumbent. This column

generation process also yields a dual bound. Specifically, if $\tilde{r}$ is an optimal solution to the pricing problem, then we have $V_{RMLP} + \pi\tilde{r} - \alpha \leq V_{MLP} \leq V_{MP} = V_P$.

However, by sacrificing a provable dual bound at some iterations, we can use the pricing problem to enhance our search in other ways. In particular, if for some $\epsilon > 0$ we add the constraint $\pi r - \alpha \leq -\epsilon$ to the pricing problem then it will return a column with negative reduced cost if one exists and we can use a different objective function. For example, we can solve the pricing problem with the objective function of P, $\min cx + dy$. Or, we can use an objective function that guides the pricing problem towards returning elements of R that are unlike ones we have already generated. For example, with $t_j = \sum_{r^i \in \bar{R}} r^i_j$, we can minimize the objective $\sum_j r_j t_j$.

The last component to define is a branching scheme to handle the situation where the column generation process has solved MLP but the optimal solution $(x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP})$ to $RMLP$ is fractional. Because we have assumed that N is an integer matrix and y is a vector of integer variables, Ny must also be a vector of integers. We consider two cases:

1. An element of Ny^*_{RMLP} is non-integer. For this case, we choose an index i such that $(Ny^*_{RMLP})_i = v$ is non-integer and enforce $(Ny)_i \leq \lfloor v \rfloor$ on one branch and $(Ny)_i \geq \lceil v \rceil$ on the other. We enforce $(Ny)_i \leq \lfloor v \rfloor$ ($(Ny)_i \geq \lceil v \rceil$) in the pricing problem by enforcing $r_i \leq \lfloor v \rfloor (\geq \lceil v \rceil)$.

2. All elements of Ny^*_{RMLP} are integer, but (x^*_{RMLP}, y^*_{RMLP}) contains non-integers. For this case, we create the branches $z_r = 0$ and $z_r = 1$ for an r with the largest non-integer value $z^*_{r,RMLP}$. However, we immediately evaluate and prune the branch with $z_r = 1$ by solving $P_N(r)$.

   Note that branching on a z variable is thus equivalent to removing columns from $\bar{R}$. Therefore, if we have already solved $P_N(r^1)$, we should add constraints to the pricing problem to exclude r^1. In fact, we can do more, because we know that $r^2 \leq r^1 \Rightarrow V_N(r^2) \geq V_N(r^1)$. By adding the constraint $\sum_j (r_j - r^1_j) \geq 1$ to the pricing problem, we ensure that it returns an r that satisfies $r_j > r^1_j$ for some element j, which excludes elements from R whose restrictions cannot yield a better solution than the optimal solution to $P_N(r^1)$.

We summarize the branch-and-price guided search scheme (BPGS) in Algorithm 1.

The most important feature of our extended formulation is that solutions, r, to the pricing problem induce restrictions, $P_N(r)$, that may have an optimal solution of high-quality for P.

Algorithm 1 Branch-and-price guided search (BPGS)

1: set $\bar{R} = \emptyset$
2: **while** MP has not been solved **do**
3: select an unevaluated node
4: **while** MLP associated with node has not been solved **do**
5: solve $RMLP$ associated with node
6: solve the pricing problem to find an improving vector r to add to $\bar{R}$
7: solve $P_N(r)$
8: **end while**
9: branching if necessary
10: **end while**

We take advantage of this feature in Step 7 by immediately solving $P_N(r)$ for the solution, r, to the pricing problem. We next study computationally the potential of the proposed approach to produce both high quality solutions quickly and strong dual bounds.

3. Applying BPGS to MCFCNF

Before we discuss how we apply the scheme, we briefly review the arc-based formulation of the MCFCNF. Let $D = (V, A)$ be a network with node set V 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. We let δ_i^k indicate whether node i is a source ($\delta_{s(k)}^k = 1$), a sink ($\delta_{t(k)}^k = -1$) or an intermediate node ($\delta_i^k = 0$) for commodity k. 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 assume that $f_{ij} \geq 0$ and $c_{ij} \geq 0$ for all $(i, j) \in A$. 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 MCFCNF 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

$$\sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(j,i) \in A} x_{ji}^k = \delta_i^k \quad \forall i \in V,\ \forall k \in K, \tag{5}$$

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

$$y_{ij} \in \{0, 1\} \ \ \forall (i, j) \in A. \tag{7}$$

We focus on the harder variant of the problem where a commodity has to be routed along a single path. Hence, we have

$$x_{ij}^k \in \{0,1\} \quad \forall k \in K,\ \forall (i,j) \in A. \tag{8}$$

The objective is to minimize the sum of fixed and variable costs. Constraints (5) ensure flow balance. Constraints (6) 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 MCFCNF has a weak LP relaxation and can be strengthened by disaggregating the coupling constraints to

$$x_{ij}^k \leq y_{ij} \quad \forall k \in K,\ \forall (i,j) \in A. \tag{9}$$

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

Although much research has been done on strengthening the formulation of MCFCNF, usually through *cover*-based valid inequalities, its weak linear relaxation still makes realistic-size instances hard to solve to optimality. Thus, much of the research involving large-scale instances has focused on developing heuristic search methods for producing primal solutions of high quality. Metaheuristics (Crainic et al. (2000),Crainic and Gendreau (2002),Ghamlouch et al. (2003), Ghamlouch et al. (2004)) have proven (at least computationally) to be some of the most effective techniques for producing good primal solutions to instances of MCFCNF. Hewitt et al. (2010) present an integer programming based heuristic search method, which is shown to be competitive if not superior to existing techniques.

The integer programming based heuristic search method presented in Hewitt et al. (2010) served as the inspiration for this work, as it prompted the study of what problem-specific steps inherent in an integer programming based local search method can be generalized, resulting in a method that is closer to *black box* methods such as local branching and RINS.

To apply BPGS to MCFCNF, we must choose a matrix N, which will in turn dictate the structure of the extended formulation, MP, and restrictions, $P_N(r)$, we solve. We first choose the matrix $N = I$ to indicate how the approach can generalize "variable fixing" heuristics. Hence, $P_N(r)$ consists of the extra constraints $y_{ij} \leq r_{ij} \quad \forall (i,j) \in A$ and $R = \{r \mid r_{ij} = y_{ij} \quad \forall (i,j) \in A \text{ for some } (x,y) \in S_P\}$ where S_P is the set of feasible solutions to P.

Given the dual variables α and π_{ij}, the pricing problem is

$$\min \sum_{(i,j)\in A} \pi_{ij} r_{ij} - \alpha$$

$$\sum_{j:(i,j)\in A} x_{ij}^k - \sum_{j:(j,i)\in A} x_{ji}^k = \delta_i^k \quad \forall i \in V,\ \forall k \in K, \tag{10}$$

$$\sum_{k\in K} d^k x_{ij}^k \le u_{ij} y_{ij} \ \ \forall (i,j) \in A, \tag{11}$$

$$y_{ij} = r_{ij} \quad \forall (i,j) \in A, \tag{12}$$

$$r_{ij} \in \{0,1\} \ \ \forall (i,j) \in A, \tag{13}$$

$$x_{ij}^k \in \{0,1\} \quad \forall k \in K,\ \forall (i,j) \in A. \tag{14}$$

Note that while the feasible region of the pricing problem, when projected onto the (x, y) variables, is the same as that of the original problem, the objective function is significantly different and makes the problem much easier to solve than the original. Because it is a minimization problem and the dual variables π are nonpositive, the objective function encourages the variables r_{ij} to be set to 1 even in solutions to the linear programming relaxation. In fact, there is an optimal solution with $r_{ij} = 1\ \forall (i,j) \in A$. This is not desirable because the restricted problem associated with this r will be too loosely constrained to be solvable. To eliminate this problem we use the structure of an optimal solution to MCFCNF to further constrain the pricing problem without eliminating solutions r^* associated with optimal solutions (x_P^*, y_P^*) to P. In particular, given our assumption that $f_{ij} \ge 0\ \forall (i,j) \in A$, we can assume that in an optimal solution an arc is installed only if it is used by a commodity. Thus we add the inequalities

$$r_{ij} \le \sum_{k\in K} x_{ij}^k \quad \forall (i,j) \in A \tag{15}$$

to the pricing problem. We also observe that $c_{ij} \ge 0\ \forall (i,j) \in A$ implies that, in an optimal solution, the route a commodity takes from its source to its sink will not contain any cycles. Thus, to eliminate cycles in the routes chosen for commodities by the pricing problem, we add the constraints

$$\sum_{j:(i,j)\in A} x_{ij}^k \le 1 \quad \forall i \in V,\ \forall k \in K, \tag{16}$$

$$\sum_{i:(i,s(k))\in A} x_{is(k)}^k \le 0 \quad \forall k \in K, \tag{17}$$

and

$$\sum_{j:(t(k),j)\in A} x^k_{t(k)j} \leq 0 \quad \forall k \in K. \tag{18}$$

The first set ensures that a commodity does not leave a node multiple times, whereas the next two sets ensure that a commodity never enters (leaves) its source (sink). We note that constraints (15 – 18) are only necessary because there is a feasible solution to the pricing problem with $r_{ij} = 1 \; \forall (i,j) \in A$. In other applications, this may not be necessary.

We have also chosen to try to strengthen the bound V_{MLP} produced by BPGS through the use of well-known valid inequalities for MCFCNF. Specifically, we augment BPGS to a branch-price-and-cut algorithm, adding steps that dynamically add the disaggregated constraints (9) and the *cover* inequalities $\sum_{k\in C} x^k_{ij} \leq |C| - 1$ to $RMLP$, where a cover with respect to an arc (i,j) is a set $C \subseteq K$ such that $\sum_{k\in C} d^k > u_{ij}$ (Nemhauser and Wolsey (1988)).

Also, because our extended formulation retains the variables of the compact formulation, we can apply well-known pre-processing rules for the MCFCNF when solving MP. In particular, we use the observation that for the single path variant of MCFCNF, commodity k cannot use arc (i,j) if its quantity exceeds the arc's capacity. Thus, we fix $x^k_{ij} = 0$ when $d^k > u_{ij}$, and we can do so when we solve instances of $RMLP$, $P_N(r)$, and our pricing problem, $G(\pi, \alpha)$.

3.1. Computational Results

We next study computationally whether this application of BPGS to MCFCNF produces high quality solutions quickly and strong dual bounds. To do so, we implemented BPGS in C++, with CPLEX 11.2 used as the MIP/LP solver. All experiments were performed on a machine with 8 Intel Xeon cores running at 2.26 GHz with 24 GB RAM. Furthermore, the solution of a pricing problem is terminated as soon as a column with negative reduced cost is found and restrictions $P_N(r)$ are solved with an optimality tolerance of 1% and a time limit of one minute.

Due to their diversity in size and difficulty, for all experiments in this paper we use the MCFCNF instances from Ghamlouch et al. (2004) in the set identified as C. The instances have been used to benchmark the performance of meta-heuristics designed for the MCFCNF variant in which a commodity may be split across multiple paths ($0 \leq x^k_{ij} \leq 1$). However, the instances remain feasible when splitting is not allowed. The instances are classified as

follows: F denotes instances where the ratio of fixed to variable cost for an arc is high and V otherwise, T denotes instances that are tightly capacitated relative to total demand and L otherwise. Finally, the naming scheme is $\#nodes - \#arcs - \#commodities - (F|V) - (T|L)$.

Because BPGS is an exact method we first compare it with an exact solver. Specifically, we compare the primal solutions and dual bounds BPGS produces in 30 minutes with those CPLEX produces when executed on a single processor for 30 minutes with an optimality tolerance of 1% and other settings left at their default values. To determine the quality of the primal solutions BPGS produces, we also compare them with the dual bound CPLEX produces when run for 6 hours. For the CPLEX experiments and when BPGS solves pricing problems $G(\pi, \alpha)$ and restrictions $P_N(r)$ the disaggregated constraints $x_{ij}^k \leq y_{ij}$ were provided to CPLEX as *user cuts*. This functionality allows CPLEX to dynamically determine which of these constraints should be added to the formulation.

We report in Table 1

- the best primal solution found by CPLEX in 30 minutes (CPLEX 30 Minute Primal), the dual bound produced by CPLEX after 30 minutes (CPLEX 30 Minute Dual), the dual bound produced by CPLEX after 6 hours (CPLEX 6 Hour Dual),
- the best primal solution found by BPGS in 30 minutes (BPGS 30 Minute Primal), the dual bound produced by BPGS after 30 minutes (BPGS 30 Minute Dual),
- a comparison (Primal Gap) of the quality of primal solutions produced by BPGS and CPLEX in the form of
  100×(BPGS 30 Minute Primal−CPLEX 30 Minute Primal)/(BPGS 30 Minute Primal),
  the optimality gap (Opt. Gap) of the solution produced by BP with respect to the dual bound produced by CPLEX after 6 hours, calculated as
  100 × (BPGS 30 Minute Primal − CPLEX 6 Hour Dual)/(BPGS 30 Minute Primal),
  and a comparison (Dual Gap) of the dual bounds produced by BPGS and CPLEX in 30 minutes in the form of
  100 × (BPGS 30 Minute Dual − CPLEX 30 Minute Dual)/(BPGS 30 Minute Dual).

We see that CPLEX outperformed BPGS in terms of producing high quality primal solutions. However, BPGS produced a dual bound that is nearly as strong (on average only 1.57% worse) as the bound produced by CPLEX, even though BPGS uses fewer classes of cuts. To understand why BPGS fell short in terms of producing high-quality primal solutions,

Table 1: Primal Comparison of BPGS with CPLEX

	CPLEX			BPGS				
Instance	30 Minute Primal	30 Minute Dual	6 Hour Dual	30 Minute Primal	30 Minute Dual	Primal Gap	Opt. Gap	Dual Gap
20-230-40-V-L	423,933	423,166	423,830	426,234	421,959	0.54	0.72	-0.29
20-230-40-V-T	399,148	398,611	396,606	398,870	391,914	-0.07	0.06	-1.71
20-230-40-F-T	669,587	667,791	668,645	675,007	661,318	0.80	1.07	-0.98
20-230-200-V-L	95,791	90,704	93,373	101,119	88,795	5.27	10.30	-2.15
20-230-200-F-L	143,442	129,199	134,808	163,226	130,065	12.12	20.85	0.67
20-230-200-V-T	99,334	95,767	97,371	106,938	94,905	7.11	10.45	-0.91
20-230-200-F-T	141,087	127,913	133,398	153,814	129,761	8.27	16.84	1.42
20-300-40-V-L	433,320	429,303	429,963	432,009	428,546	-0.30	0.63	-0.18
20-300-40-F-L	599,917	592,755	595,069	617,954	583,204	2.92	4.08	-1.64
20-300-40-V-T	502,512	500,982	500,409	506,855	493,016	0.86	1.16	-1.62
20-300-40-F-T	648,314	638,717	642,075	651,615	636,245	0.51	1.98	-0.39
20-230-200-V-L	76,886	72,983	74,142	83,589	72,496	8.02	12.69	-0.67
20-300-200-F-L	123,055	109,338	112,374	137,941	109,927	10.79	20.74	0.54
20-300-200-V-T	76,709	74,053	75,221	82,520	73,410	7.04	10.26	-0.88
20-300-200-F-T	110,940	103,046	105,116	124,298	101,687	10.75	17.10	-1.34
30-520-100-V-L	54,466	54,077	53,919	57,397	52,743	5.11	5.78	-2.53
30-520-100-F-L	96,964	86,789	93,218	105,760	86,981	8.32	17.94	0.22
30-520-100-V-T	53,957	53,470	53,509	55,167	52,054	2.19	3.08	-2.72
30-520-100-F-T	100,031	95,155	97,883	108,575	92,896	7.87	12.36	-2.43
30-520-400-V-L	116,422	109,917	112,045	277,465	109,218	58.04	60.39	-0.64
30-520-400-F-L	155,832	143,809	147,189	400,727	140,085	61.11	64.11	-2.66
30-520-400-V-T	117,753	113,432	114,647	236,471	113,018	50.20	52.03	-0.37
30-520-400-F-T	164,341	147,508	150,341	360,620	144,512	54.43	59.10	-2.07
30-700-100-V-L	48,064	47,594	47,585	50,624	46,817	5.06	5.99	-1.66
30-700-100-F-L	61,171	58,407	59,869	64,047	57,488	4.49	8.81	-1.60
30-700-100-V-T	47,781	47,395	47,283	50,937	44,523	6.20	6.95	-6.45
30-700-100-F-T	56,804	56,129	56,274	59,649	54,148	4.77	5.90	-3.66
30-700-400-V-L	101,298	95,002	96,876	257,970	94,623	60.73	63.17	-0.40
30-700-400-F-L	146,650	127,675	130,984	430,758	117,422	65.96	70.36	-8.73
30-700-400-V-T	98,506	93,014	94,432	227,634	91,780	56.73	59.14	-1.34
30-700-400-F-T	140,343	125,887	128,027	359,041	123,885	60.91	64.94	-1.62
Average						18.93	22.22	-1.57

we next group the results by the number of commodities in an instance as reported in Table 2. For a given number of commodities, we report the average optimality gap reported in Table 1 over all instances with that number of commodities (Avg. Opt Gap) the average number of times a restriction $P_N(q)$ was solved (Avg. Number of Solves) and the average number of times solving a restriction produced an improved feasible solution (Avg. Number of Imp. Solutions). Recall that solving a restriction is terminated prematurely when the time limit is reached and that even if a restriction is solved within the time limit it may not return a solution because the best-known solution value is used as a cutoff for the objective value.

We see that when BPGS only solves a small number of restrictions within the 30 minutes of run time, it is unable to produce high-quality solutions. This may be an indication that the restrictions produced by the pricing problem ultimately converge to ones which induce high quality solutions, but that the time limit of 30 minutes is not enough for instances with a large number of commodities to reach that point. This suggests two enhancements that may

Table 2: Results by Number of Commodities, $|K|$, in an Instance

$\|K\|$	# Instance	Avg. Opt. Gap	Avg. Number of Solves	Avg. Number of Imp. Solutions
40	7	1.39	550.00	10.43
100	8	8.35	71.13	4.50
200	8	14.90	16.13	4.00
400	8	61.65	11.63	3.38

improve its performance: (1) exploit the loose coupling between solving the restricted master problem, the pricing problems, and the restrictions by developing a parallel implementation, and (2) be more selective when choosing restrictions to solve. These enhancements are discussed in more detail in the next section.

4. Enhancing Branch-and-Price Guided Search

As observed above, the solution of the restricted master problem, of the pricing problems, and of the restrictions is only loosely coupled and lends itself naturally to parallelization: dedicate a single processor to managing the solution process and solving the linear programming relaxation of the active restricted master problem and one or more processors to the solution of restrictions. This is exactly what we have done.

In the basic algorithm, as outlined in Algorithm 1, every restriction returned by the pricing problem is immediately evaluated. In a parallel implementation, where pricing problems and restrictions are solved on different processors, a mechanism for selecting restrictions to solve is a necessity. Given a set of restrictions $\bar{R} \subseteq R$ and the subset $\bar{R}_S \subset \bar{R}$ of restrictions that have already been solved, a *primal solution construction* (PC) process produces feasible solutions by choosing an $r \in \bar{R} \setminus \bar{R}_S$ and solving $P_N(r)$. More specifically, if there exists an $r \in \bar{R} \setminus \bar{R}_S$ such that $0 < z^*_{r,RMLP} < 1$, then an r with the largest value of $z^*_{r,RMLP}$ is selected, and if no such r exists, then r is selected randomly from $\bar{R} \setminus \bar{R}_S$. We will refer to this parallel implementation of BPGS as pBPGS.

To further exploit the advantages of a parallel implementation, a number of local search ideas have been incorporated as well. Observe that $\tilde{r} \geq r^{BEST}$ implies that the optimal solution of $P_N(\tilde{r})$ is at least as good as the optimal solution of $P_N(r^{BEST})$, or, that $V_N(\tilde{r}) \leq V_N(r^{BEST})$. This suggests solving $P_N(\tilde{r})$ with an $\tilde{r} > r^{BEST}$. A *primal solution improvement* (PI) process does that and uses a variety of schemes for constructing $\tilde{r}$, where the scheme is chosen randomly albeit with a bias towards schemes that have contributed the most to

solution improvement so far (similar to the biased roulette wheel approach used in Hewitt et al. (2010)).

Scheme augment-best: Let u_i represent an upper bound on the value $(Ny)_i$ for $(x, y) \in S_P$. To obtain a $\tilde{r} > r^{BEST}$, we start from $\tilde{r} = r^{BEST}$ and then randomly choose a subset of indices i with $r_i^{BEST} < u_i$. We then randomly draw an integer δ from $[1, u_i - r_i^{BEST}]$ and set $\tilde{r}_i = \tilde{r}_i + \delta$. While we choose the indices to increment randomly (to diversify the search), we do so with a bias based on metrics we believe will lead to neighborhoods that are likely to contain good solutions. Specifically, we use two metrics, both based on the solution to $RMLP$, to choose the indices i for which we increment $\tilde{r}_i$. The first metric is the dual variable π_i associated with $(Ny - \bar{R}z \leq 0)_i$ and the second is the value $(\bar{R}z^*_{RMLP})_i$. The larger these values are, the more likely it is that we choose index i. We refer to solutions created in this way as belonging to the *augment-best* neighborhood.

The next two schemes use the concept of path-relinking (Glover (1998)), i.e., by combining the structural information from two good solutions we may produce a restriction that yields an even better solution. In particular, we use two known restrictions to get a new restriction $\tilde{r}$. Given r^{BEST} and another restriction r, we create $\tilde{r}$ by setting $\tilde{r}_i = max(r_i, r_i^{BEST})$. We have two schemes for choosing the restriction r with which we relink.

Scheme priced-r: Choose a restriction r that was produced by the pricing problem. We refer to this neighborhood as the *priced-r* neighborhood.

Scheme solution-r: Choose a restriction r that is induced by another solution. Specifically, we set $r = Ny_P^1$ for a solution (x_P^1, y_P^1) that was found earlier either on this processor or potentially on another processor. We refer to this neighborhood as the *solution-r* neighborhood.

With these local search ideas care has to be taken not to create a restriction $\tilde{r}$ with $\|\tilde{r}\| \approx \sum_i u_i$, as the resulting restriction may be too loosely constrained to be solved quickly. In our current implementation, and for all our local search schemes, we ensure that all $\tilde{r}$ satisfy $\|\tilde{r}\|_1 \leq \gamma$ where γ is an algorithm parameter.

Our parallel implementation not only allows local search schemes to be run concurrently on different processors, but also allows for different search strategies. On each processor p dedicated to running a local search scheme, we define a range $[l_p, u_p]$ and each time a

restriction $\tilde{r}$ is to be created, γ is drawn randomly from that range. Thus we can have some processors search small neighborhoods and others search large neighborhoods. We will refer to the parallel implementation of BPGS enhanced with local search schemes as pBPGS$^+$.

We summarize the different processes, their description, and which algorithms use them (indicated by a 'Yes' in the appropriate cell) in Table 3. We note that whereas the BP process executed by BPGS also solves restrictions $P_N(r)$, as seen in Algorithm 1, in pBPGS and pBPGS$^+$, the BP process does not perform that step (Step 7).

Table 3: Process Summary

Name	Description	BPGS	pBPGS	pBPGS$^+$
BP	Branch-and-price tasks, including managing branch-and-bound tree, solving pricing problems, and solving linear relaxations.	Yes	Yes	Yes
PC	Solves $P_N(r)$ for $r \in \bar{R}$	No *	Yes	Yes
PI	Solves $P_N(\tilde{r})$ for $\tilde{r}$ created with local search schemes. Note $\tilde{r}$ not necessarily in $\bar{R}$.	No	No	Yes
	* done in BP			

5. Computational Results

We next study computationally the impact of these heuristics and their parallel execution. Again, because pBPGS and pBPGS$^+$ are exact algorithms, we chose to benchmark them against the commercial MIP solver CPLEX 11.2. Like BPGS, pBPGS and pBPGS$^+$ were coded in C++ and all other statements made above regarding hardware platform and runtime settings remain valid. pBPGS consists of two processes (one BP and one PC), while pBPGS$^+$ consists of five processes (one BP, one PC, and three PI). Message passing between processes was implemented via a combination of MPI (Gropp (1999)) and text files. We chose three PI processes to enable searching neighborhoods of three potentially different sizes. For application to the MCFCNF, we set the lower bounds, l_1, l_2, l_3 on γ for each PI process to the same value, $.2|A|$, where $|A|$ represents the number of arcs in the network. We then set $u_i = (.2 + .15 * i)|A|$ for $i = 1, 2, 3$. For the following experiments we allowed pBPGS and pBPGS$^+$ to run for 30 minutes.

Unless otherwise noted, computation times are reported in seconds. In Section 5.1 we study whether the parallelization and enhancements discussed in Section 4 lead to algorithms with better primal-side performance than BPGS. Next, in Section 5.2 we compare the primal performance of pBPGS$^+$ with CPLEX and examine the dual-side performance of pBPGS$^+$ in depth. Then, in Section 5.3 we compare the primal-side performance of pBPGS$^+$ with the

IP-based local search heuristic presented in Hewitt et al. (2010). Lastly, in Section 5.4 we study applying pBPGS$^+$ to MCFCNF using a different restriction than the one presented in Section 3.

5.1. BPGS vs. pBPGS$^+$

We first study whether the parallelization and local search schemes discussed above enhance the performance of BPGS. We report in Table 4

- the best primal solution found by BPGS after 30 minutes (Primal),
- the best primal solution found by pBPGS after 30 minutes (Primal), the percentage quality increase over BPGS (BPGS Gap) computed as 100 $\times$ (pBPGS Primal $-$ BPGS Primal)/(pBPGS Primal), the optimality gap (Opt. Gap), and the time needed to produce a better solution than BPGS (Time to Beat),
- the best primal solution found by pBPGS$^+$ after 30 minutes (Primal), the percentage quality increase over pBPGS (pBPGS Gap) computed as 100 $\times$ (pBPGS$^+$ Primal $-$ pBPGS Primal)/(pBPGS$^+$ Primal), the optimality gap (Opt. Gap), and the time needed to produce a better solution than pBPGS (Time to Beat).

For these results, optimality gaps are calculated with respect to a dual bound produced by CPLEX when run for 6 hours.

The results show that solving pricing problems and restrictions produced by the pricing problem in parallel, i.e., pBPGS, leads to significantly better primal solutions. Incorporating local search schemes leads to another significant increase in solution quality. We also note that pBPGS$^+$ needs little time to produce better solutions than pBPGS.

However, while, on average, the solutions found when executing pBPGS$^+$ are 5% better than when executing pBPGS, most of this improvement is found in the instances with 400 commodities, as can be seen in Table 5, where, for a given number of commodities, we average the optimality gap for BPGS, pBPGS and pBPGS$^+$ (as reported in Table 4) over all instances with that number of commodities.

The poor performance of BPGS and pBPGS on the 400-commodity instances can be attributed to the time required to solve the pricing problems and the linear programming relaxations for these instances. In Table 6, we report, averaged over instances with a given number of commodities, the number of pricing problem solved by pBPGS (pBPGS Avg.

Table 4: Primal Comparison of BPGS, pBPGS, and pBPGS$^+$

	BPGS	pBPGS				pBPGS$^+$			
Instance	Primal	Primal	BPGS Gap	Opt. Gap	Time to Beat	Primal	pBPGS Gap	Opt. Gap	Time to Beat
20-230-40-V-L	426,234	424,361	-0.44	0.13	75	423,933	-0.10	0.02	4
20-230-40-V-T	398,870	399,810	0.24	0.80		398,870	-0.24	0.57	2
20-230-40-F-T	675,007	677,331	0.34	1.28		668,699	-1.29	0.01	7
20-230-200-V-L	101,119	99,669	-1.45	6.32	321	94,843	-5.09	1.55	130
20-230-200-F-L	163,226	145,396	-12.26	7.28	367	138,476	-5.00	2.65	50
20-230-200-V-T	106,938	101,318	-5.55	3.90	645	98,947	-2.40	1.59	63
20-230-200-F-T	153,814	146,908	-4.70	9.20	610	139,889	-5.02	4.64	154
20-300-40-V-L	432,009	432,037	0.01	0.48		430,314	-0.40	0.08	3
20-300-40-F-L	617,954	613,908	-0.66	3.07	853	597,059	-2.82	0.33	7
20-300-40-V-T	506,855	508,653	0.35	1.62	299	501,889	-1.35	0.29	10
20-300-40-F-T	651,615	650,686	-0.14	1.32	300	643,395	-1.13	0.21	7
20-230-200-V-L	83,589	79,215	-5.52	6.40	319	76,333	-3.78	2.87	77
20-300-200-F-L	137,941	126,944	-8.66	11.48	325	118,204	-7.39	4.93	82
20-300-200-V-T	82,520	78,773	-4.76	4.51	925	76,986	-2.32	2.29	35
20-300-200-F-T	124,298	116,045	-7.11	9.42	629	109,776	-5.71	4.25	54
30-520-100-V-L	57,397	57,397	0.00	6.06	358	54,387	-5.53	0.86	10
30-520-100-F-L	105,760	105,045	-0.68	11.26	1,761	96,277	-9.11	3.18	110
30-520-100-V-T	55,167	56,031	1.54	4.50		53,812	-4.12	0.56	30
30-520-100-F-T	108,575	102,923	-5.49	4.90	806	99,355	-3.59	1.48	54
30-520-400-V-L	277,465	122,604	-126.31	8.61	344	114,867	-6.74	2.46	254
30-520-400-F-L	400,727	175,011	-128.97	15.90	384	152,837	-14.51	3.70	535
30-520-400-V-T	236,471	120,213	-96.71	4.63	328	116,730	-2.98	1.78	317
30-520-400-F-T	360,620	176,655	-104.14	14.90	363	155,982	-13.25	3.62	529
30-700-100-V-L	78,697	51,313	-53.37	7.27	612	47,883	-7.16	0.62	13
30-700-100-F-L	64,047	63,703	-0.54	6.02	639	60,384	-5.50	0.85	183
30-700-100-V-T	50,937	50,740	-0.39	6.81	598	47,686	-6.40	0.85	20
30-700-100-F-T	86,207	59,654	-44.51	5.67	455	56,809	-5.01	0.94	96
30-700-400-V-L	257,970	106,932	-141.25	9.40	145	98,850	-8.18	2.00	223
30-700-400-F-L	430,758	154,019	-179.68	14.96	360	138,376	-11.30	5.34	625
30-700-400-V-T	227,634	103,734	-119.44	8.97	362	97,352	-6.56	3.00	284
30-700-400-F-T	359,041	148,276	-142.14	13.66	348	133,759	-10.85	4.29	561
Average			-38.46	6.80	501		-5.32	1.99	146

Number Priced), the time required to solve those pricing problems (pBPGS Avg. Pricing Solve Time), the time required to solve the resulting linear programming relaxations (pBPGS Avg. $RMLP$ Solve time), the number of restrictions solved by pBPGS (pBPGS Avg. Number $P_N(q)$ Solved), the number of improving solutions found by doing so (pBPGS Avg. Number Imp. Solutions Found), the number restrictions solved by pBPGS$^+$ (pBPGS$^+$ Avg. Number $P_N(q)$ Solved), and the number of improving solutions found by doing so (pBPGS$^+$ Avg. Number Imp. Solutions Found). We see that the time required to solve pricing problems and linear programming relaxations is significantly larger for the instances with 400 commodities, which limits the number of pricing problems solved. (The times required to solve pricing problems and linear programming relaxations for pBPGS$^+$ are similar to those for pBPGS.)

We see that pBPGS$^+$ solves many more restrictions than pBPGS and, subsequently, finds more improving solutions, except for the instances with 40 commodities. There are two reasons why pBPGS$^+$ is solving more restrictions. Firstly, pBPGS$^+$ was executed on 5

Table 5: Optimality Gaps by Number of Commodities in an Instance

$\|K\|$	# Instance	Avg. BPGS Opt. Gap	Avg. pBPGS Opt. Gap	Avg. pBPGS$^+$ Opt. Gap
40	7	1.39	1.24	0.22
100	8	8.35	6.56	1.17
200	8	14.90	7.31	3.10
400	8	61.65	11.38	3.27

Table 6: Results by Number of Commodities in an Instance

	pBPGS					pBPGS$^+$	
$\|K\|$	Avg. Number Priced	Avg. Pricing Solve Time	Avg. *RMLP* Solve Time	Avg. Number $P_N(q)$ Solved	Avg. Number Imp. Solutions Found	Avg. Number $P_N(q)$ Solved	Avg. Number Imp. Solutions Found
40	601.29	2.14	.11	601.29	10.14	4,042.86	6.29
100	109.23	12.47	3.14	109.23	4.63	1,456.13	15.25
200	118.14	3.91	10.03	47.38	3.50	337.88	17.25
400	25.48	21.13	43.89	25.48	3.50	138.75	8.00

processors whereas pBPGS was executed on 2 processors. Secondly, the number of restrictions that pBPGS can solve is bounded by the number of pricing problems solved, whereas the local search schemes of pBPGS$^+$ create restrictions independent of the pricing problems solved. The fact that pBPGS$^+$ finds fewer improving solutions for instances with 40 commodities is because the local search schemes enabled pBPGS$^+$ to quickly find a provably optimal solution and terminate.

As each PI process uses three different schemes for creating restrictions that represent a neighborhood of the best-known solution, we next analyze the performance of each. The *augment-best* neighborhood was responsible for the majority of the solution improvement over all instances at 89% followed by the *priced-r* neighborhood at 9% and the *solution-r* neighborhood at 2%. One reason that the *solution-r* neighborhood is not that effective may be the lack of intelligence in the selection of the solution to "link" the best known solution with. Currently, that solution is selected randomly without considering how similar of dissimilar it is from the best known solution.

Regardless, we conclude that pBPGS$^+$ is a significant upgrade over both BPGS and pBPGS and next benchmark it against CPLEX and an IP-based local search heuristic.

5.2. pBPGS$^+$ vs. CPLEX

In our next set of experiments, we ran CPLEX for 6 hours with an optimality tolerance of 1% and other parameters left at their default values. This means CPLEX was run with MIPEmphasis set to *balanced*, which means that CPLEX "uses tactics intended to find

a proven optimal solution quickly" (ILOG (2008)). Table 7 shows the results for the 31 instances considered. We give

- the value of the best primal solution found by CPLEX in 6 hours (CPLEX Primal), the time taken by CPLEX to find that solution (CPLEX Time to Best), the optimality gap (CPLEX Opt. Gap) calculated with the dual bound produced by CPLEX, and the time CPLEX needed to find a solution better than that found by pBPGS$^+$, if able, (CPLEX Time to Beat pBPGS$^+$),

- the value of the best primal solution found by pBPGS$^+$ in 30 minutes (pBPGS$^+$ Primal), the time taken by pBPGS$^+$ to find that solution (pBPGS$^+$ Time to Best), the optimality gap (pBPGS$^+$ Opt. Gap) calculated with the dual bound produced by pBPGS$^+$, the optimality gap (pBPGS$^+$ Opt. Gap CPLEX) calculated with the dual bound produced by CPLEX, the time pBPGS$^+$ needed to produce a primal solution within 5% of the dual bound produced by CPLEX (pBPGS$^+$ Time to 5% Opt.), and the time pBPGS$^+$ needed to find a solution better than that found by CPLEX, if able (pBPGS$^+$ Time to Beat CPLEX),

Table 7: Primal and Dual Comparison with default CPLEX

	CPLEX				pBPGS$^+$					
Instance	Primal	Time to Best	Opt. Gap	Time to Beat pBPGS$^+$	Primal	Time to Best	Opt. Gap	Opt. Gap CPLEX	Time to 5% Opt.	Time to Beat CPLEX
20-230-40-V-L	426,356	1	0.59		423,933	4	0.94	0.02	1	
20-230-40-V-T	399,148	1	0.64		398,870	2	0.86	0.57	2	
20-230-40-F-T	669,587	1	0.14		668,699	7	0.88	0.01	4	
20-230-200-V-L	95,273	20,082	1.99		94,843	1,355	3.52	1.55	127	912
20-230-200-F-L	138,287	15,247	2.52	15,247	138,476	956	6.05	2.65	55	
20-230-200-V-T	99,074	19,599	1.72		98,947	444	4.98	1.59	77	435
20-230-200-F-T	139,898	20,718	4.65		139,889	1,178	6.38	4.64	382	1,178
20-300-40-V-L	433,320	1	0.77		430,314	17	0.58	0.08	3	
20-300-40-F-L	599,917	1	0.81		597,059	29	2.19	0.33	12	
20-300-40-V-T	502,512	1	0.42		501,889	89	2.35	0.29	4	
20-300-40-F-T	648,314	1	0.96		643,395	78	0.86	0.21	50	
20-230-200-V-L	76,392	12,915	2.95		76,333	1,014	4.12	2.87	233	749
20-300-200-F-L	119,304	21,144	5.81		118,204	536	6.94	4.93	507	277
20-300-200-V-T	76,486	18,401	1.65	1,040	76,986	650	4.57	2.29	51	
20-300-200-F-T	110,417	20,386	4.80		109,776	483	6.15	4.25	194	194
30-520-100-V-L	54,466	62	1.00		54,387	100	3.04	0.86	10	
30-520-100-F-L	96,348	7,766	3.25		96,277	1,272	7.19	3.18	676	1,272
30-520-100-V-T	53,957	33	0.83		53,812	733	3.29	0.56	12	
30-520-100-F-T	99,396	4,952	1.52		99,355	389	5.83	1.48	46	389
30-520-400-V-L	114,902	20,090	2.49		114,867	1,690	2.89	2.46	597	1,690
30-520-400-F-L	152,026	21,288	3.18	5,625	152,837	1,736	9.23	3.70	875	
30-520-400-V-T	116,485	21,107	1.58	16,378	116,730	1,593	1.99	1.78	317	
30-520-400-F-T	156,660	21,459	4.03		155,982	1,787	5.77	3.62	932	1,787
30-700-100-V-L	48,064	11	1.00		47,883	367	1.74	0.62	130	
30-700-100-F-L	60,384	12,488	0.85		60,384	1,717	3.34	0.85	85	
30-700-100-V-T	47,781	22	1.04		47,686	1,436	4.60	0.85	20	995
30-700-100-F-T	56,804	1,330	0.93	1,330	56,809	774	4.74	0.94	96	
30-700-400-V-L	99,087	20,643	2.23		98,850	1,196	2.42	2.00	443	962
30-700-400-F-L	139,839	21,307	6.33		138,376	1,706	5.59	5.34		841
30-700-400-V-T	97,078	18,122	2.73	6,830	97,352	1,521	3.51	3.00	531	
30-700-400-F-T	132,820	20,491	3.61	6,742	133,759	1,544	5.33	4.29	1,544	
Average		10,957.10	2.16	7,598.86		851.71	3.93	1.99	267	899

Before, comparing the performance of pBPGS$^+$ with CPLEX, we want to highlight the fact that in 30 minutes pBPGS$^+$ produced a primal solution, a dual bound, and a provable optimality gap that was only 3.93% on average. In fact, pBPGS$^+$ produced high-quality solutions quickly, needing less than 5 minutes, on average, to produce a solution that is within 5% of optimality for 30 of the 31 instances, and needing less than 15 minutes, on average, to produce a solution that is within 2% of optimality. But we see that CPLEX also performed quite well, producing solutions in 6 hours that are, on average, within 2.16% of optimality. However, CPLEX requires, on average, about 3 hours to get to these solutions. In fact, this average is skewed due to the presence of instances with 40 commodities, which are all solved to within 1 percent of optimality in less than 1 second. If we restrict ourselves to the instances with more than 40 commodities, we see that pBPGS$^+$ produced its best solutions, on average, in about 19 minutes whereas CPLEX produced its best solutions, on average, in about 4 hours.

We next turn our attention to Table 8 to study the strength of our extended formulation and what steps are responsible for the final dual bound produced by pBPGS$^+$. We report

- the value of the linear programming relaxation (LPR) of our extended formulation, including pre-processing, (LP Bound Extended),
- the root node bound produced by pBPGS$^+$ after dynamically adding cover and disaggregate inequalities (Root Node Bound), and a comparison (Root Node Gap) of this bound with the bound given by the LPR of the extended formulation in the form of $100 \times$ (Root Node Bound $-$ LP Bound Extended)/(Root Node Bound),
- the bound produced by pBPGS$^+$ at the end of execution, after branching, (Final Bound), and a comparison (Final Gap) of this bound with the bound produced at the root node in the form of $100 \times$ (Final Bound $-$ Root Node Bound)/(Final Bound),
- the bound produced by CPLEX in 6 hours (CPLEX Bound), and a comparison (CPLEX Gap) of this bound with the bound produced by pBPGS$^+$ in 30 minutes in the form of $100 \times$ (CPLEX Bound $-$ Final Bound)/(CPLEX Bound),
- the number of columns needed to solve the Root Node LP before cuts are dynamically added (# Columns to Solve Root Node LP),

- and the total number of columns found by solving the pricing problem during the execution of the algorithm (Total # Columns Found),

Table 8: Dual Bound Performance over Time

Instance	LP Bound Extended	Root Node Bound	Root Node Gap	Final Bound	Final Gap	CPLEX Bound	CPLEX Gap	# Columns To Solve Root Node LP	Total # Columns Found
20-230-40-V,L	378,623	419,958	11.33	419,958	0.00	423,830	0.91	8	24
20-230-40-V,T	371,210	393,405	9.13	395,453	0.52	396,606	0.29	14	155
20-230-40-F,T	602,226	658,704	12.04	662,786	0.62	668,645	0.88	12	48
20-230-200-V,L	68,490	89,783	23.72	91,505	1.88	93,373	2.00	5	80
20-230-200-F,L	98,350	131,204	25.05	131,375	0.13	134,808	2.55	4	126
20-230-200-V,T	73,975	94,020	21.32	94,020	0.00	97,371	3.44	3	149
20-230-200-F,T	100,802	130,969	23.03	130,969	0.00	133,398	1.82	4	138
20-300-40-V,L	385,846	427,820	10.37	427,820	0.00	429,963	0.50	6	20
20-300-40-F,L	488,775	577,950	15.84	583,963	1.03	595,069	1.87	10	491
20-300-40-V,T	467,525	490,042	8.78	490,042	0.00	500,409	2.07	28	364
20-300-40-F,T	565,653	632,314	13.23	637,857	0.87	642,075	0.66	24	264
20-230-200-V,L	58,235	72,579	19.88	73,189	0.83	74,142	1.29	4	79
20-300-200-F,L	87,294	109,995	21.32	109,995	0.00	112,374	2.12	5	116
20-300-200-V,T	61,061	73,468	17.11	73,468	0.00	75,221	2.33	6	168
20-300-200-F,T	83,544	103,021	18.67	103,021	0.00	105,116	1.99	7	127
30-520-100-V,L	43,562	52,735	16.91	52,735	0.00	53,919	2.20	10	175
30-520-100-F,L	66,922	89,350	25.47	89,350	0.00	93,218	4.15	19	108
30-520-100-V,T	46,938	52,041	11.77	52,041	0.00	53,509	2.74	33	192
30-520-100-F,T	75,912	93,472	18.73	93,472	0.00	97,883	4.51	14	99
30-520-400-V,L	95,370	111,551	14.63	111,551	0.00	112,045	0.44	6	25
30-520-400-F,L	121,010	138,731	12.64	138,731	0.00	147,189	5.75	5	18
30-520-400-V,T	98,961	113,787	13.05	114,410	0.54	114,647	0.21	7	27
30-520-400-F,T	124,674	146,986	14.77	146,986	0.00	150,341	2.23	6	27
30-700-100-V,L	38,500	47,049	17.94	47,049	0.00	47,585	1.13	1	157
30-700-100-F,L	45,708	57,368	21.35	58,369	1.71	59,869	2.51	6	110
30-700-100-V,T	41,213	45,493	10.39	45,493	0.00	47,283	3.79	37	68
30-700-100-F,T	44,375	54,118	18.21	54,118	0.00	56,274	3.83	9	54
30-700-400-V,L	77,904	96,453	19.07	96,453	0.00	96,876	0.44	6	19
30-700-400-F,L	104,913	130,641	19.71	130,641	0.00	130,984	0.26	6	14
30-700-400-V,T	81,487	93,939	13.44	93,939	0.00	94,432	0.52	5	18
30-700-400-F,T	107,666	126,624	15.13	126,624	0.00	128,027	1.10	5	14
Average			16.58		0.26		1.95	10.16	112.06

Although we do not include the results, we note that our extended formulation (without pre-processing) is not, for the most part, stronger than the compact formulation, as it produced a slightly stronger bound for only three instances. Column Root Node Gap indicates that the cuts we dynamically add at the root node did significantly strengthen the bound produced by the LPR of our extended formulation. However, column Final Gap indicates that branching did not significantly strengthen the dual bound. This is likely due to the fact that pBPGS$^+$ rarely processed more than 25-30 nodes in 30 minutes. Next, we see from the CPLEX Gap column that while the dual bound produced by pBPGS$^+$ in 30 minutes is weaker than the one produced by CPLEX in 6 hours, it is only 1.95% worse, on average. Lastly, we note that few columns were needed to solve the LPR at the root node, and that there is a loose correlation between the CPLEX Gap and Total # Columns Found results;

pBPGS$^+$ often produced a strong dual bound relative to CPLEX on the instances where it found few columns during execution. For these instances, we note that at most nodes, because of the values taken by the variables in the solution to the LPR, pBPGS$^+$ branched on the z_r variables instead of on elements of the vector (Ny). This suggests that branching on the z_r variables themselves may be a better branching scheme for pBPGS$^+$.

5.3. Comparison with IP Search

We next compare the primal-side performance of pBPGS$^+$ with the IP-based local search heuristic presented in Hewitt et al. (2010). To do so, we ran that heuristic, which we refer to as IP Search, for 30 minutes on a single processor (it is not designed to be executed on multiple processors in parallel). IP Search was run on the same machine and used the same CPLEX installation for solving linear and integer programs as BPGS. Also, the parameters used to run IP Search are the same as those reported in Hewitt et al. (2010). We report in Table 9

- the value of the best primal solution found by IP Search in 30 minutes (IP Search Primal),
- the value of the best primal solution found by pBPGS$^+$ in 30 minutes (pBPGS$^+$ Primal), a comparison (pBPGS$^+$ Gap) of the quality of primal solutions produced by pBPGS$^+$ and IP Search in the form of $100 \times$ (pBPGS$^+$ Primal $-$ IP Search Primal) /(pBPGS$^+$ Primal).

We see that in all but two instances pBPGS$^+$ was able to find an equivalent or better solution than IP Search. We also note that the time IP Search needed to find its best solution was, on average, nearly identical to the time needed by pBPGS$^+$. We conclude from these experiments that pBPGS$^+$ gives better results than IP Search.

5.4. Degree Bounding Restriction

Having established the effectiveness of applying pBPGS$^+$ to the MCFCNF with a restriction induced by the constraints $y_{ij} \leq r_{ij}$, called the *Variable Fixing Restriction*, we turn our attention to applying pBPGS$^+$ to the MCFCNF with a different restriction. In particular, we next consider a restriction that bounds the out-degree of each node with respect to the arcs for which the fixed charge is paid and refer to this restriction as the *Degree Bounding*

Table 9: Primal Comparison of IP Search and pBPGS$^+$

Instance	IP Search Primal	pBPGS$^+$ Primal	Gap
20-230-40-V-L	423,933	423,933	0.00
20-230-40-V-T	398,870	398,870	0.00
20-230-40-F-T	668,699	668,699	0.00
20-230-200-V-L	95,695	94,843	-0.90
20-230-200-F-L	141,700	138,476	-2.33
20-230-200-V-T	100,884	98,947	-1.96
20-230-200-F-T	141,734	139,889	-1.32
20-300-40-V-L	430,253	430,314	0.01
20-300-40-F-L	597,059	597,059	0.00
20-300-40-V-T	501,766	501,889	0.02
20-300-40-F-T	643,395	643,395	0.00
20-230-200-V-L	76,946	76,333	-0.80
20-300-200-F-L	119,590	118,204	-1.17
20-300-200-V-T	77,055	76,986	-0.09
20-300-200-F-T	110,609	109,776	-0.76
30-520-100-V-L	54,427	54,387	-0.07
30-520-100-F-L	97,199	96,277	-0.96
30-520-100-V-T	53,812	53,812	0.00
30-520-100-F-T	100,391	99,355	-1.04
30-520-400-V-L	116,465	114,867	-1.39
30-520-400-F-L	156,433	152,837	-2.35
30-520-400-V-T	118,193	116,730	-1.25
30-520-400-F-T	161,513	155,982	-3.55
30-700-100-V-L	47,883	47,883	0.00
30-700-100-F-L	61,254	60,384	-1.44
30-700-100-V-T	47,736	47,686	-0.10
30-700-100-F-T	56,931	56,809	-0.21
30-700-400-V-L	100,368	98,850	-1.54
30-700-400-F-L	144,077	138,376	-4.12
30-700-400-V-T	98,040	97,352	-0.71
30-700-400-F-T	135,512	133,759	-1.31
Average			-0.95

Restriction. Specifically, $P_N(r)$ consists of the extra constraints $\sum_{j:(i,j)\in A} y_{ij} \leq r_i \quad \forall i \in V$ and $R = \{r \mid r_i = \sum_{j:(i,j)\in A} y_{ij} \quad \forall i \in V \text{ for some } (x, y) \in S_P\}$.

We compare the primal solutions and dual bounds produced when each restriction is used in Table 10. The Primal, Time to Best, and Final Bound columns are as before, a comparison of the quality of the two solutions, calculated as 100 × (Degree Bounding Primal − Variable Fixing Primal)/(Degree Bounding Primal), is reported in column Primal Gap, and a comparison of the quality of the two dual bounds, calculated as 100×(Degree Bounding Final Bound− Variable Fixing Final Bound)/(Degree Bounding Final Bound) is reported in column Final Bound Gap. We see that using the Variable Fixing restriction leads to a better primal solution for nearly every instance and the difference in quality is greatest when instances have 400 commodities. This is likely due to the different preprocessing implications of the two restrictions. Note because we have $|K| * |A|$ binary x_{ij}^k variables, instances with 400 commodities are rather large. By fixing y_{ij} to 0 for some arcs (i, j), the Variable Fixing restriction also fixes the variables x_{ij}^k to 0 for every commodity k, leaving a smaller problem

to solve. The Degree Bounding restriction, however, does not fix the value of any variables, leaving instances that are the same size as the original problem. However, we see from column Final Bound Gap that for 20 of 31 instances using the Degree Bounding restriction ultimately produces a stronger dual bound. While the Variable Fixing restriction clearly leads to better primal solutions, we see that using the Degree Bounding restriction often produces better dual bounds.

Table 10: Primal and Dual Comparison of Variable Fixing and Degree Bounding Restrictions

	Variable Fixing			Degree Bounding				
Instance	Primal	Time to Best	Final Bound	Primal	Time to Best	Final Bound	Primal Gap	Final Bound Gap
20-230-40-V-L	423,933	4	419,958	424,973	2	422,843	0.24	0.68
20-230-40-V-T	398,870	2	395,453	398,870	1	392,590	0.00	-0.73
20-230-40-F-T	668,699	7	662,786	671,202	3	665,753	0.37	0.45
20-230-200-V-L	94,843	1,355	91,505	95,516	1,352	91,034	0.70	-0.52
20-230-200-F-L	138,476	956	131,375	140,667	1,772	132,057	1.56	0.52
20-230-200-V-T	98,947	444	94,020	99,528	1,736	95,357	0.58	1.40
20-230-200-F-T	139,889	1,178	130,969	140,233	1,790	130,907	0.25	-0.05
20-300-40-V-L	430,314	17	427,820	430,341	13	428,401	0.01	0.14
20-300-40-F-L	597,059	29	583,963	597,170	24	586,656	0.02	0.46
20-300-40-V-T	501,889	89	490,042	501,766	141	497,185	-0.02	1.44
20-300-40-F-T	643,395	78	637,857	643,395	159	636,770	0.00	-0.17
20-230-200-V-L	76,333	1,014	73,189	76,566	1,769	72,699	0.30	-0.67
20-300-200-F-L	118,204	536	109,995	120,743	1,487	110,928	2.10	0.84
20-300-200-V-T	76,986	650	73,468	76,642	1,529	73,765	-0.45	0.40
20-300-200-F-T	109,776	483	103,021	110,257	1,373	103,218	0.44	0.19
30-520-100-V-L	54,387	100	52,735	54,546	831	53,218	0.29	0.91
30-520-100-F-L	96,277	1,272	89,350	97,217	1,031	90,080	0.97	0.81
30-520-100-V-T	53,812	733	52,041	53,816	872	52,787	0.01	1.41
30-520-100-F-T	99,355	389	93,472	99,425	1,654	94,371	0.07	0.95
30-520-400-V-L	114,867	1,690	111,551	116,324	1,293	110,946	1.25	-0.55
30-520-400-F-L	152,837	1,736	138,731	153,543	1,708	146,106	0.46	5.05
30-520-400-V-T	116,730	1,593	114,410	117,338	1,684	113,576	0.52	-0.73
30-520-400-F-T	155,982	1,787	146,986	158,837	1,770	149,302	1.80	1.55
30-700-100-V-L	47,883	367	47,049	47,883	828	47,883	0.00	1.74
30-700-100-F-L	60,384	1,717	58,369	61,015	524	58,354	1.03	-0.03
30-700-100-V-T	47,686	1,436	45,493	47,781	600	46,330	0.20	1.81
30-700-100-F-T	56,809	774	54,118	57,175	1,700	54,449	0.64	0.61
30-700-400-V-L	98,850	1,196	96,453	100,976	1,717	96,295	2.11	-0.16
30-700-400-F-L	138,376	1,706	130,641	142,285	1,370	128,711	2.75	-1.50
30-700-400-V-T	97,352	1,521	93,939	98,828	1,450	93,779	1.49	-0.17
30-700-400-F-T	133,759	1,544	126,624	137,469	1,779	126,937	2.70	0.25
Average		851.71			1,095.55		0.72	0.53

6. Conclusions

We have proposed a novel strategy for solving integer programs that automates and formalizes many of the heuristic ideas encountered in integer programming based local search heuristics. Branch-and-price guided search produces high quality solutions quickly, but is still an exact algorithm. At the heart of the approach is the use of column generation both for creating restrictions of the problem to solve to produce primal solutions and for produc-

ing a dual bound on the value of the optimal solution to the problem. Also, the paradigm of solving restrictions to produce primal solutions leads to very simple and effective methods for creating local search neighborhoods of the current solution that are independent of problem structure. While branch-and-price guided search is not completely generic, applying it to a specific problem requires little customization.

Computational experiments demonstrate that branch-and-price guided search produces high quality solutions quickly for MCFCNF. In particular, branch-and-price guided search often produced solutions in 30 minutes that are better than a state-of-the-art MIP solver could produce in 6 hours and that a heuristic customized for the MCFCNF could produce in 30 minutes. However, neither of these benchmark methods were executed in a manner that took advantage of parallelism.

While the dual bounds produced by branch-and-price guided search are not yet as strong as those produced by the MIP solver, this hopefully can be addressed by enhancing it with additional classes of valid inequalities or preprocessing techniques, both of which can easily be accommodated.

There are several avenues for future research. We have seen that different restrictions can have different advantages (one produces better primal solutions, the other better dual bounds). This suggests it is worth studying how more than one restriction can be used at a time. Should the constraint set $Ny \leq q$ encode multiple restrictions? Or, should we solve different extended formulations based on different restrictions in parallel? Also, it will be interesting to explore other problems. Initial experience with applying branch-and-price guided search to a maritime inventory routing problem is promising.

Acknowledgements

Research supported in part by the Air Force Office of Scientific Research under grants FA9550-07-1-0177 and FA9550-09-1-0061.

References

Archetti, Claudia, M. Grazia Speranza, Martin W. P. Savelsbergh. 2008. An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science* **42** 22–31.

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

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

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

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

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

Fukasawa, R., H. Longo, J. Lysgaard, M.P. Aragão, M. Reis, E. Uchoa, R.F. Werneck. 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming* **106** 491–511.

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

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

Glover, F. 1998. A template for scatter search and path relinking. *Lecture Notes in Computer Science* **1363** 13–54.

Gropp, William. 1999. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. The MIT Press.

Hewitt, M., G.L. Nemhauser, M.W.P. Savelsbergh. 2010. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing* **22** 314–325.

Hooker, JN, G Ottosson. 2003. Logic-based benders decomposition. *Mathematical Programming* **96** 33–60. doi:DOI 10.1007/s10107-003-0375-9.

ILOG. 2008. *ILOG CPLEX User's Manual*. ILOG.

Nemhauser, George L., Lawrence A. Wolsey. 1988. *Integer and Combinatorial Optimization*. Wiley, New York.

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

Schmid, V., K. F. Doerner, R. F. Hartl, M. W. P. Savelsbergh, W. Stoecher. 2008. A hybrid solution approach for ready-mixed concrete delivery. *Transportation Science* **43** 70–85.

Song, Jin-Hwa, Kevin Furman. 2010. A maritime inventory routing problem: Practical approach. *Computers and Operations Research* Published online before print. doi:10.1016/j.cor.2010.10.031.