A Parallel Local Search Framework for the Fixed-Charge Multicommodity Network Flow Problem

Lluís-Miquel Munguía
College of Computing, Georgia Institute of Technology, Atlanta GA 30332, lluis.munguia@gatech.edu

Shabbir Ahmed
Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta GA 30332

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta GA 30332

George L. Nemhauser
Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta GA 30332

Vikas Goel
ExxonMobil Upstream Research Company, Houston, TX 77098

Yufen Shao
ExxonMobil Upstream Research Company, Houston, TX 77098

We present a parallel local search approach for obtaining high quality solutions to the Fixed Charge Multicommodity Network Flow problem (FCMNF). The approach proceeds by improving a given feasible solution by solving restricted instances of the problem where flows of certain commodities are fixed to those in the solution while the other commodities are locally optimized. We derive multiple independent local search neighborhoods from an arc-based mixed integer programming (MIP) formulation of the problem which are explored in parallel. Our parallel implementation takes advantage of the hybrid memory architecture in modern platforms and the effectiveness of MIP solvers in solving small problems instances. Computational experiments on FCMNF instances from the literature demonstrate the competitiveness of our approach against state of the art MIP solvers and other heuristic methods.

1. Introduction

The Fixed-Charge Multicommodity Network Flow (FCMNF) problem is a classic optimization problem arising in numerous applications. Given a directed network and a set of commodities, the objective is to route every commodity from its origin to destination through the network so as to minimize the total cost. The cost associated with an arc is the sum of a fixed cost derived from its use and a variable cost proportional to the flow going through it. The total cost is derived from the sum of all arc costs. Each arc also has a capacity that constrains the total flow passing through it.

The FCMNF problem was proven to be NP-Hard by Magnanti and Wong (1984). In practice, realistic sized instances of the FCMNF problem are extremely difficult to solve to optimality. Consequently a variety of heuristic approaches and integer programming techniques have been developed and proven to be effective means to achieve high quality solutions quickly. In this paper, we introduce a local search heuristic framework for the FCMNF problem that is explicitly designed for both parallel shared-memory systems and distributed-memory systems. Our method finds competitive solutions by exploring a large number of local search neighborhoods concurrently. Given a feasible solution s, the local searches proceed by solving restricted instances of the problem where flows of certain commodities are fixed to those in the solution s while that of the other commodities are optimized. We take advantage of a state-of-the-art Mixed Integer Programming (MIP) solver to drive these local searches.

Recent works have introduced successful heuristic methods for obtaining high quality solutions. Most common heuristics constist of embedding a problem-specific mechanism for improving solutions in the context of a metaheuristic search framework. Ghamlouche et al. (2004) identify cycles in the network as a heuristic strategy for finding alternative flow routes. The same methodology is used in Ghamlouche et al. (2011) in combination with machine learning tecniques in order to improve and guide the local search. Chouman and Crainic (2010) use a similar approach to identify arc-balanced cycles in combination with a tabu search. A different heuristic approach is presented by Yaghini et al. (2012), where the authors define local search neigborhoods based on simplex pivots in the context of a simmulated annealing framework. Other meta-heuristic frameworks for the FCMNF problem have been developed, such as scatter search (Alvarez et al. (2005)) and evolutionary algorithms (Kleeman et al. (2012)).

Heuristic strategies can also be used in the context of an exhaustive search framework. An example can be found in the local branching technique introduced by Fischetti and Lodi (2003). Their method uses linear inequalities to branch on smaller subproblems, which are tackled by a black-box MIP solver. Examples of applications of local branching for the FCMNF problem can be found in Rodríguez-Martín and Salazar-González (2010). The efficacy of the previously cited work resides in the use of heuristics algorithms in combination with exact mixed-integer programming techniques.

The path-based model is a strong formulation that is also used in conjunction with MIP column generation techniques due to the exponential number of variables. Katayama et al. (2009) use the path-based formulation enhanced by the use of strong inequalities in conjunction with an arc capacity scaling approach. In Katayama (2013), the same scheme is improved with the use of local branching ideas to polish the solutions obtained through arc capacity scaling strategies.

Despite providing high quality solutions quickly, heuristic methods cannot provide optimality certificates because of their exclusive focus on the primal side. Hewitt et al. (2010) introduce an algorithm that provides lower bounds on the optimal solution in addition to primal solution improvements. Such improvements are found by solving strategically restricted MIP subproblems while tighter lower bounds are found with mathematical programming approaches. In further work, Hewitt et al. (2013) combine the use of restricted MIPs in the context of a branch-and-price framework that also provides a performance guarantee upon completion. The authors take advantage of parallelism to solve the pricing problems and restrictions.

Parallelizations of large neighborhood search algorithms have been successfully implemented in other applications such as the LNG inventory routing problem (Badrinarayanan et al. (2014)). To our knowledge, parallel computing remains a relatively unexplored field for the FCMNF problem. Crainic and Gendreau (2002) propose an asychronous parallel tabu search where every processor communicates with a centralized solution pool. They introduce and test several communication policies as well as strategies for handling the exchanged information. In Crainic et al. (2006), special emphasis is put on the control of the information diffusion between the different processors. In this case, the authors present a multilevel parallel local search algorithm that employs parallel local searches defined by sets of fixed arcs. Crainic and Toulouse (2010) provide a comprehensive literature review on the application of parallelism in meta-heuristics.

We present experimental results that show the effectiveness of our parallel local search approach. For the instances in the C problem set (Frangioni (2013)), our method identifies primal solutions that are within an average optimality GAP of 0.58% with respect to the best known lower bound in an average time of 152 seconds per instance. We also test our parallel algorithm against the GT problem set (Hewitt et al. (2010)), which contains substantially bigger instances. Our method takes less than 200 seconds on average to obtain

a better solution than the best one found by CPLEX running for 5 hours. We are able to identify considerably better solutions when our method is allowed more time.

The remainder of the paper is structured as follows. Section 2 presents an arc based MIP formulation of the FCMNF problem and some of its characteristics that motivate our heuristic. Sections 3 and 4 provide a detailed description of our local search methodology and its parallel implementation on hybrid-memory parallel architectures, respectively. Section 5 presents computational experiments and results on standard instances from the literature. Finally, Section 6 provides some concluding remarks.

2. Problem description

Our local search approach is based on an arc-based MIP formulation of the FCMNF problem, which is described as follows. Let $G = (V, A)$ be a directed network, where V is the set of vertices and A the set of arcs. Let K be a set of commodities to be routed through G. Each commodity $k \in K$ is specified by a source vertex $s_k \in V$, a destination $t_k \in V$ and a quantity q_k of flow to be routed. Each arc $(i, j) \in A$ has an associated fixed cost f_{ij} that is imposed only when commodities are routed through it. Arcs also have a variable cost c_{ij} that is proportional to the flow traversing it and a maximum flow capacity u_{ij}. The problem consists of finding a routing for every commodity in K such that the arc capacities are respected and the costs are minimized. Let the flow variable x_{ij}^k denote the proportion of commodity $k \in K$ that is routed through the arc $(i, j) \in A$. In addition to the flow variables, we also introduce the binary variables y_{ij}, which reflect whether each arc (i, j) is used. The FCMNF problem can then be formulated as the following MIP:

$$\min_{x,y} \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} q_k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{1}$$

subject to:

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = d_i^k \quad \forall i \in V, \forall k \in K \tag{2}$$

$$\sum_{k \in K} q_k x_{ij}^k \leq u_{ij} y_{ij} \quad \forall (i,j) \in A \tag{3}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i,j) \in A \tag{4}$$

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

Restriction (2) ensures the fraction of commodity entering a vertex must be equal to the exiting flow unless this vertex is a commodity origin or destination, i.e., the conservation of flow. The flow differential for a vertex i and a commodity k is expressed by d_i^k, which is defined as:

$$d_i^k = \begin{cases} 1 & \text{if } i = s_k \\ -1 & \text{if } i = d_k \\ 0 & \text{otherwise.} \end{cases}$$

The coupling constraints (3) guarantee that the flow through each arc does not exceed the arc capacity. The capacity restrictions have a two-fold function, as they also ensure that the fixed cost is imposed when an arc is used. All commodity flow variables relative to the same arc are aggregated in the same constraint. A tighter and stronger LP relaxation can be obtained by introducing a set of $|A| \cdot |K|$ independent constraints:

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

These are redundant with respect to (3). We choose not to include them in our model due to performance issues resulting from their large number.

We consider two problem variations that differ in whether each commodity may be split through multiple paths or not. In the formulation above, the flow variables are continuous, as specified in constraint (5). Alternatively, each variable x_{ij}^k is binary and (5) is replaced by:

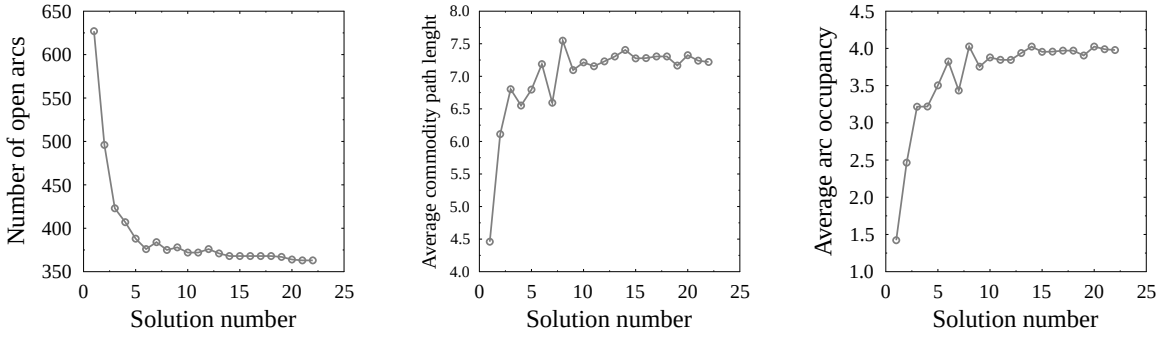
$$x_{ij}^k \in \{0,1\} \quad \forall (i,j) \in A, \forall k \in K. \tag{7}$$

The techniques described in this paper are compatible with both problem variants.

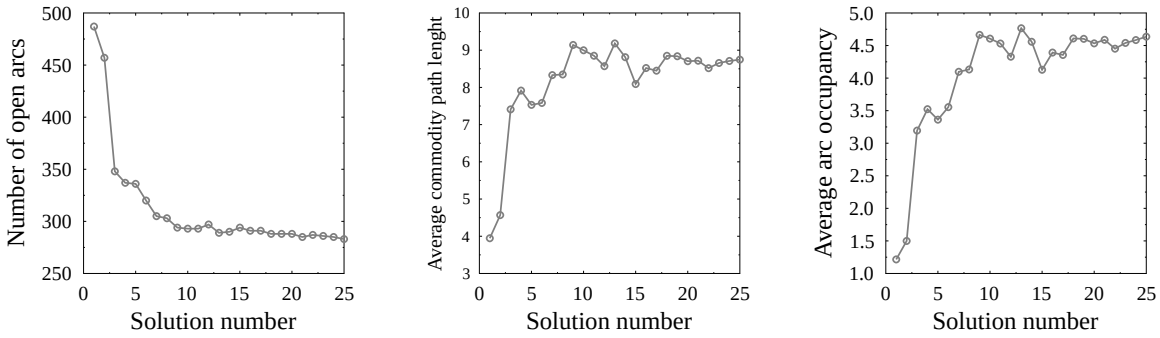
The fixed costs are a major complicating factor in solving the FCMNF problem. When they are omitted we have an LP if the flow can be split and an IP otherwise. We use these simplified models to obtain preliminary primal solutions that satisfy the flow restrictions, although the solutions are far from being optimal.

The impact of the fixed costs can be observed in Figure 1. We present observations gathered from the set of solutions found during the optimization process of two instances. The first instance has 500 vertices, 2500 edges, and 200 commodities while the latter has

500 vertices, 3000 arcs and 150 commodities. We plot the number of arcs used in each of the found feasible solutions. The second metric is the average number of arcs required to route a specific commodity (average commodity path length). We also present the average number of commodities held by a single arc in each solution (arc occupancy). Solutions are arranged such that a higher solution number indicates a lower objective value.



(a) Problem instance 1: Vertices:500, Arcs:2500, Commodities:200



(b) Problem instance 2: Vertices:500, Arcs:3000, Commodities:150

Figure 1 **Solution statistics for two different FCMNF instances. The number of arcs that are used in each feasible solution is shown in the leftmost chart. The center graph depicts the average number of arcs required to route a specific commodity (average commodity path length). The average number of commodities held by a single arc in each solution(arc occupancy) is presented in the right-hand chart. Solutions are arranged such that a higher solution number indicates a better objective value.**

We observe that the most economical arc designs make use of the fewest number of open arcs, resulting in a better utilization. When fewer open arcs are used, commodities must be transported through less direct routes, resulting in longer paths, but a better utilization of arc capacities. These insights motivate arc sharing across commodities to reach high quality solutions.

3. Local search methodology

In the proposed local search scheme, an initial primal solution to an FCMNF instance is improved iteratively by sequentially applying heuristic local searches. In each heuristic local search, solutions are improved by solving a smaller, tractable MIP subproblem that is derived from the original instance. Such reduction in the problem size is obtained by fixing a chosen subset of the variables to the corresponding values of the previously obtained feasible solution. The selection of variables is such that arc sharing is encouraged to reduce costs. New improved solutions then replace the primal incumbent after each iteration, and the scheme is repeated. Algorithm 1 summarizes the procedure.

Algorithm 1 Serial iterative local search scheme

Input: Feasible solution s from a FCMNF instance
Output: Improved feasible solution to the FCMNF instance
  while termination criteria is not met **do**
    $newSol =$ LocalSearch(s)
    if $newSol$ represents improvement over s **then**
      $s = newSol$
    end if
  end while
  Return s

We parallelize Algorithm 1 by decoupling the sequence of local searches to an arbitrary number of independent sequences, each of which can be explored in parallel. To achieve this decomposition, we introduce a local search partitioning mechanism. Thus, each independent subproblem sequence resolution can produce an improved primal solution. The improvements found in the different sequences are then combined into a single feasible solution using a solution recombination scheme. This parallel procedure may be repeated an arbitrary number of times by using the obtained solution as an input for the next iteration. We depict this parallel local search procedure in Figure 2. Next, we describe each component of the overall method in detail.

3.1. The local search mechanism

Given a commodity c and a feasible solution s, we define the set of adjacent commodities $Adj(c)$ as the group of commodities that share flow at least in one arc with c in s, i.e., $Adj(c) = \{c' \in K | \exists (i,j) \in A \text{ s.t. } x^c_{(i,j)} > 0 \text{ and } x^{c'}_{(i,j)} > 0\}$. A local search neighborhood is

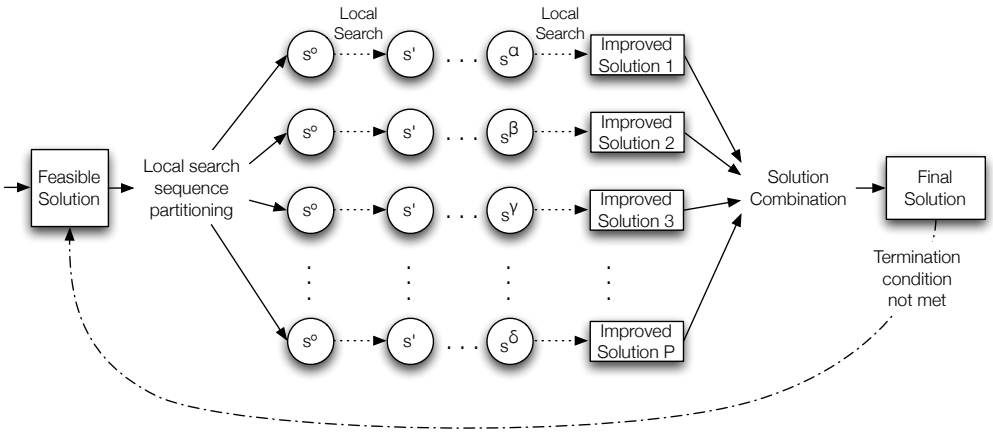


Figure 2 **Parallel decomposition of Algorithm 1. Local searches are distributed in a total of P sequences. As a result, P feasible solutions are obtained and combined in an improved solution.**

then defined in the form of a new MIP subproblem, where the flow of the adjacent commodities and c are free and the remaining flow x is taken from s and fixed. Simultaneously, the arc use variables y are also free for arcs that are not used by the remaining flow. A pseudo-code description of the local search procedure is given in Algorithm 2.

Depending on the selected commodity, the resulting subset of fixed variables may vary in size. Commodities with a large number of adjacent commodities produce difficult MIP subproblems due to the small number of variable fixings. In contrast, commodities with little flow interaction may yield exessively restricted local searches. Disparities in the instance size can be reduced by establishing a threshold on the number of variables that can be fixed in addition to an optimization time limit.

Algorithm 2 Local neighborhood search

Input: Feasible solution s and commodity c from a FCMNF instance
Output: Feasible solution to the FCMNF instance
function LocalSearch(*Solution* s, *Commodity* c)
Compute $Adj(c)$ based on s
for all commodities k not in $Adj(c)$ **do**
Fix the commodity flow variables x_{ij}^k to the values in s, $\forall (i,j) \in A$
end for
solveSubproblem()
return bestSolution()
end function

3.2. Partitioning the subproblem sequence

Given the above local search mechanism and a feasible solution to a FCMNF instance, we can define as many different variable fixings as commodities in the instance. Each derived MIP subproblem is characterized by a specific commodity and its adjacent flow and can be optimized in parallel.

Consider a set of local searches, each of which is identified by a specific commodity. As a first step towards parallelization, we require such local search to be decomposed into a set of disjoint subsets. For this purpose, partitions that yield load-balanced optimizations are highly desirable. As an additional requirement, we put commodities with heavy flow interaction in the same subset. With this specific grouping, we would expect each subset to correspond to a highly interrelated subset of the variables.

The partitioning problem can be transformed into a graph partitioning problem as follows. A new weighted graph $G' = (V', A')$, called the connection graph, is determined from a feasible solution s, the set of arcs A and the set of commodities K. In G', the set K represents the vertices: $V' = K$. $A' = \{(u, v) | u, v \in V' \text{ and } \exists (i, j) \in A \text{ s.t. } x^u_{(i,j)} > 0 \text{ and } x^v_{(i,j)} > 0\}$, i.e., there is an arc (u, v) in $G'(V', A')$ if and only if commodities u and v share flow in an arc in s. In addition, we specify the weight of an edge $(u, v) \in A'$ to be the number of shared arcs.

Thus, the division of commodities between P parallel processors can be translated into a graph partitioning problem of the connection graph, where the cut between the P different subsets is minimized. The connection graph partitioning can be accomplished by specialized graph partitioning algorithms, including the Kernighan-Lin method (Kernighan and Lin (1970)), which are available in graph partitioning libraries such as Metis (Karypis and Kumar (1998)).

3.3. The solution recombination procedure

Consider a set S of improved solutions obtained from a single original feasible solution after a parallel local search. Given the nature of the local search neighborhood, it is likely that the solutions that compose S are highly similar in most parts of the arc design. Solutions can be effectively combined by focusing on the arc variations that have been produced as a product of the local search phase. To do so, a new MIP subproblem is devised, where the y variables corresponding to the arcs that are not used in any solution from the set are fixed to zero. Note that each of the solutions in S is still feasible in this new MIP subproblem

and can, therefore, be used as a starting solution. In Algorithm 3 a pseudo-code description of the solution recombination mechanism is given.

Algorithm 3 Solution recombination algorithm

Input: Feasible solution set S

Output: Feasible solution to the FCMNF instance with an accumulation of the improvements.

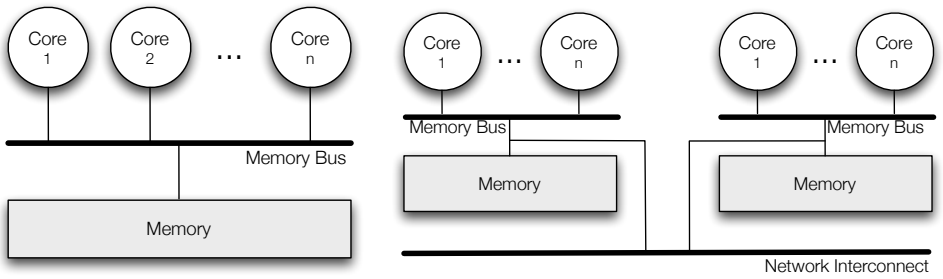
```
function SolutionRecombination(Solution set S)
    for (i, j) ∈ Arcs do
        if (i, j) is not used in any s ∈ S then
            Fix variable y_ij to 0
        end if
    end for
    Add all s ∈ S as starting solutions
    solveSubproblem()
    return bestSolution()
end function
```

Similar to our heuristic local search scheme, the subset of fixed variables in each solution recombination may vary in size. Solutions in the earlier stages of the computation may incorporate many changes in the routing, resulting in a difficult MIP subproblem due to the small number of arc fixings. The opposite effect may be obtained if little improvement is found during the local search phase. We resolve the differences in the problem size by specifying an optimization time limit.

4. Parallel implementation

In this section, we present more details of the algorithm implementation. For parallel scalability, our scheme is designed for hybrid memory systems that combine both distributed-memory systems as well as parallel shared-memory machines. Throughout the paper, we refer to a computing node (or processor) as a set of multiple CPU cores that share a single, unified memory subsystem as shown in Figure 3a *(shared-memory system)*. For scalability, parallel execution may take place across several computing nodes concurrently. In this case, the memory space is segmented and distributed between the individual processors as depicted in Figure 3b *(distributed-memory system)*. As such, a collection of parallel processors constitute a parallel distributed-memory system. Efficient implementations require algorithm design techniques tailored for each memory environment. An important distinction in the implementation is found in the synchronization of parallel components. Parallel

distributed-memory systems usually rely on synchronous message passing techniques such as MPI (Gropp et al. (1999)) to perform communications between processors. In contrast, communication between the parallel cores in a single computing node is much simplified becuase memory is shared.



(a) Shared-memory system (b) Distributed-memory system

Figure 3 **Schematic depiction of parallel systems with different memory configurations. System (a) features a unified memory space, which is accessible by every parallel core simultaneously. In contrast, system (b) has a segmented memory space, which is distributed between the different parallel cores.**

We start describing our parallel scheme by its implementation in a single shared-memory parallel processor as shown in Algorithm 4. Consider a processor with C parallel processor cores and a starting solution s that is initially given to each of them. Each core proceeds to improve its local solution in parallel by resolving a different sequence of MIP subproblems and accumulating the improvements found in the optimization process. Given that the parallel cores share the same memory space, the work partitioning can be arranged dynamically. In order to do so, we employ a shared data structure which holds the subproblems that every core has access to. When a subproblem is solved, it is removed from the shared set. In this fashion, subproblems are distributed dynamically between the parallel cores such that the routing of each commodity is optimized by exactly one of them.

Each variable fixing yields a smaller integer problem as previously described, which is then solved using a black-box MIP solver. As a product of the parallel local search, improvements in the routing are accumulated in at most C different solutions. We employ our solution recombination step to combine them to a single feasible solution, which may be used as input to the next iteration. To ensure full system utilization, we may assume the number of commodities to be greater than the overall number of parallel cores. When P is smaller than K, only K parallel cores are used during the local search phase. The algorithm

Algorithm 4 Parallel local search iteration

Input: Feasible solution s from a FCMNF instance, set of commodities K
Output: Improved feasible solution to the FCMNF instance

```
function ParallelLocalSearch(Solution s, Commodity set K)
    for every thread t_i ∈ C in parallel do
        Initialize Sol_{t_i} as a copy of s
        while there exists commodity k ∈ K do
            Remove k from K
            newSol = LocalSearch(Sol_{t_i}, k)
            if newSol represents improvement over Sol_{t_i} then
                Sol_{t_i} = newSol
            end if
        end while
    end for
    return SolutionRecombination(Sol)
end function
```

is designed such that each parallel core may improve its local solution copy by sequentially solving multiple MIP subproblems and replacing the solution when improvements are found.

We adapt our implementation to a hybrid memory parallel system by augmenting Algorithm 4 in a two-layered scheme, where each level is dedicated to a different level of parallelism. The first layer is responsible for the local search partitioning between the different distributed-memory processors by the use of MPI. The second execution tier takes place in the shared-memory setting of each parallel processor and is responsible for the actual local search exploration. It is in this inner execution level where the MIP subproblems are collectively resolved by the parallel cores in each processor.

Figure 4 describes the interactions between each tier and the workflow of the execution. As a first step, the set of MIP subproblems is partitioned and distributed among the processors. After the partitioning, each computing node proceeds to find improvements in the primal solution by the same procedure as shown in Algorithm 4. The execution returns to the distributed-memory context, where an all-to-all solution exchange communication is performed in order to combine the improvements found by the different processors. In Algorithm 5 we present a pseudo-code description of the distributed-memory execution layer.

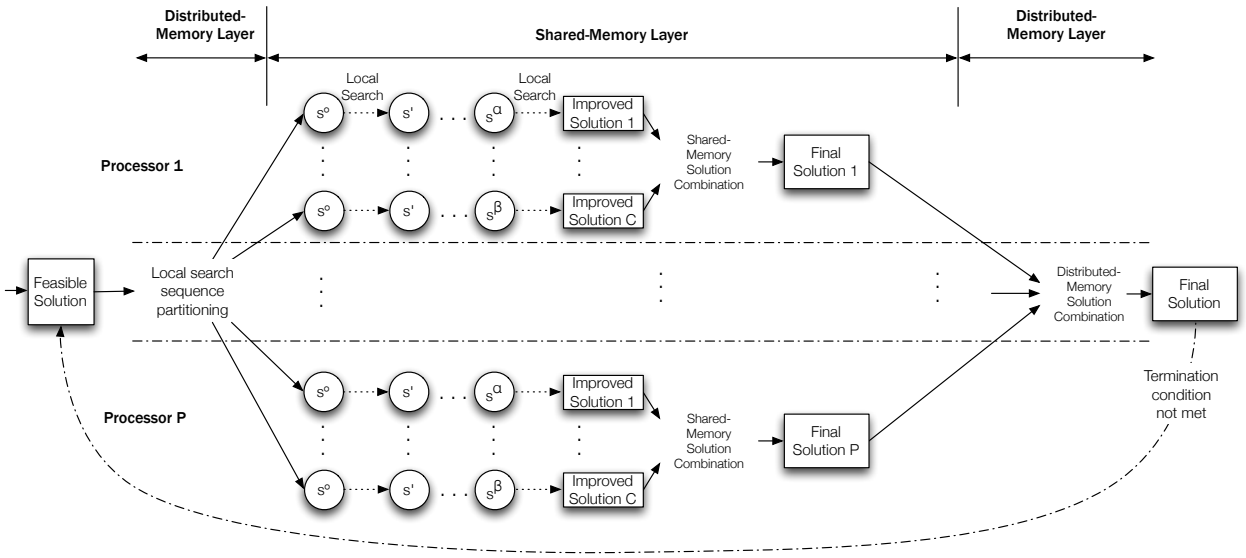


Figure 4 **Parallel hybrid memory framework. The process described in Figure 2 is expanded to accomodate the parallel execution in distributed-memory systems.**

Algorithm 5 Distributed-memory parallel local search framework

Input: Feasible solution s from a FCMNF instance, set of commodities K

Output: Improved feasible solution to the FCMNF instance

function DISTRIBUTEDLOCALSEARCH(*Solution* s)
  while termination criteria is not met **do**
    Let $Part$ be a partition of the commodity set K in P subsets based on s
    for all processors P_i in Parallel **do**
      $Solution_{P_i}$ =PARALLELLOCALSEARCH($s, Part_{P_i}$)
      $AllProcSolutions$ = ALL-TO-ALLEXCHANGE($Solution_{P_i}$)
    end for
    s =SOLUTIONRECOMBINATION($AllProcSolutions$)
  end while
  return s
end function

5. Experimental results

In this section, we study the performance of our parallel local search in terms of solution quality, parallel scalability and load balance. Our framework is implemented in C++ and uses CPLEX 12.4 as the MIP solver. Our tests were performed on an 8-node computing cluster, with each node having two Intel Xeon X5650 6-core processors, 24 GB of RAM memory and a Red Hat Enterprise Linux distribution. Whenever CPLEX was used for comparison purposes, its configuration was set to default. CPLEX is configured with 12 threads, as it can take advantage of parallelism in shared-memory machines. We test the

competitiveness of our parallel method by comparing its performance against previous heuristic approaches presented in the literature. For this purpose, two FCMNF problem instance sets are used, the C instances (Frangioni (2013)) and the GT instances (Hewitt et al. (2010)). In order to assess the scalability of our parallel method, we test the performance under different processor configurations.

The choice of the local search parameters such as the search time limit can have a significant impact on the effectiveness of the method. An optimal combination is highly dependent on the instance size, the number of arcs it has and the number of commodities. To cope with this variety, we test several parameter configurations and report on the most effective combination. In general, we found the optimal balance between local search and solution recombination is to allocate approximately twice the time for recombination as that allocated to each local search. For the C instance set, the limits were set as 10 seconds for each local search and 20 seconds for solution recombination. When solving the GT intances, the local search time limit was set at 20 seconds and 50 seconds was allowed for the recombination process.

5.1. C instance set performance results

The C instance set is comprised of 37 FCMNF medium-sized instances. They have networks with 20 or 30 vertices, a number of arcs ranging from 230 to 700 and a number of commodities ranging from 10 to 400. The configuration of each problem instance is specified with the tuple $\{Nvertices, NArcs, NCommodities, F/V, T/L\}$, where F indicates that the instance's fixed costs are predominant in relation to the variable costs (V otherwise). T characterizes a tigthly capacitated problem instance and L denotes loose arc capacities. In order to evaluate the quality of the solutions obtained with our scheme, we compare it against the results presented in prior publications and default CPLEX on the problem variant where the flow routing can be split between different paths. Specifically, results are compared with those reported in the IP Search scheme ($IPSearch$) by Hewitt et al. (2010), two sets of results obtained with the capacity-scaling combined scheme ($Comb1$ and $Comb2$) of Katayama (2013), and CPLEX with a time limit of 1 hour. We refer to the results obtained with our Parallel Local Search as $ParLS$.

The performance results over the C instance set is shown in Table 1, where the best lower bound for each instance is reported followed by the best primal solution found by each method. The best values are denoted in bold. When a value is optimal, it is marked

with an asterisk. We also specify the time required to reach the best solution. The lower bounds for each problem were either obtained from the literature or found by CPLEX with a 12-hour limit. The reported GAP values are relative to the optimal solution or to the best lower bound. It is computed as $GAP = \frac{P_{sol} - L_B}{P_{sol}} \cdot 100$, where P_{sol} is the solution to each instance and L_B its lower bound.

ParLS finds an optimal solution in 15 out of 37 instances. In comparison to the incumbents reported in the literature, better or equally good solutions are found in 20 cases. When we consider all 37 instances, we obtain solutions that are within an average optimality GAP of 0.58%. The convergence to such solutions is obtained very quickly, averaging 152 seconds per instance. Comparing our results to previous research is a difficult task

Table 1

C instance set optimization results

		Primal solution value					Time to best solution (s)				
Problem	LB / Opt	IPSearch	Comb1	Comb2	CPLEX	ParLS	IPSearch	Comb1	Comb	CPLEX	ParLS
100/400/010/VL	28423*	**28423***	28426	**28423***	**28423***	28486	35	**0**	2	1	3
100/400/010/FL	23949*	**23949***	24459	**23949***	**23949***	24022	9	23	106	112	**1**
100/400/010/FT	63066	65885	68410	64207	**64143**	64207	813	**12**	2736	3600	53
100/400/030/VT	384802*	384836	384809	**384802***	**384802***	**384802***	330	**2**	1503	680	42
100/400/030/FL	49018*	49694	49588	**49018***	**49018***	**49018***	886	167	864	3600	**29**
100/400/030/FT	132129	141365	142191	138152	138587	**136861**	888	**12**	11028	3600	79
20/230/040/VL	423848*	424385	**423848***	**423848***	**423848***	424075	4	**0**	1	1	2
20/230/040/VT	371475*	371779	371906	**371475***	**371475***	371573	41	**1**	3	**1**	**1**
20/230/040/FT	643036*	643187	643649	**643036***	**643036***	**643036***	45	**1**	15	5	10
20/230/200/VL	94213*	95097	94218	**94213***	94218	**94213***	822	**104**	3060	3600	123
20/230/200/FL	137642*	141253	137702	**137642***	137854	**137642***	691	254	4409	3600	**144**
20/230/200/VT	97914*	99410	97968	**97914***	**97914***	**97914***	821	68	2161	606	**52**
20/230/200/FT	135863	140273	136265	136031	136144	**135867**	**156**	263	4538	3600	240
20/300/040/VL	429398*	**429398***	**429398***	**429398***	**429398***	**429398***	19	**0**	1	1	1
20/300/040/FL	586077*	**586077***	587512	**586077***	**586077***	**586077***	29	**1**	8	3	16
20/300/040/VT	464509*	**464509***	**464509***	**464509***	**464509***	**464509***	24	**1**	4	**1**	23
20/300/040/FT	604198*	**604198***	**604198***	**604198***	**604198***	**604198***	68	**1**	4	**1**	3
20/300/200/VL	74753	75319	74840	**74811**	74929	**74811**	802	**101**	5048	3600	361
20/300/200/FL	113862	117543	115801	115748	**115541**	115580	686	350	7769	3600	**250**
20/300/200/VT	74991*	76198	74995	**74991***	**74991***	**74991***	388	65	2158	791	**58**
20/300/200/FT	106672	110344	107315	107315	**107102**	**107102**	396	258	3798	3600	**122**
30/520/100/VL	53958*	54113	53976	**53958***	**53958***	53978	218	**7**	673	721	20
30/520/100/FL	93570	94388	94201	**93967**	**93967**	**93967**	226	150	3266	3600	**83**
30/520/100/VT	52046*	52174	52248	**52046***	**52046***	**52046***	455	**7**	2004	3600	32
30/520/100/FT	96260	98883	97833	97385	**97107**	97862	815	4992	4802	3600	**158**
30/520/400/VL	112735	114042	112787	**112774**	**112774**	112787	394	**95**	5917	3600	542
30/520/400/FL	147790	154218	149486	149423	**149242**	149677	750	1184	3387	3600	**463**
30/520/400/VT	114641*	114922	**114641***	**114641***	**114641***	**114641***	621	**54**	3726	3600	461
30/520/400/FT	150685	154606	152630	**152576**	153005	154137	466	358	5149	3600	**288**
30/700/100/VL	47603*	47612	**47603***	**47603***	**47603***	**47603***	32	**4**	64	18	179
30/700/100/FL	59958*	60700	60067	**59958***	60066	60058	741	228	2888	3600	**111**
30/700/100/VT	45872*	46046	46070	**45872***	**45872***	45879	371	**9**	5559	3600	258
30/700/100/FT	54904*	55609	55164	**54904***	**54904***	**54904***	387	**26**	4456	3600	173
30/700/400/VL	97189	98718	97901	**97875**	97914	98090	**222**	466	4727	3600	243
30/700/400/FL	131690	152576	134723	**134620**	135892	136257	860	2115	7312	3600	**223**
30/700/400/VT	94508	96168	95267	**95250**	95293	95651	**365**	1570	6376	3600	374
30/700/400/FT	128243	131629	129910	129910	130140	131104	225	3955	8165	3600	428
Average GAP and time		1.73	0.89	0.47	0.51	0.58	408	456	3180	2382	152

due to the diversity in the experimental conditions and the differences in the hardware and sofware. However, our parallel method identifies quality solutions that are competitive with the ones reported in prior work and requires much less time to achieve them. The incumbents reported in $Comb2$ and $CPLEX$ are results that represent an improvement over those obtained with our approach. But the solution times are larger by a factor of 20 and 15 respectively.

5.2. GT instances

The GT set is comprised of 24 FCMNF instances, which range in the number of arcs from 2000 to 3000 and have 50 to 200 commodities. When an arc-based formulation is considered, the instance sizes range between 102000 and 603000 variables. They are additionally presented in two versions, whether the problem instances are tightly capacitated (F_T) or loosely capacitated (F_L). We compare the performance of our parallel local search (reported as $ParLS$) with the IP Search scheme from Hewitt et al. (2010) ($IPSearch$), both with a time limit of one hour. We also compare the best results obtained by CPLEX with a time limit of 5 hours ($CPLEX5H$). We tested several CPLEX emphasis configurations and found that a balanced configuration between optimality and feasibility was the best performer. CPLEX is also used to determine the lower bound on every instance. Results are detailed in Table 2, where the best found primal solutions are given, as well as the time required by the parallel local search to improve the best solution found by the other two methods. The reported GAP values are calculated with respect to the best found lower bound.

Results demonstrate the considerable difficulty in solving the GT instance set, as CPLEX is only able to achieve an average optimality GAP of 24.09% after 5 hours of execution. A big part of the challenge resides in the complexity of obtaining tight lower bounds due to the weakness of the arc-based formulation. In comparison to CPLEX and IP Search, our parallel local search scheme finds better solutions for every instance. On average, it requires less than 200 seconds to improve the best solution found by CPLEX running for 5 hours. Improvements become more noticeable in the instances with more commodities because these instances benefit more from parallelism and are more challenging for CPLEX.

5.3. Scaling results

One of the primary goals of our approach is to exploit a large degree of parallelism. We rely on the concurrent exploration of a high number of local searches to find competitive

Table 2

GT instance set optimization results

		Primal solution value			Optimality GAP			Time to improve solution (s)	
Problem	Lower Bound	CPLEX5H	IP Search	ParLS	CPLEX5H	IP Search	ParLS	CPLEX5H	IPSearch
F_T,500,2000,50	4326550	5038580	4949780	**4892012**	14.13	12.59	**11.56**	114	178
F_T,500,2000,100	6368730	7592260	7619670	**7273916**	16.12	16.42	**12.44**	78	75
F_T,500,2000,150	7208800	8640390	8807650	**8014986**	16.57	18.15	**10.06**	317	234
F_T,500,2000,200	8845440	11858000	11893100	**10617796**	25.41	25.63	**16.69**	257	257
F_T,500,2500,50	3927990	4585510	4600200	**4406080**	14.34	14.61	**10.85**	72	72
F_T,500,2500,100	5330490	6942260	6953660	**6365848**	23.22	23.34	**16.26**	134	134
F_T,500,2500,150	5930530	8094410	7571640	**7037860**	26.73	21.67	**15.73**	216	488
F_T,500,2500,200	8327720	11963100	11452900	**10727261**	30.39	27.29	**22.37**	312	396
F_T,500,3000,50	3529370	4333310	4262350	**4035362**	18.55	17.2	**12.54**	99	166
F_T,500,3000,100	5442880	7164410	7186810	**6634387**	24.03	24.27	**17.96**	229	214
F_T,500,3000,150	6236240	8773910	8709390	**7517445**	28.92	28.4	**17.04**	150	155
F_T,500,3000,200	7283080	11236600	10390700	**9751002**	35.18	29.91	**25.31**	308	510
F_L,500,2000,50	3432140	3882110	3823610	**3722839**	11.59	10.24	**7.809**	136	196
F_L,500,2000,100	5497770	6706100	6453880	**6005177**	18.02	14.81	**8.449**	146	271
F_L,500,2000,150	6750150	8205000	8081600	**7510651**	17.73	16.48	**10.13**	198	211
F_L,500,2000,200	8031600	10181700	9828350	**9338097**	21.12	18.28	**13.99**	424	592
F_L,500,2500,50	3176040	3818440	3612030	**3491664**	16.82	12.07	**9.039**	90	213
F_L,500,2500,100	5062110	6893490	6400140	**5909401**	26.57	20.91	**14.34**	133	303
F_L,500,2500,150	6542600	10022900	9089920	**8138918**	34.72	28.02	**19.61**	155	319
F_L,500,2500,200	7717740	11937300	10099200	**9788913**	35.35	23.58	**21.16**	384	1976
F_L,500,3000,50	2958630	3668660	3457280	**3369303**	19.35	14.42	**12.19**	110	254
F_L,500,3000,100	4855420	6692780	6015950	**5773133**	27.45	19.29	**15.9**	178	613
F_L,500,3000,150	6031650	9378030	8919720	**7741294**	35.68	32.38	**22.08**	223	254
F_L,500,3000,200	6722660	11240900	10040000	**9195115**	40.19	33.04	**26.89**	264	691
			Average value		24.09	20.96	15.43	196	365

solutions faster. The next set of experiments is aimed at showing the effectiveness and benefits of the application of parallelism.

In Figure 5, scaling results are shown for a representative set of instances. Since our approach parallalelizes over the set of commodities, we test a variety of instances with a number of commodities ranging from 50 to 200. For each problem, we report performance results for several processor configurations, ranging from executions on one processor (12 parallel cores) to eight (96 parallel cores). The same results are specified in terms of time in Table 3, where we show the time required to reach different GAP values with respect to the best solution found for each instance as well as with respect to the lower bound.

Overall, parallelism is beneficial and substantially better solutions are achieved by using a larger number of processors. However, little improvement is observed when the number of processors exceeds or equals the commodity count. This is the case for the instances with 50 and 100 commodities, which are comparatively easier than instances with similarly sized networks and a higher commodity count. The impact of parallelism differs from instance to instance due to their variability and the heuristic nature of our approach, including the eventuality that not every local search may yield improvements at every iteration.

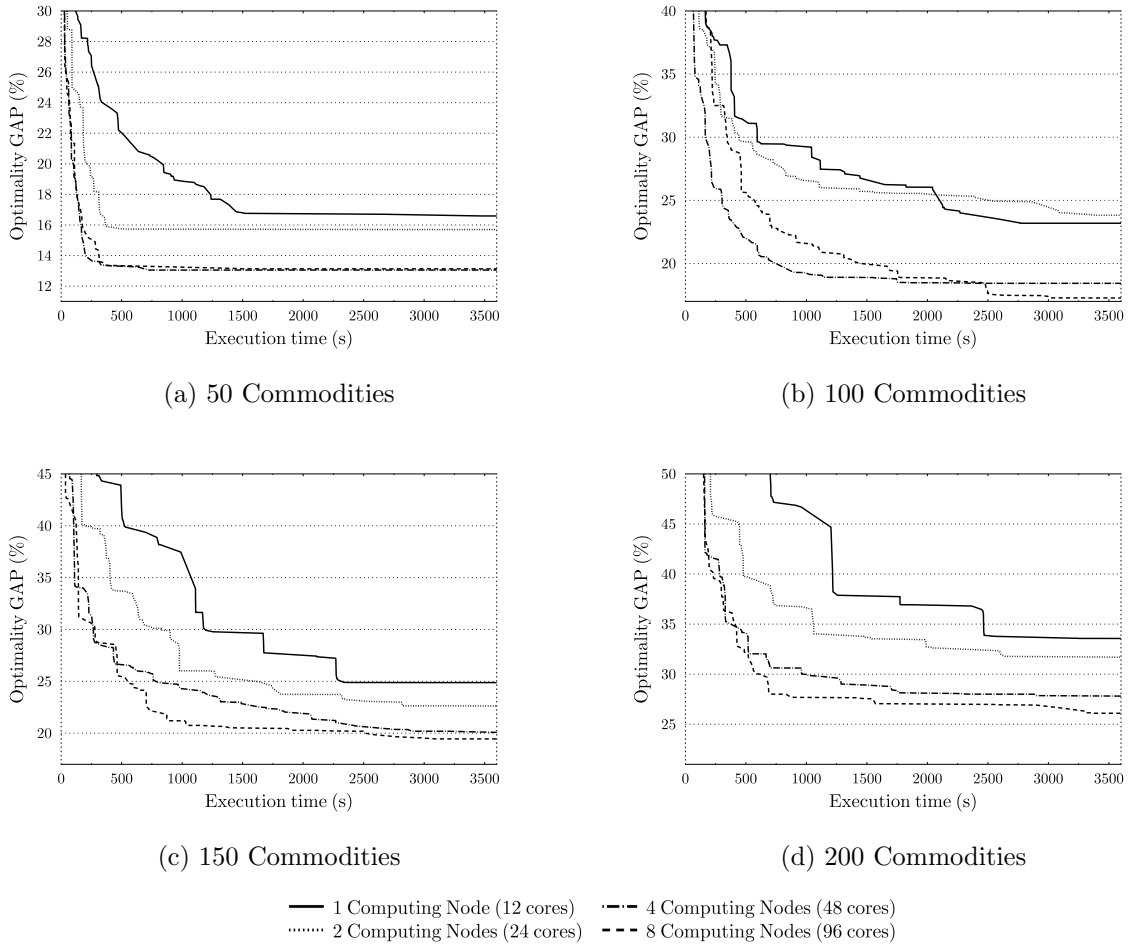


Figure 5 **Scaling results for a selected test instance with 500 vertices, 3000 arcs and varying commodities. Each plot depicts the improvement in optimality GAP with respect to the best lower bound as a function of time. The executions shown in each problem differ in the number of parallel cores used.**

Instances with more commodities show better scalability, as more parallelism is exploited and there exists more opportunities for solution improvements.

5.4. Load balancing

In addition to scalability, load balancing is another relevant aspect of parallel efficiency. Load balancing refers to the uniform distribution of work between parallel processors. High processor utilization is essential in order to maximize the throughput in parallel computations. In the case of our algorithm achieving an even load balance depends on how the work is partitioned and the time limit parameters that decide the granularity of the local searches.

Table 3

Time required to reach certain gap with respect to the best found primal solution and the best lower bound

Problem	Number of computing nodes	Time required to reach GAP with respect to best solution (s)					Time required to reach GAP with respect to best lower bound (s)				
		20%	10%	5%	1%	0%	35%	30%	25%	20%	15%
F_T,500,3000,50	1 (12 cores)	137	638	1388	–	–	18	137	315	845	–
F_T,500,3000,50	2 (24 cores)	48	184	317	–	–	10	51	146	240	–
F_T,500,3000,50	4 (48 cores)	29	85	144	227	725	9	29	62	104	182
F_T,500,3000,50	8 (96 cores)	28	78	161	314	–	3	28	48	111	282
F_T,500,3000,100	1 (12 cores)	378	2124	–	–	–	378	594	2124	–	–
F_T,500,3000,100	2 (24 cores)	285	2069	–	–	–	247	443	2529	–	–
F_T,500,3000,100	4 (48 cores)	126	300	596	–	–	81	165	303	886	–
F_T,500,3000,100	8 (96 cores)	223	517	1057	2490	3506	221	348	577	1667	–
F_T,500,3000,150	1 (12 cores)	1110	2112	–	–	–	1110	1196	2336	–	–
F_T,500,3000,150	2 (24 cores)	406	978	2325	–	–	406	825	1637	–	–
F_T,500,3000,150	4 (48 cores)	111	433	1303	2902	–	111	267	790	–	–
F_T,500,3000,150	8 (96 cores)	143	462	703	2210	3589	143	281	563	2508	–
F_T,500,3000,200	1 (12 cores)	1216	–	–	–	–	2465	–	–	–	–
F_T,500,3000,200	2 (24 cores)	478	1951	–	–	–	1060	–	–	–	–
F_T,500,3000,200	4 (48 cores)	281	518	1128	–	–	455	999	–	–	–
F_T,500,3000,200	8 (96 cores)	197	427	674	3108	3379	423	623	–	–	–

We characterize the total execution time of a specific processor P_i as the sum of the useful computation time TC_{P_i}, the communication time TX_{P_i} and the idle time TI_{P_i}. We may define the communication time as the time spent performing communication between processors, whereas the idle time TI_{P_i} accounts for the idle time spent by a processor on synchronization or waiting for other processors to finish their computations. Then, we define the utilitzation of a processor P_i as the ratio of useful computation time over the total execution time:

$$U(P_i) = \frac{TC_{P_i}}{TC_{P_i} + TX_{P_i} + TI_{P_i}}$$

Figure 6 displays the average core utilization for a representative FCMNF instance under different processor configurations and different time limit parameters. Each parameter configuration is specified with two time limits, where the first number corresponds to the local search time limit while the second refers to the solution recombination time limit. We show that average processor utilizations remain very high through all the tested combinations. When a single computing node (12 cores) is used, the shared-memory dynamic work allocation mechanism proves to be an effective means of load balance, as we achieve an utilization as high as 98%. The utilization also decreases with higher processor counts due to the requirement of more static partitions. Allowing more time for each local search also

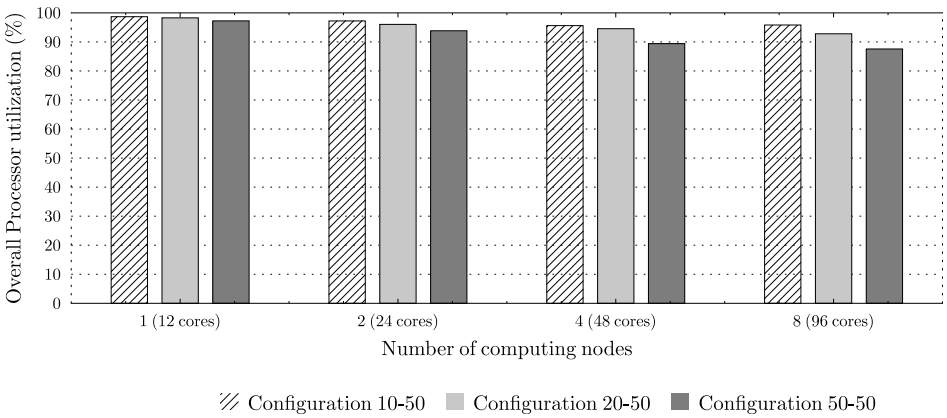


Figure 6 **Overall processor utilization results for a selected test instance with 500 vertices, 3000 arcs and 200 commodities. Each data set corresponds to a different time limit parameter configuration, where the first number refers to the local search time limit and the second refers to the solution recombination time limit.**

has a detrimental effect on processor utilization. This is due to the fact that the processors that end their work prematurely will have increased idle time penalties.

6. Conclusions

We propose a scalable parallel approach for the Fixed Charge Multicommodity Network Flow problem that is designed for both shared memory parallel systems and distributed memory systems. By the use of heuristic local searches based on solving restricted MIP subproblems obtained by variable fixings, improvements in the flow routing are found in parallel and are further combined to obtain improved solutions. We rely on the network characteristics of the instances and the given solutions to define core components of the algorithm, such as the work partitioning and the solution recombination mechanism. Computational experiments demonstrate the effectiveness and scalability of our approach, as high-quality solutions are obtained for two problems sets from the literature.

Large sized FCMNF problem instances represent a computational challenge. Commercially available solvers and previous heuristic methods struggle to provide solutions and lower bounds that are within a reasonable optimality gap. It is precisely in the size of these instances where many opportunities to exploit parallelism can be found. We demonstrate the value of parallel computing and heuristic approaches for effectively generating good primal solutions to large FCMNF problem instances. Optimality certificates in the form

of lower bounds are still difficult to achieve. In future research, we will seek to solve this shortcoming by applying the notions presented in this paper in the context of an exact algorithm that combines primal and dual aspects.

Acknowledgments

We would like to thank Rodolfo Carvajal for helpful discussions. This research has been supported in part by ExxonMobil Upstream Research Company and the Air Force Office of Scientific Research.

References

Alvarez, A. M., J. L. González-Velarde, K. De-Alba. 2005. Scatter search for network design problem. *Annals of Operations Research* **138** 159–178.

Badrinarayanan, V. A., K. C. Furman, V. Goel, Y. Shao, G. Li. 2014. Parallel large-neighborhood search techniques for lng inventory routing. *Submited to Informs Journal on Computing* URL `http://www.optimization-online.org/DB_HTML/2014/04/4319.html`.

Chouman, M., T. Crainic. 2010. *A MIP-tabu search hybrid framework for multicommodity capacitated fixed-charge network design*. CIRRELT.

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

Crainic, T. G., Y. Li, M. Toulouse. 2006. A first multilevel cooperative algorithm for capacitated multicommodity network design. *Comput. Oper. Res.* **33** 2602–2622.

Crainic, T. G., M. Toulouse. 2010. Parallel meta-heuristics. *Handbook of Metaheuristics*, *International Series in Operations Research & Management Science*, vol. 146. Springer US, 497–541. doi:10.1007/978-1-4419-1665-5_17.

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

Frangioni, A. 2013. Multicommodity problems. URL `http://www.di.unipi.it/optimize/Data/MMCF.html`.

Ghamlouche, I., T. G. Crainic, M. Gendreau. 2004. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations research* **131** 109–133.

Ghamlouche, I., T. G. Crainic, M. Gendreau, I. Sbeity. 2011. Learning mechanisms and local search heuristics for the fixed charge capacitated multicommodity network design. *International Journal of Computer Science* **8**.

Gropp, W., E. Lusk, A. Skjellum. 1999. *Using MPI: portable parallel programming with the message-passing interface*, vol. 1. 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.

Hewitt, M., G. L. Nemhauser, M. W. P. Savelsbergh. 2013. Branch-and-price guided search for integer programs with an application to the multicommodity fixed-charge network flow problem. *INFORMS Journal on Computing* **25** 302–316.

Karypis, G., V. Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* **20** 359–392.

Katayama, N. 2013. A combined capacity scaling and local branching approach to capacitated multi-commodity network design problem. Working paper.

Katayama, N., M. Z. Chen, M. Kubo. 2009. A capacity scaling heuristic for the multicommodity capacitated network design problem. *Journal of Computational and Applied Mathematics* **232** 90–101.

Kernighan, B. W., S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal* **49** 291–307.

Kleeman, M. P., B. A. Seibert, G. B. Lamont, K. M. Hopkinson, S. R. Graham. 2012. Solving multicommodity capacitated network design problems using multiobjective evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on* **16** 449–471.

Magnanti, T. L., R. T. Wong. 1984. Network design and transportation planning: Models and algorithms. *Transportation Science* **18** 1–55.

Rodríguez-Martín, I., J. J. Salazar-González. 2010. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research* **37** 575–581.

Yaghini, M., M. Momeni, M. Sarmadi. 2012. A simplex-based simulated annealing algorithm for node-arc capacitated multicommodity network design. *Applied Soft Computing* **12** 2997 – 3003.