# Solving the Uncapacitated Fixed-Charge Network Flow with Metaheuristics

*Christophe Duhamel*[*]

[*] Laboratoire LIMOS, FRE 2239 CNRS, Université Blaise Pascal
Clermont-Ferrand, France
Email: christophe.duhamel@isima.fr

## 1 Introduction

The Uncapacitated Fixed-charge Network Flow (UFNF) problem is an instance of the broader class of Network Design (ND) problems. Let $G = (S, A)$ be a digraph where $S$ is the set of nodes and $A$ is the set of arcs. Each arc $(i, j)$ is associated with two costs: a fixed cost $c_{i,j}$ for selecting $(i, j)$ and a variable cost $d_{i,j}$ for routing flow on $(i, j)$. The problem is to find a set of arcs that allow a supply node to send resources to a set of demand nodes, such that the sum of structural ($c_{i,j}$) and operational ($d_{i,j}$) costs is minimized.

Despite its apparent simplicity, this problem is clearly a $\mathcal{NP}$-hard optimization problem. It generalizes both the Steiner Tree Problem in Graphs (STPG) and the transhipment problem. UFNF is interesting since it can be seen as a basic model for several real-life problems arising in telecommunication, distribution and transportation contexts, among others. It is also interesting because it uses some Quality of Service (QoS) criteria – namely the operational cost – in the design of the solution.

Numerous exact algorithms have been published for the UFNF during the last two decades. [2, 3, 10, 9] have developed several branch and bound techniques on a bipartite underlying network. Cruz and al. [4] use branch and bound combined with lagrangean relaxation. Magnanti et al. [7] have used a combination of branch and bound with Bender's cuts and Hochbaum and Segev [5] a heuristic based on lagrangean relaxation, both of them for the UFNF without transhipment nodes. Recently, some polyhedral methods have been proposed. Koch and Martin [6] report very good results for STPG, while Ortega and Wolsey [8] describe a branch and cut technique for the UFNF. Surpringly, no metaheuristics seem to have been applied to UFNF.

In this paper, we propose two metaheuristics to solve the UFNF. The first one, GLIT, is an generic Iterative Improvement Method (IIM) and is presented in section 4. The second one is a classical Genetic Algorithm (GA) and is briefly introduced in section 5. Both are defined to work on a common framework defined in section 3. Computational results are reported in section 6 and some concluding remarks are discussed in section 7.

## 2 Mathematical Formulation

The UFNF can be formulated as an minimum-cost network flow from the supply node $s$ to a set $D$ of demand nodes, with additional constraints on the choice of the arcs. $T$ is the set of transhipment nodes, $d_k$ is the demand of node $k \in D$ and $U = \sum_{k \in D} d_k$ is the total demand.

$$\text{Min} \quad \sum_{(i,j)\in E} c_{i,j}.x_{i,j} + \sum_{(i,j)\in E} d_{i,j}.f_{i,j}$$

s.c.

$$\sum_{(i,k)\in E} f_{i,k} - \sum_{(k,j)\in E} f_{k,j} = d_k \qquad \forall k \in D \qquad (1)$$

$$\sum_{(i,t)\in E} f_{i,t} - \sum_{(t,j)\in E} f_{t,j} = 0 \qquad \forall t \in T \qquad (2)$$

$$f_{i,j} \leqslant U.x_{i,j} \qquad \forall (i,j) \in E \qquad (3)$$

$$x_{i,j} \in \{0,1\} \qquad \forall (i,j) \in E$$

$$f_{i,j} \geqslant 0 \qquad \forall (i,j) \in E$$

The objective function minimizes a combination of both structural and operational costs. Constraints (1) and (2) are classical flow conservation constraints, while constraints (3) ensure coupling between decision variables $x_{i,j}$ and flow variables $f_{i,j}$.

# 3 Defining a Common Framework

One of the main problem when defining an IIM is that it is usually designed to suit the problem's specificities. From a software engineering point of view, it is clearly unattractive since slight modifications in problem's formulation may involve a lot of work to adapt the method.

Our idea is to perform a better separation between the description of the problem and the strategical elements of an IIM such that problem's dependency is reduced. This is done by defining a working space $\mathcal{W}$ and a decoding function $f$ that maps $\mathcal{W}$ to the initial solution's space $\mathcal{S}$. Then, all the logical components of an IIM (definition of a solution, a move, evaluation function, exploration and selection strategies, etc...) are defined according to $\mathcal{W}$ and $f$. Implementing a new IIM needs only to select the right existing elements and, sometimes, to develop few new ones. This approach has several advantages: portability, genericity and ability to reuse components. One drawback may be however a loss of efficiency.

We can also gain strong features associated with $\mathcal{W}$. Three reasons led us to the choice of sequences as working solutions:

1. it is easy to define a decoding function; most of the time we only need to slightly modify a greedy heuristic
2. there exists obvious metrics over sequences' space. This may be useful when implementing some strategies (path relinking, intensification and diversification for instance)
3. since sequences have been sometimes used in GAs, some moves are still defined

The meaning of sequences depends on the decoding function $f$. Concerning the UFNF, it has been chosen to set the sequence as an ordering of nodes and $f$ as a modified Prim algorithm. Given a space $\mathcal{P}$ of parameters, there are three kinds of neighborhood generators:

1. $\mathcal{P} \rightarrow \mathcal{W}$: a new solution is built given a set of parameters. This is the *Rebuild* generator.

2. $\mathcal{W} \times \mathcal{P} \rightarrow \mathcal{W}$: a new solution is built given an existing solution and a set of parameters. The *Swap* generator exchanges two consecutive elements in the sequence while the *Exchange* generator exchanges distant elements. The *RaR* generator removes an element and inserts it back into another location.
3. $\mathcal{W}^2 \times \mathcal{P} \rightarrow \mathcal{W}^2$: two new solutions are built given two existing solutions and a set of parameters. These are typically the well-known crossover generators (*1-point*, *2-points* and *uniform*).

Crossovers often destroy the structure of the sequence and feasibility needs to be restored. This is achieved by noting that missing elements in one sequence are duplicated in the other one. The restoration procedure looks for duplicated elements in each sequence and then performs exchanges.

# 4 GLIT

When analyzing existing IIMs, one can be struck by the fact that they share the same basic principles and differ only on a few points: neighborhood activation and exploration, candidate selection and validation. It may suggest an homogeneous descriptive model for IIMs. This remark has led to the development of a formal model based on the key-idea of neighborhood. Its exploration is controlled by two kinds of rules: pruning rules perform *a priori* cuts in the neighborhood in order to reduce its size, then filtering rules only keep interesting elements, according to their evaluation. The candidate move is then chosen among the remaining elements.

Some features have also been investigated, such as a classification of neighborhood. The *range* criterion has been introduced. It is a measure of the mean modification level induced by moves in the neighborhood. It allows us to set three scopes: short, medium and long range. The range should be interesting when defining exploration strategies (the higher the range, the wider the search).

GLIT is a possible instantiation of our formal model. It is not *stricto sensu* a new metaheuristic since it does not rely on a new exploration strategy. It is rather a framework which provides all the needed functional mechanisms. They are general enough to be able to simulate the behaviour of most of existing IIMs (simulated annealing, tabu search, GRASP, VNS among others). Therefore, GLIT relies on two structures:

1. the neighborhood generators list $\mathcal{G}$ stores all the available neighborhood generators, with respect to their range. Each operator is associated with an activation probability. At each iteration, one of them is chosen, according to both stochastic and deterministic criteria.
2. the preselection list $\mathcal{L}$ stores all the successfully evaluated moves during a neighborhood exploration. The final candidate is chosen among them. Three parameters help defining the behaviour of $\mathcal{L}$: the insertion length $l_1$, the blocking flag $f$ and the selection length $l_2$. The first one tells how many elements may be stored. The flag tells if exploration should stop when the list is full. The third one is the number of best elements on which a random selection is performed. $\mathcal{L}$ will also be used as a volatile memory for the evaluation of the quality of the neighborhood.

The third major feature of GLIT is its ability to adjust itself using adaptative mechanisms. This is done by modifying activation probabilities according to the quality of both the candidate and the neighborhood. History of the previous current solutions helps us to adjust the settings of $\mathcal{L}$.

# 5 Genetic Algorithms

Genetic Algorithms (GAs) are probably one of the oldest IIM. It is a population-based metaheuristic, based on a metaphor with natural process of evolution. Each element in the population is characterized

by its genetic material and its quality is measured by a fitness function that maps the elements' space to the initial solutions' space.

At each generation, new elements are created using three different operators:

1. *recombination*: genetic material is exchanged between two existing elements to produce two offsprings (crossovers 1-point, 2-points and uniform)
2. *mutation*: one element is modified to produce one offspring (Swap, Exchange and RaR)
3. *immigration*: a brand new element is created (Rebuild)

Elements of the new generation are chosen among the existing elements and the offsprings, according to a selection criterion. Rate of recombination/mutation/immigration as well as choice of the elements to be used also play a key-role in the efficiency of GA.

We have chosen an elitist management of the population coupled with a rank-based selection of the parent elements (Stochastic Universal Sampling [1]). Population's size has been set to $m$ and number of generations to $30 + m$, where $m$ is the number of nodes of the problem.

We have also tried an hybridation between GA and GLIT. GLIT is used as a mutation operator to improve elements of the population. Therefore, it is only allowed to run for a limited number of iterations (50) and with only the Swap, Exchange and RaR generators.

# 6 Computational Results

We present some numerical results obtained with the application of GLIT and GAs on the UFNF. The algorithms were implemented in C and compiled with `AIX` version `2.95.2` of `gcc`. All experiments have been performed on a `IBM 166 MHz RS-6000` workstation with 1Gbytes of memory.

Since there is no benchmark problems for the UFNF, we have generated four tests sets, according to four differents choices of underlying graph topology: 4-connected, 8-connected, random triangulated and random complete graphs. The number of nodes varies from 10 to 400 nodes. Ratio $c_{i,j}/d_{i,j}$ has been set to 10.0. Some computational results are presented in table 1 for the last two underlying topologies. For each problem, $z^*_{\text{IP}}$ is the optimal value, when available. For each method, columns $\underline{z}$, $\sigma$ and CPU denote respectively the best value found, the standard deviation and the average CPU time (in seconds). Bold values show that an optimal solution has been found and $*.*$ shows that the CPU limit time (30 minutes) has been exceeded.

We can see that our three methods are quite effective since they are able to find an optimal solution whenever the optimal value is known. However, for the largest instances, CPU times become quickly prohibitive, especially for the GAs. The main reason is that each evaluation rely on a call to Prim algorithm. One surprise comes from the relative inefficiency of our hybridation. It does not produce better results and is less stable than pure GA. On the other hand, GLIT performs very well: it is fast (comparing to our GAs), efficient and stable.

# 7 Concluding Remarks

We have developed two metaheuristics and an hybridation to solve the UFNF. Both are working on a common framework to improve portability and genericity. Computational results show that our approach is quite effective, even if working on a different solutions' space may be time-consuming for large instances. However, we still have an effective method for fast prototyping. A more careful

| Problem | $z^*_{IP}$ | GLIT | | | GA | | | COUPLING | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $\underline{z}$ | $\sigma$ | CPU | $\underline{z}$ | $\sigma$ | CPU | $\underline{z}$ | $\sigma$ | CPU |
| A_T_10 | 15366 | **15366** | 0.00 | 0.02 | **15366** | 0.00 | 0.36 | **15366** | 0.00 | 0.39 |
| A_T_20 | 15445 | **15445** | 0.00 | 0.13 | **15445** | 0.00 | 2.01 | **15445** | 0.00 | 2.18 |
| A_T_30 | 22686 | **22686** | 0.00 | 0.58 | **22686** | 0.00 | 7.36 | **22686** | 0.00 | 6.96 |
| A_T_40 | 37490 | **37490** | 10.86 | 3.15 | **37490** | 24.54 | 22.14 | **37490** | 43.51 | 18.25 |
| A_T_50 | 40438 | **40438** | 24.15 | 5.23 | **40438** | 33.82 | 36.63 | **40438** | 52.49 | 41.06 |
| A_T_75 | 52603 | **52603** | 168.38 | 24.78 | **52603** | 625.79 | 157.21 | 52617 | 314.20 | 140.49 |
| A_T_100 | 65545 | 65581 | 336.71 | 53.71 | 65581 | 511.73 | 476.86 | 65581 | 680.71 | 434.12 |
| A_T_150 | - | 92910 | 724.05 | 315.77 | | | *.* | | | *.* |
| A_T_200 | - | 116129 | 728.24 | 439.79 | | | *.* | | | *.* |
| A_C_10 | 14135 | **14135** | 0.00 | 0.01 | **14135** | 0.00 | 0.30 | **14135** | 0.00 | 0.30 |
| A_C_20 | 14858 | **14858** | 0.00 | 0.07 | **14858** | 0.00 | 1.47 | **14858** | 0.00 | 1.32 |
| A_C_30 | 21506 | **21506** | 0.00 | 0.35 | **21506** | 0.00 | 4.04 | **21506** | 0.00 | 5.68 |
| A_C_40 | 35110 | **35110** | 195.05 | 2.94 | **35110** | 336.89 | 17.39 | **35110** | 313.26 | 17.17 |
| A_C_50 | 37295 | **37295** | 301.38 | 12.92 | **37295** | 412.76 | 91.08 | **37295** | 246.69 | 81.51 |
| A_C_75 | - | 48653 | 546.99 | 72.77 | 48653 | 603.21 | 540.34 | 48772 | 725.20 | 561.57 |
| A_C_100 | - | 61444 | 399.45 | 223.98 | 61654 | 470.31 | 1790.74 | 61668 | 490.46 | 1750.37 |
| A_C_150 | - | 82476 | 409.88 | 1376.20 | | | *.* | | | *.* |
| A_C_200 | - | | | *.* | | | *.* | | | *.* |

Table 1: computational results for random triangulated (A_T) and complete (A_C) graphs

analysis of GLIT's behaviour should lead to further improvements in stability and also in speed. A wider comparison with classical IIMs (simulated annealing, tabu search, GRASP and VNS) is also under experiment, along with improvement of the coupling between AG and GLIT. Finally, UFNF is only a starting model and some model extensions will include facility location and capacity assignment.

# References


[1] J.E. Baker. *Reducing bias and inefficiency in the selection algorithm. Proc. 2-nd Int. Conf. Gen. Alg., Cambridge, MA*, 14–21, 1987.

[2] R.S. Barr, F. Glover, D. Klingman. *A new optimization method for large scale fixed charge transportation problems. Oper. Res.*, 29, 3:448–463, 1981.

[3] A.V. Cabot, S.S. Erenguc. *Some branch and bound procedures for fixed-cost transportation problems. Naval Res. Log. Quart.*, 31:145–154, 1984.

[4] F.R.B. Cruz, J. MacGregor Smith, G.R. Mateus. *Solving to optimality the uncapacitated fixed-charge network flow problem. Computers Oper. Res.*, 25, 1:67–81, 1998.

[5] D.S. Hochbaum, A. Seguev. *Analysis of a flow problem with fixed charges. Networks*, 19:199–210, 1989.

[6] T. Koch, A. Martin. *Solving Steiner tree problems in graphs to optimality. Networks*, 32, 3:207–232, 1998.

[7] T.L. Magnanti, L. Mireault, R.T. Wong. *Tailoring Benders decomposition for uncapacitated network design. Math. Program. Study*, 26:112–154, 1986.

[8] F. Ortega, L. Wolsey. *A branch-and-cut algorithm for the single commodity uncapacitated charge network flow problem.* working paper, 2000.

[9] U.S. Palekar, M.K. Karwan, S. Zionts. *A branch and bound method for the fixed-charge transportation problems. Management Sci.*, 36, 9:1092–1105, 1990.

[10] J.E. Schaeffer. *Use of penalties in the branch and bound procedure for the fixed charge transportation problems. European J. Oper. Res.*, 43:305–312, 1989.