CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Cycle-Based Evolutionary Algorithm for the Fixed-Charge Capacitated Multi-Commodity Network Design Problem

Dimitris C. Paraskevopoulos
Tolga Bektaş
Teodor Gabriel Crainic
Chris N. Potts

February 2013

CIRRELT-2013-08

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Université Laval · UQÀM · HEC Montréal · École Polytechnique Montréal · Université de Montréal

A Cycle-Based Evolutionary Algorithm for the Fixed-Charge Capacitated Multi-Commodity Network Design Problem

Dimitris C. Paraskevopoulos[1], Tolga Bektaş[2], Teodor Gabriel Crainic[3,*], Chris N. Potts[4]

[1] School of Management, University of Bath, Bath, BA2 7AY, United Kingdom

[2] School of Management,Centre for Operational Research, Management Science and Information systems (CORMSIS), University of Southampton, Southampton, SO17 1BJ, United Kingdom

[3] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Management and Technology, Université du Québec à Montréal, P.O. Box 8888, Station Centre-Ville, Montréal, Canada H3C 3P8

[4] School of Mathematics, Centre for Operational Research, Management Science and Information systems (CORMSIS), University of Southampton, Southampton, SO17 1BJ, United Kingdom

Abstract. This paper presents an evolutionary algorithm for solving the fixed-charge multi-commodity network design problem (MCNDP), which concerns routing multiple commodities from origins to destinations by designing a network through selecting arcs, with an objective of minimizing the fixed costs of the selected arcs plus the variable costs of the flows on each arc. The proposed algorithm evolves a pool of solutions using principles of scatter search, interlinked with an iterated local search as a post-improvement method. New cycle-based neighbourhood operators are presented which enable complete or partial rerouting of multiple commodities. An efficient perturbation strategy, inspired by ejection chains, is introduced to perform local compound cycle-based moves to explore different parts of the solution space. The algorithm also allows infeasible solutions violating arc capacities within the "ejection cycles" and restores feasibility by systematically applying correction moves. Computational experiments on benchmark MCNDP instances show that the proposed solution method consistently produces high-quality solutions in reasonable computational times.

Keywords: Multi-commodity network design, scatter search, evolutionary algorithms, ejection chains, iterated local search

Acknowledgements. Partial funding for this project has been provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), through its Discovery Grant programs. With regard to the UK, the funding has been provided by the Engineering and Physical Sciences Research Council (EPSRC), through its Standard Grant Scheme. We also gratefully acknowledge the support of Fonds de recherche du Québec, Canada, through their infrastructure grants. Lastly, this research was completed when the first author was a member of the Centre for Operational Research, Management Science and Information Systems (CORMSIS) and the Schools of Management and Mathematics at the University of Southampton.

* Corresponding author: TeodorGabriel.Crainic@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2013

1 Introduction

The fixed-charge multi-commodity network design problem (MCNDP) consists of designing a network on a given graph by selecting arcs to route a set of commodities between origin-destination pairs. Each arc of the graph has a predefined capacity specifying the maximum flow that the arc can accommodate, a fixed cost that is incurred only if the arc is selected, and a variable cost proportional to the amount of flow along the arc. Each commodity is defined by an origin and a destination node and the amount to be transported. The objective is to minimize the total cost of establishing the arcs and routing the flows.

The MCNDP has attracted much attention in the literature due to both its complexity (the problem is NP-hard in the strong sense), and a wide variety of applications in the areas of telecommunications, logistics, production and transportation systems (Balakrishnan, Magnanti and Mirchandani, 1997; Magnanti and Wong, 1986; Minoux, 1986). Despite the significant efforts devoted to the development of exact methodologies for the MCNDP (Crainic, Frangioni and Gendron, 2001; Holmberg and Yuan, 2000; Hewitt, Nemhauser and Savelsbergh, 2010), the literature still favours heuristic approaches when large-scale problem instances are involved. One of the most successful local search strategies for the MCNDP is proposed by Ghamlouche, Crainic and Gendreau (2003), where new cycle-based neighborhood operators are incorporated in a tabu search framework. The cycle-based operators are subsequently used within a path-relinking algorithm (Ghamlouche, Crainic and Gendreau, 2004), a multilevel cooperative framework (Crainic, Li and Toulouse, 2006), and a scatter search (SS) (Crainic and Gendreau, 2007). In the latter paper, the authors conclude that the proposed SS failed to meet their expectations and further research is needed to realize the full potential of SS.

Inspired and motivated by the advances in the heuristic approaches for the MCNDP, this paper contributes to the existing body of work by: (i) proposing an efficient iterated local search (ILS) that utilizes new and enhanced cycle-based neighbourhood operators, long and short term memory structures, and an innovative perturbation strategy based on ejection chains (Glover, 1996) that aims at guiding the search towards unexplored solution neighbourhoods; (ii) describing an efficient SS that dynamically adjusts the preferences for inherited solutions based on search history; and (iii) presenting results of computational experiments conducted on benchmark instances using an algorithm incorporating the various elements described above. The majority of the heuristics for the MCNDP utilize a trajectory-based or an evolutionary framework to select arcs for inclusion in the network and call a commercial optimizer (e.g., CPLEX) to solve the flow subproblem. As the flow subproblems becomes larger, the solution time for repeatedly finding minimum cost flows might become significant, even though linear programming optimizers are relatively efficient. Towards this end, we call CPLEX as few times as possible in order to reduce the computational time requirements.

The remainder of this paper is organized as follows. Section 2 provides a brief review of the recent literature on the MCNDP. Section 3 presents the evolutionary algorithm and all of its components, namely the initialization phase, the SS, and the ILS. In Section 4, we describe details of the computational experiments including parameters tuning, and we also present results of applying the proposed algorithm to benchmark MCNDP instances from the literature. Conclusions are given in Section 5, where future research directions are also presented.

2 Literature

A number of efficient algorithms have appeared in the literature to address the inherent complexity of solving the MCNDP. In this section, we provide a brief review of the available methods but focus on heuristic, as opposed to exact, solution algorithms for reasons stated earlier.

Crainic, Gendreau and Farvolden (2000) propose a simplex-based tabu search method for the MCNDP using a path-flow based formulation of the problem. Their method combines column generation with pivot-like moves of single commodity flows to define the path flow variables. In a similar fashion, Ghamlouche, Crainic and Gendreau (2003) describe cycle-based neighbourhoods for metaheuristics aimed at solving MCNDPs. The main idea of the cycle-based local moves is to redirect commodity flows around cycles in order to remove existing arcs from the network and replace them with new arcs. This concept helps to enrich the solution neighbourhood, since multiple commodities can be considered for rerouting and the range of moves is broader because flow redirections are no longer restricted to paths linking origins and destinations. They use the proposed neighbourhood structures in a tabu search algorithm, where a commodity flow subproblem is solved to optimality at each iteration.

Ghamlouche, Crainic and Gendreau (2004) propose an evolutionary algorithm for the MCNDP. Their solution framework is based on path relinking, in which cycle-based neighbourhoods are used to generate an elite candidate set of solutions in a tabu search algorithm and for moving from the initial to the guiding solution. Various rules are presented for choosing the initial and guiding solutions. When updating the pool of solutions, the dissimilarity of solutions is considered as an additional component in the solution value. Later, Alvarez, Gonzalez-Velarde and De-Alba (2005) describe an SS algorithm for the MCNDP. They use GRASP, originally proposed by Feo and Resende (1995), to produce a diversified initial set of solutions. Each commodity path is subject to a post improvement process. The solutions are combined by choosing the best path for each commodity, among the solutions that are being combined. A feasibility restoration mechanism is also available for solutions that are infeasible. The authors randomly generate instances to evaluate the performance of the proposed methodology. In contrast to

the recombination process of Alvarez, Gonzalez-Velarde and De-Alba (2005), our SS does not consider commodity paths to build a solution; instead, independent arcs are combined to create the solutions. A parallel cooperative strategy to solve the MCNDP is described by Crainic and Gendreau (2002) using tabu search and various communication strategies. In a similar fashion, Crainic, Li and Toulouse (2006) propose a multilevel cooperative search on the basis of local interactions among cooperative searches and controlled information gathering and diffusion. The focus of their algorithm is on the specification of the problem instance solved at each level and the definition of the cooperation operators.

Katayama, Chen and Kubo (2009) propose a column and row generation heuristic for solving the MCNDP. The main idea stems from the relaxation of the arc capacity constraints, while a column and row generation technique is developed to solve the relaxed problem. Using similar ideas, Yaghini, Rahbar and Karimi (2012) present a hybrid simulated annealing (SA) and column generation (CG) algorithm for solving the MCNDP. SA is used to define the open and closed arcs, while the flow subproblem is solved via the CG component.

A local branching technique for solving the MCNDP is proposed by Rodrguez-Martin and Salazar-Gonzalez (2010). Even though the method, originally proposed by Fischetti and Lodi (2003), is exact by nature, high quality heuristic solutions can be produced using an MIP solver as a "black box". A solution framework that employs a combination of mathematical programming algorithms and heuristic search techniques is introduced by Hewitt, Nemhauser and Savelsbergh (2010). Their methodology uses very large neighbourhood search in combination with an IP solver on an arc-based formulation of the MCNDP, and a linear programming relaxation of the path-based formulation using cuts discovered during the neighbourhood search. A follow-up study by Hewitt, Nemhauser and Savelsbergh (2012) introduces a generic branch-and-price guided algorithm for integer programs with an application to the MCNDP.

3 Solution Methodology

In this section, we first present a formal definition of the problem including the notation that will be used in the rest of the paper and then describe in detail the components of the main algorithm.

3.1 Problem definition

The MCNDP is defined on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. A subset of the arcs of $\mathcal{A}$ is to be selected. Each arc $(i, j) \in \mathcal{A}$ has an

associated fixed cost f_{ij} to be paid if it is selected for inclusion in the network, has a cost per unit of flow c_{ij}, and has a capacity u_{ij}. A set of commodities denoted by $\mathcal{P}$ is given, where each commodity has an origin, a destination, and a quantity to be shipped from origin to destination. Problems with more than one origin or destination per commodity can be reduced to this form by splitting commodities (see Holmberg and Yuan, 2000).

The goal of the problem is to define a set of arcs that are to be included in the final design of the network along with the commodity flows that these arcs accommodate, to minimize the total cost of the selected arcs and the optimal flow distribution on the resulting network. For the sake of simplicity, we will refer to the arcs that are included in the final design of the network as *open* arcs; otherwise, the arcs should be considered as *closed*. The selection of arcs to be open or closed is represented by the binary variables y_{ij}, where $y_{ij} = 1$ if the arc $(i, j) \in \mathcal{A}$ is open, and $y_{ij} = 0$ otherwise. The flow on each arc $(i, j) \in \mathcal{A}$ that is used for shipping each commodity $p \in \mathcal{P}$ from its origin to its destination is denoted by x_{ij}^p. Conservation of flow constraints must be satisfied at each node, and $\sum_{p \in \mathcal{P}} x_{ij}^p \leq u_{ij}$ for all $(i, j) \in \mathcal{A}$. The cost of a solution s that is defined by variables x_{ij}^p and y_{ij} for $(i, j) \in \mathcal{A}$ and $p \in \mathcal{P}$ is denoted by:

$$f(s) = \sum_{(i,j) \in \mathcal{A}} \sum_{p \in \mathcal{P}} c_{ij} x_{ij}^p + \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij}. \tag{1}$$

Two types of mathematical formulations for the problem appear in the literature; an arc-based and a path-based formulation. We refer to Gendron, Crainic and Frangioni (1998), Frangioni and Gendron (2001) and Hewitt, Nemhauser and Savelsbergh (2010) for details on these mathematical formulations.

3.2 Evolutionary algorithm

Our proposed solution methodology is an evolutionary algorithm that evolves a population of solutions using the principles of SS (Glover, 1996) and applies ILS (Lourenço, Martin and Stützle, 2002) as an improvement method. Following the basic template of the SS framework, our solution approach can be mainly presented in three distinct phases: (i) an *Initialization phase* where a population of good and diverse solutions is produced and the Reference Set (set R) is initialised; (ii) a *Scatter Search phase* where a recombination process takes place to produce offspring; and (iii) an *Education phase* where these offspring (hosted in set C) are "educated" by attempting to improve their quality via the proposed ILS. A pseudo-code of the overall solution framework is given in Algorithm 1. We explain each phase in more detail in the following subsections.

Algorithm 1: Evolutionary Algorithm

Input: λ (initial population size), μ (Reference Set size), where $\lambda \geq \mu$, δ (number of local search iterations without an improvement), κ (Candidate Set size), $\vartheta_{\max}$ (number of CPLEX calls within local search without an improvement)

Output: R, $s_{\text{best}} \in R$

1. Initialization phase

$R \leftarrow \text{ConstructionHeur}(\lambda, \mu)$;

while *termination conditions* **do**

  2. Scatter Search phase

  $C = \text{SolutionCombination}(\kappa, \mu)$;

  3. Education phase

  for *individual* s *of* C **do**

    $s' \leftarrow \text{ILS}(s, \delta, \vartheta_{\max})$;

    Update $\text{RefSet}(R, s')$;

Typical termination criteria for evolutionary algorithms include the number of generations or a computation time limit. To be able to compare the performance of our solution methodology to the literature, a computation time limit of one hour was used.

3.3 Initialization phase

The *initialization phase* uses a constructive heuristic to produce a population of diversified and good-quality solutions. The heuristic is based on three rules that are described below.

The **first rule** selects an unrouted commodity that has the largest total demand, and finds the best path on which the commodity can be routed without splitting. The procedure iterates in a similar fashion until as many commodities as possible are routed. Because the procedure does not necessarily route all of the commodities, the next rule considers splitting the commodity flows into parallel paths as described below.

The **second rule** selects an unrouted commodity with the smallest demand and attempts to route as many units of this commodity as possible along the minimum cost path from origin to destination. More specifically, the procedure starts by routing all the units of the demand of a particular commodity over the minimum cost path (as determined by Dijkstra's algorithm). If the arcs capacity constraints are violated by routing all the units of this commodity, feasibility is restored by removing the excess flow. The algorithm then proceeds to route this excess flow by considering alternative and more expensive paths that are chosen in a non-decreasing order of their total cost. The procedure iterates until all units of all commodities are routed. The rationale behind

choosing commodities with smallest demand first is to force the procedure to split the commodities with a larger demand in preference to those with a smaller demand.

The **third rule** is the combination of the first two and incorporates randomization for diversification purposes. In particular, the randomization chooses with equal probability, at each iteration, one of the two above described rules to route the next unrouted commodity. This commodity will be the one with the largest demand if the first rule is chosen, or the one with the smallest demand if the second rule is involved. The solutions produced by this rule should therefore involve some of the commodities routed as a whole along a single path, while other commodities may be split and routed along several parallel paths.

In each of the above rules, the paths are determined using Dijkstra's shortest-path algorithm. The cost of an arc, considered by Dijkstra, is the sum of the flow cost and the fixed cost incurred if the arc is open. Given the set of all open arcs, a linear programming flow subproblem is then solved using an off-the-shelf commercial solver (CPLEX, in our implementation), to obtain optimum commodity flows. The initialization phase terminates after λ (different) solutions have been generated. The aim is to select μ of these solutions to form the initial Reference Set, as described below.

3.4 Scatter Search

The SS phase evolves the Reference Set (Glover, 1996) of solutions using an efficient recombination method as follows. A subset generation method selects κ solutions from the Reference Set, and a solution combination method is then applied to produce one solution. This procedure is repeated until 2μ offspring, which double the number of parent solutions in the Reference Set, are produced. We choose μ of the 2μ offspring that have the smallest total cost to proceed to the next phase. In this Education phase, the ILS then improves the quality of each offspring, before these offspring are tested for insertion into the Reference Set according to elitist criteria. These phases are explained further in the following sections.

3.4.1 Reference Set

The goal of using a Reference Set R is to maintain a balance between quality and diversity, and to avoid a premature convergence of the algorithm. An obvious measure of the quality of a solution s is its cost $f(s)$. An alternative quality measure that becomes relevant after the evolutionary process has started is the *solvency ratio* as defined by

$$\mathrm{SR}(s) = \mathrm{NEO}(s)/\mathrm{hits}(s), \tag{2}$$

where hits(s) denotes the number of times that solution s has participated in the recombination process to produce an offspring, and NEO(s) denotes the Number of Educated Offspring of s, which is the number of times that an offspring of s has been educated and included in R. The smaller the solvency ratio, the lower the value of the particular solution s is to the evolution process. In this way, a higher cost solution with respect to the usual objective function f may be beneficial to the search if it produces well-educated offspring. Our diversity measure uses the Hamming distance between pairs of solutions. For solutions s and s', the Hamming distance is defined by

$$D(s, s') = \sum_{(i,j)\in\mathcal{A}} |y_{ij}^{s} - y_{ij}^{s'}|. \tag{3}$$

The total dissimilarity of the reference set is then defined by

$$\mathrm{TD}(R) = \sum_{s,s'\in R} D(s, s'), \tag{4}$$

where the sum is over all $\mu(\mu-1)/2$ pairs of solutions in set R.

The creation of the initial Reference Set proceeds as follows. The first μ solutions among the λ generated within the Initialization phase are inserted into R. The remaining $\lambda-\mu$ solutions are then considered sequentially for replacing a solution in R. Specifically, if such a solution s satisfies the condition $f(s) < f(s_{\mathrm{best}})$, where s_{best} is a least cost solution in R, or if there is a solution $r \in R$ for which $f(s) < f(r)$ and $D(r, s_{\mathrm{best}}) < D(s, s_{\mathrm{best}})$, then s is inserted into R. Otherwise, s is not included in R and is discarded from further consideration. To maintain the size of R when s is inserted, solution $s_{\mathrm{worst}} \in R$, where s_{worst} has the largest cost among solutions in R, is removed from R.

At each generation of the evolutionary process, μ educated offspring are generated, and sequential decisions are made on whether to replace a solution in R with the offspring under consideration. At the later stages of the evolutionary process, this decision depends on solvency ratios, but a different process is used at the start of the evolutionary process when solvency ratios cannot be meaningfully computed. Specifically, for the first two generations of the evolutionary process, an offspring s replaces $s_{\mathrm{worst}} \in R$ in the reference set R if either $f(s) < f(s_{\mathrm{best}})$, or if $f(s) < f(s_{\mathrm{worst}})$ and $\mathrm{TD}(R) < \mathrm{TD}(R \setminus \{s_{\mathrm{worst}}\} \cup \{s\})$, where s_{best} and s_{worst} are defined above and total dissimilarity values are computed using equation (4).

After the first two generations of the evolutionary process, decisions about replacing a solution of the reference set with an educated offspring s use solvency ratios. More precisely, if either $f(s) < f(s_{\mathrm{best}})$, or if $f(s) < f(s'_{\mathrm{worst}})$ and $\mathrm{TD}(R) < \mathrm{TD}(R \setminus \{s'_{\mathrm{worst}}\} \cup \{s\})$, where $s'_{\mathrm{worst}} \in R$ has the minimum solvency ratio among solutions in R, then s replaces s'_{worst} in R. This procedure differs from other studies where the usual practice is always to remove the worst-cost solution s_{worst} from R without regard to any effect on the evolutionary process.

3.4.2 Solution combination method

To create a new generation, 2μ offspring are generated, and the best μ of these are improved, or "educated", through our ILS. The educated offspring are then considered for inclusion in R, as explained in Section 3.4.1. We now discuss how our proposed solution combination method generates each offspring.

Each offspring is generated from a *candidate set* (CS) comprising κ solutions from the Reference Set R. The solutions in CS are chosen probabilistically with a bias towards promising parents as determined by their solvency ratios. Specifically, the probability of a solution s being included in the candidate set is proportional to $\mathrm{SR}(s)$. In this way, a solution s with a low $\mathrm{SR}(s)$ is gradually neglected, and the focus is on new solutions that produce well-educated offspring. Because the solvency ratio changes as new generations are created, the scatter search phase has a dynamic character, and premature convergence is typically averted. Furthermore, to enable diversification, a penalty is used to weaken the impact of a frequently selected parent as explained below.

The solution elements, which are the arcs of the solutions in CS, are combined to produce an offspring. For a given solution s, each arc (i, j) is either open if $y^s_{ij} = 1$ or closed if $y^s_{ij} = 0$. We associate a value $f(s) + \alpha\mathrm{hits}(s)$ with solution s, where $f(s)$ is the cost of the solution, $\mathrm{hits}(s)$ is the number of times that solution s has participated in the recombination process over all previous iterations, and α is a scaling parameter. The term $\alpha\mathrm{hits}(s)$ is introduced to weaken the impact of a frequently selected parent and thereby enable diversification. We now introduce a voting procedure to determine whether each arc (i, j) will be open or closed in the offspring, according to the following scores:

$$\mathrm{Op}_{ij} = \sum_{s \in \mathrm{CS}} \frac{y^s_{ij}}{f(s) + \alpha\mathrm{hits}(s)} \qquad \forall (i, j) \in \mathcal{A} \tag{5}$$

$$\mathrm{Cl}_{ij} = \sum_{s \in \mathrm{CS}} \frac{1 - y^s_{ij}}{f(s) + \alpha\mathrm{hits}(s)} \qquad \forall (i, j) \in \mathcal{A}. \tag{6}$$

Op_{ij} and Cl_{ij} are the scores for arc (i, j) being open and closed, respectively. Thus, if $\mathrm{Op}_{ij} > \mathrm{Cl}_{ij}$, the preferred status of arc (i, j) is open; otherwise its preferred status is closed.

The output of the SS phase is a preferred status of open or closed for each arc (i, j). To build a feasible solution, the commodity flows on each arc of the network, given the status of the arcs, need to be determined. We adopt the natural approach of solving the associated capacitated multicommodity network flow problem to find these flows, and the CPLEX LP optimizer is used for this purpose. Due to the nature of the recombination process, it might be that the set of arcs that are open according to their preferred status leads to an infeasible solution. To restore feasibility, a reconstruction mechanism is also

incorporated, similar to the one described in Section 3.3. The goal is to maintain as many open arcs as the recombination process suggests. Towards this end, the arcs with a preferred status of open are each assigned a very small positive fixed and variable costs, so that the construction heuristic will generally choose them, while other arcs are assigned their original costs.

3.5 Education phase: the ILS heuristic

The μ elite offspring, chosen among the 2μ produced by the SS phase, are individually "educated" (i.e., improved) using ILS. The resulting improved solutions are then used to update the Reference Set according to the criteria discussed in Section 3.4.1.

The proposed ILS has two main components, namely a local search and a perturbation strategy. The proposed local search uses new neighbourhood operators and short term memory (represented by memory structure $\vec{g}$) to avoid cycling. The perturbation strategy, namely Ejection Cycles, partially modifies the current solution according to information gathered during the search (long-term memory represented by memory structure $\vec{h}$) in the spirit of Ejection Chains (Glover 1996). The components of the ILS are given in Algorithm 2.

Algorithm 2: Iterated Local Search

Input: s (current offspring), δ (number of local search iterations without an improvement), $\vartheta_{\max}$ (number of CPLEX calls without an improvement)
Output: s
$\vartheta \leftarrow 0$, $\vec{h} \leftarrow 1$;
while $\vartheta < \vartheta_{\max}$ **do**
  $\vec{g} \leftarrow 0$;
  $(s', \vec{h}) \leftarrow \text{LocalSearch}(\delta, s, \vec{g})$;
  $s' \leftarrow \text{CPLEX}(s')$;
  if $f(s') \geq f(s)$ **then**
    $s^* \leftarrow \text{EjectionCycles}(s', \vec{h})$
    $s \leftarrow s^*; \vartheta \leftarrow \vartheta + 1$;
  else
    $\vartheta \leftarrow 0; s \leftarrow s'$;

3.5.1 Neighbourhoods and moves

Our ILS neighbourhood is based on the cycle-based operator, as originally proposed by Ghamlouche, Crainic and Gendreau (2003). Their approach is to select a pair of nodes

containing a positive flow and then re-route the flows of the individual commodities between these nodes. In this paper, we design a more efficient and effective approach based on the notion of inefficient arcs and inefficient chains, as described below. Further, we allow a partial re-routing of flow that maintains flow feasibility. In contrast, Ghamlouche, Crainic and Gendreau (2003) remove all flow between the two selected nodes, and if the new flows that are added do not result in a feasible solution, then a feasibility restoring routine is applied.

Consider a solution defined by the variables x_{ij}^p and y_{ij} for each arc $(i,j) \in A$ and each commodity $p \in \mathcal{P}$. For each open arc (i,j), where $y_{ij} = 1$ and $x_{ij}^p > 0$ for at least one commodity p, we define the *inefficiency ratio* as

$$I_{ij} = \frac{\sum_{p \in P} c_{ij} x_{ij}^p + f_{ij}}{\sum_{p \in P} x_{ij}^p}, \tag{7}$$

which is a measure of the cost incurred for each unit of flow that is sent along arc (i,j). The lower the value of I_{ij}, the more efficient we regard arc (i,j) for accommodating flows. The average inefficiency ratio is defined as $\bar{I} = \sum_{(i,j) \in A} I_{ij} y_{ij} / \sum_{(i,j) \in A} y_{ij}$, and we define a set of *inefficient arcs* as $\mathcal{A}_I = \{(i,j) | y_{ij} = 1, I_{ij} > \bar{I}\}$, so that $(i,j) \in \mathcal{A}_I$ if arc (i,j) has a higher than average inefficiency ratio. Our aim is to create neighbourhood moves that remove flows from some of the inefficient arcs in set $\mathcal{A}_I$.

We now describe how our *inefficient chains* are constructed from a subset of the inefficient arcs. First, an arc is randomly chosen from the set $\mathcal{A}_I$ of inefficient arcs to form a component of the first inefficient chain. If the current partial inefficient chain extends from node i to node j, then an arc $(h,i) \in \mathcal{A}_I$ or $(j,k) \in \mathcal{A}_I$ is added to the current chain (where nodes h and k are not included in the current chain), where the added arc is chosen to have an inefficiency ratio that is as large as possible. Whenever an arc is included in a chain, it is deleted from $\mathcal{A}_I$. The process of extending the current chain continues until no further extension is possible. Unless $\mathcal{A}_I$ is empty or contains a single arc, the process iterates with a random arc chosen to start a new chain. When the process ends, any chains containing a single arc are discarded. The latter have the chance to be included in inefficient chains in the next ILS iterations, since inefficient chains are reconstructed from scratch at each ILS iteration.

Having constructed a set K of inefficient chains, we now describe how our neighbourhood is formed. Each neighbour is based on a sub-chain of an inefficient chain $k \in K$ and is defined by the starting node i and the ending node j of the sub-chain. If a chain comprises nodes $n_1 - n_2 - \cdots - n_m$, then the (i,j) values are considered in the order

$$(n_1,n_2),(n_1,n_3),\ldots,(n_1,n_m),(n_2,n_3),(n_2,n_4),\ldots,(n_2,n_m),\ldots,(n_{m-1},n_m).$$

However, based on our initial computational tests, we restrict our attention to sub-chains between i and j comprising at most 4 arcs, which helps to reduce computation times while not significantly restricting the diversity of potential neighbourhood moves.

The key aspect of our neighbourhood is the re-routing of flow from arcs of the sub-chain to other arcs of the network. An initial random decision is made as to whether a *full re-routing* or a *partial re-routing* is to be attempted for this sub-chain. First, a list $\mathcal{P}_I$ of commodities is formed that have a positive flow through at least one arc of the sub-chain, and this list is ordered in non-increasing order of amount commodity flow in the sub-chain. To obtain a neighbour, the list of commodities is scanned and a re-routing of flow is attempted for each commodity p of $\mathcal{P}_I$ in turn. Suppose that the flow enters the sub-chain at node i^p, leaves the sub-chain at node j^p and the amount of flow is v^p. Dijkstra's algorithm is then applied to find a shortest path from node i^p to node j^p with the goal of finding a suitable re-routing of flow. The arc costs for the shortest path problem are set to infinity for the arcs between i^p and j^p in the sub-chain to prevent them from being selected to receive flow. For the full re-routing neighbourhood, the arc cost is set to infinity for any arc (i, j) that cannot accommodate an additional v^p units of flow, is set to $c_{ij}v^p$ for other arcs (i, j) that are open in the current solution and is set to $c_{ij}v^p + f_{ij}$ for other arcs (i, j) that are closed in the current solution. Further, for partial re-routing, the arc cost is set to infinity for any arc (i, j) that cannot accommodate any additional flow, is set to $c_{ij} \min\{\bar{u}_{ij}, v^p\}$ for other arcs (i, j) that are open in the current solution, where $\bar{u}_{ij}$ is the unused capacity in arc (i, j), and is set to $c_{ij} \min\{\bar{u}_{ij}, v^p\} + f_{ij}$ for arcs (i, j) that are closed in the current solution. If, in either neighbourhood, Dijkstra's algorithm does not find a shortest path with value less than infinity, then the flow remains unchanged in the trial solution being constructed and the procedure turns its attention to the next commodity in the list. When Dijkstra's algorithm finds a shortest path with cost less than infinity, we proceed as follows. For full re-routing, a flow of value v_p is added to each arc of the shortest path in the trial solution and is removed from the arcs of the sub-chain. For partial re-routing, the maximum flow z^p that can be sent along the shortest path is first determined. The change of flow is then $\min\{v^p, z^p\}$ with this flow being added to the arcs in the shortest path in the trial solution and deleted from the flows in the sub-chain. When all of the commodities of $\mathcal{P}_I$ are considered, the trial solution is a potential candidate for being selected as the preferred neighbour. Additional trial solutions are created by removing the first element of list $\mathcal{P}_I$ and repeating the process, starting with a random decision as to whether a full or partial re-routing is to be attempted, until $\mathcal{P}_I$ is empty. The completed procedure is executed for every possible sub-chain.

We illustrate the idea of re-routing flows by an example shown in Figure 1. The example shows three commodities each with different line pattern, and a graph where an origin node 3 and a destination node 8 define part of the inefficient chain. The re-routing of the flows between nodes 3 and 8 causes individual commodity flow disconnections. The flow re-routings take place independently for each different commodity between its origin and destination nodes that the flow is disconnected. In particular, the solid black line commodity must travel from node 4 to node 7, the dotted must travel from node 6 to node 8, and the dashed one from node 3 to node 8. The gray solid lines are possible alternative re-routing paths within the network. The three reroutings take place separately as described above, although they result in a single neighbour.

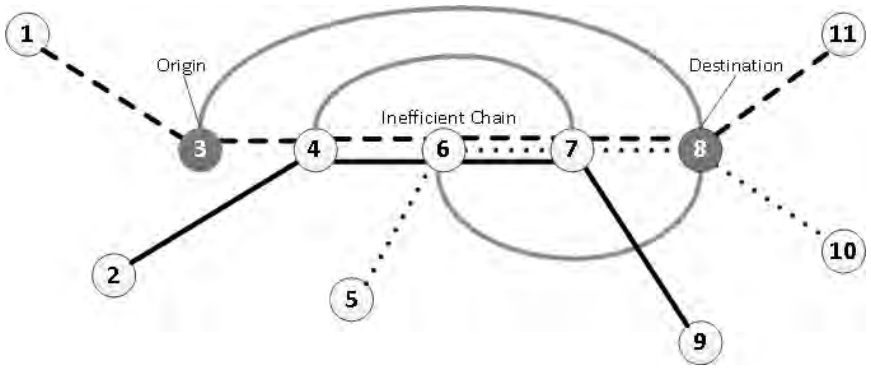


Figure 1: A typical Inefficient Chain and a Cycle operator

Another important component of our ILS is a frequency-based memory feature following the suggestions of Paraskevopoulos, Tarantilis and Ioannou (2012). For this purpose, a vector $\vec{g}$ of size equal to the number of arcs in a given instance is used to store each value g_{ij}, which is the number of times that arc (i, j) has participated in a local move. We re-initialize $\vec{g}$ to 0 each time a new best solution is found.

Using equation (8) below, the local move cost from solution s to a trial solution s' is computed as

$$\Delta f_{\text{move}} = f(s') - f(s) + \beta \sum_{(i,j)\in A} b_{ij} g_{ij}, \tag{8}$$

where β is a scaling parameter, and b_{ij} a binary parameter with value equal to 1 if the arc (i, j) participates in the current local move from s to s', and 0 otherwise. The component $\beta \sum_{(i,j)\in A} b_{ij} g_{ij}$ is added to the cost of the local move, to penalize moves that involve frequently selected arcs.

Trial solutions with smaller values of Δf_{move} are generally preferred. However, it may happen that this number is large enough to prevent the search from selecting a high-quality neighbour s'. To avert such cases, an aspiration criterion is used according to which if $f(s') < f(s_{\text{best}})$ the penalty component is ignored so that $\Delta f_{\text{move}} = f(s') - f(s)$. The trial solution s' with the smallest value of Δf_{move} is then selected by ILS to be improved by CPLEX (see Algorithm 2 for details on the ILS framework).

The neighbourhood search procedure is described by Algorithm 3. The function *IdentifyDifferentCommodities* forms the list $\mathcal{P}_I$ by identifying the different commodities that have positive flows between the nodes i and j of an inefficient chain and arranging the list in non-increasing order of these flows. *EvaluateMove* calculates the total cost of the local move, and *RemoveFirstElement* removes the first element of the list. Recall that the list is ordered in descending order of the flows of the different commodities. The *isFeasible* function is a boolean function that returns "true" if a particular combination (k, i, j) leads to some re-routing of flow, and lastly *StoreBestNeighbour* stores the current neighbour found in the solution s'.

Algorithm 3: Neighborhood Search

Input: s (current solution), M a large number
Output: s' (best neighbor)
min = M;
for *All inefficient chains k and for all combinations of nodes i, j in k* **do**
  $\mathcal{P}_I \leftarrow$ IdentifyDifferentCommodities(k, i, j);
  while $\mathcal{P}_I$ *is not empty* **do**
    if *isFeasible*$(k, i, j, \mathcal{P}_I)$ **then**
      move=EvaluateMove$(k, i, j, \mathcal{P}_I)$;
    else
      RemoveFirstElement$(\mathcal{P}_I)$;Continue;
    if move $<$ min **then**
      $s' \leftarrow$ StoreBestNeighbour$(k, i, j, \mathcal{P}_I)$; min = move;
    RemoveFirstElement$(\mathcal{P}_I)$;

3.5.2 Ejection Cycles

A major component of an ILS algorithm is its perturbation strategy (Lourenço, Martin and Stützle 2002). The goal is to partially rebuild the current local optimum solution, such that the new diversified solution preserves some information from the local optimum. The proposed perturbation strategy in this paper, namely Ejection Cycles (EC), applies multiple cycle-based moves in the spirit of ejection chains (Glover 1996). The main idea of the ejection-chains strategy is to apply a compound move consisting of a series of consecutive local moves. Adopting this idea, our EC comprise a series of consecutive cycle moves of the type described in Section 3.5.1, which aim at perturbing the structure of the current solution to achieve diversification, but with the perturbation removing some of the inefficient arcs from the solution.

We now present the details of how our sequence of local moves is found. As in Section 3.5.1, we first find a set of inefficient chains and focus on sub-chains containing 4 arcs or fewer. For a given sub-chain, the list $\mathcal{P}_I$ is formed, and i^p, j^p and v^p are computed. The list of commodities is scanned and a re-routing of flow is performed for each commodity p of $\mathcal{P}_I$ in turn. However, in this re-routing, infeasibilities may occur due to flows exceeding arc capacities. A full re-routing is performed by applying Dijkstra's algorithm to find a shortest path from i^p to j^p. For all applications of Dijkstra's algorithm that are based on this sub-chain, the arc costs for the shortest path problem are set to infinity for the arcs between i^p and j^p in the sub-chain. For each of the other arcs (i, j), the cost in the shortest path problem is set to $c_{ij}h_{ij}$ for arcs that are open in the current solution and is set to $c_{ij}h_{ij} + f_{ij}$ for arcs that are closed, where $h_{ij} - 1$ is the number of times that arc (i, j) has participated in a local move, and initialization sets $h_{ij} = 1$. The value h_{ij} is similar to g_{ij} of Section 3.5.1 except that it is initialized differently and,

unlike g_{ij} is never re-initialized. If Dijkstra's algorithm returns a shortest path length of infinity, then the current sub-chain is not considered further and another is selected. Otherwise, a flow of value v_p is added to each arc of the shortest path in the trial solution and is removed from the arcs of the sub-chain.

When rerouting of flow between nodes i^p and j^p of the sub-chain is complete for each $p \in \mathcal{P}_I$, we check if any arc has a flow that violates its capacity constraint. If there is no capacity violation, then a new feasible solution is found and the EC strategy terminates because a perturbed solution has been found. When there some flows violate arc capacities, we proceed as follows. For each arc with a capacity violation, that comprises the set of violated arcs $\mathcal{A}_V$, a set of commodities is selected whose removal from the arc restores feasibility but keeps the capacity utilization of the arc as high as possible. Thus, each arc (i, j) with a capacity violation has an associated set of commodities P_{ij} whose flows should be removed to restore feasibility. For each commodity p, the set of arcs $\{(i, j)|p \in P_{ij}\}$ is found, and a series of *infeasibility chains* is formed in the same way as for inefficient chains, as described in Section 3.5.1.

The aim now is to re-route the flow in the each infeasibility chain using the method described above incorporating Dijkstra's algorithm to find a shortest path from the the starting node i^p of the sub-chain to the ending node j^p. If a suitable path for re-routing is found, then the trial solution is updated and a cost of infinity is assigned to each arc of the infeasibility chain in all subsequent shortest path problems. The process or re-routing flow in other infeasibility chains continues, until no capacity violations at arcs occurs. If this is the case, then the EC strategy terminates because a feasible perturbed solution has been found. Since at each rerouting a cost of infinity is assigned to each arc, it happens all the possible paths to be penalised and no feasible solution to be found. In such case, the EC procedure returns to the initial feasible solution, the first commodity of set $\mathcal{P}_I$ is deleted and EC takes place for the remaining commodities in the set. As in Section 3.5.1, additional trial solutions are created by removing the first element of list $\mathcal{P}_I$ and repeating the process until $\mathcal{P}_I$ is empty. The complete procedure is applied to all subchains, and terminates when the first feasible perturbed solution is found.

The pseudo code of the EC is given in Algorithm 4, which has a close resemblance to Algorithm 3. The function $IdentifyInefficientArcs$ initialises the set of violated arcs $\mathcal{A}_V$ with the inefficient arcs that are associated with the commodities in set $\mathcal{P}_I$. The function $NeighbourExists$ is a boolean function that returns "true" if there exist an alternative path that the flow can be re-routed, regardless the capacity constraints at arcs. If no alternative paths are found (in case all neighbouring arcs have been assigned a cost of infinity), then $NeighbourExists$ returns "false". The function $Update$ identifies which of the arcs of the re-routed paths are violate in terms of capacity constraints and updates the set of violated arcs $\mathcal{A}_V$.

Algorithm 4: Ejection Cycles

Input: s (current solution), M a large number
Output: s' (best neighbor)
min $= M$
for *All inefficient chains k and for all combinations of nodes i, j* **do**
  $\mathcal{P}_I \leftarrow$ IdentifyDifferentCommodities(i, j);
  while $\mathcal{P}_I$ *is not empty* **do**
    $\mathcal{A}_V \leftarrow$IdentifyInefficientArcs$(\mathcal{P}_I)$;
    while $\mathcal{A}_V$ *is not empty* **do**
      if *NeighbourExists($k, i, j, \mathcal{A}_V$)* **then**
        move = move + EvaluateMove$(k, i, j, \mathcal{A}_V, \vec{g}, \vec{h})$; Update$(\mathcal{A}_V)$;
      else
        goto another (k, i, j);
      if move $<$ min **then**
        $s' \leftarrow$ StoreBestNeighbour(k, i, j); min = move;
    if move $\neq M$ **then**
      EndAlgorithm;
    RemoveFirstElement$(\mathcal{P}_I)$;

4 Computational Results

This section summarises the experiments conducted to evaluate the performance of the proposed algorithm. First, we describe the benchmark sets that we used as a test bed for our experimentation, followed by a parameter tuning and sensitivity analysis. Next, the impact the inefficiency ratio (Equation 7) and the solvency ratio (Equation 2) have on the solution quality is illustrated. EC's performance is questioned next and the section is completed by providing extensive comparisons of our computational results to the state-of-the-art of the literature.

4.1 Data sets

To evaluate the performance of the proposed algorithm, computational experiments are conducted on the benchmark instances described in Crainic, Gendreau and Farvolden (2000), and in particular the C and C+ instances. These sets include instances with 25, 30 and 100 nodes, 10 to 400 commodities and 100 to 700 arcs. These instances differ with respect to the nature of the arc capacities, which are either loose or tight, and the relative weight of the fixed costs and the costs per unit of flow. There is also a set R of the benchmarks described in Crainic, Gendreau and Farvolden (2000), which were

not considered here as they are all solved to optimality (Katayama, Chen and Kubo, 2009) and are not as challenging as the C and C+ instances. The proposed algorithm was implemented in a Visual Studio 2010 environment using the C++ programming language and run on an Intel i-5 2.5 GHz PC, under MS Windows 7.

4.2 Calibration

The proposed evolutionary algorithm uses five parameters; the number λ of initial solutions examined to produce the Reference Set R, the cardinality μ of R, the cardinality κ of CS, the maximum number δ of local search iterations without an improvement in the solution quality, and the maximum number $\vartheta_{\max}$ of CPLEX calls for which an improvement in the current solution is not observed. The termination criterion for the algorithm is a computational time limit of 1 hour.

Scaling parameters α and β are self-defined during the solution process, equal to the average cost of an arc in the current best solution found, i.e., $\alpha{=}\beta{=}f(s_{\text{best}})/\sum_{(i,j)\in A} y_{ij}^{s_{\text{best}}}$. The parameter λ does not appear to have a significant impact to the quality of the solutions; however, to have an adequate initial population size, we set $\lambda = 1500$. We set $\kappa = 3$: to preserve the SS character of the proposed algorithm, κ needed to be larger than 2 to enhance the recombination process, but on the other hand κ should be relatively small to ensure that a large number of possible combinations among the solutions of the reference set is considered.

Parameters δ and $\vartheta_{\max}$ are interrelated in a way to keep the total number of local search iterations balanced with the number of generations produced within the available computation time limit. In particular, our experiments indicate that values of $\vartheta_{\max}$ equal to 6, 7, and 8 are appropriate. Values below 6 deteriorated the solution quality, while values greater than 8 slowed down the solution process without a significant increase in the solution quality. In an effort to maintain a balance between the number of CPLEX calls and the local search oscillations, δ is defined with respect to $\vartheta_{\max}$ values (as Table 1 shows and will be discussed in the following).

The impact of each algorithmic component on the solution quality varied according to the size of the problem instance. In small to medium scale instances, the evolutionary strategy had more impact than local search, since the cardinality of the neighbourhood is relatively small and local search fails to adequately explore the search space. In contrast, the solution neighbourhood is enriched with more solutions and the local search impact is dominant in the solution process as the size of the problem instance increases. Driven by these observations, the size μ of the reference set takes large values for small-medium scale problems, and relatively small values for large scale instances, as Table 1 shows. The goal is to obtain the best quality solutions within the predefined limit on running time.

Table 1 shows the computational experiments conducted to investigate on the algorithm's behaviour with respect to different sets of parameters. Different parameter sets were used for different groups of problems, i.e., for large scale problems the reference set was given relatively small sizes and δ was assigned high values, while on the small to medium scale the opposite settings were used, for reasons described above.

Table 1: Calibration of the algorithm's parameters

Group	**25-100-(10,30)**		**20-(230,300)-40**		**20-(230,300)-200**	
	$\vartheta_{\max}, \delta, \mu$	25-100-30FT	$\vartheta_{\max}, \delta, \mu$	20,230,40FT	$\vartheta_{\max}, \delta, \mu$	20,230,200VT
Parameter Sets	6,40,40	86102	6,50,30	644180	6,70,20	98209
	6,40,50	85969	6,50,40	644995	6,70,30	98338
	6,40,80	85530	6,50,50	643187	6,70,40	98486
	7,30,40	85996	7,40,30	644413	7,50,20	98584
	7,30,50	85948	7,40,40	644265	7,50,30	98767
	7,30,80	86089	7,40,50	643253	7,50,40	98945
	8,20,40	86059	8,30,30	644085	8,40,20	98451
	8,20,50	85932	8,30,40	643649	8,40,30	98767
	8,20,80	85535	8,30,50	643538	8,40,40	98807
Best	6,40,80	85530	6,50,50	643187	6,70,20	98209
Group	**100-400-(10,30)**		**30-(520,720)-100**		**30-(520,720)-400**	
	$\vartheta_{\max}, \delta, \mu$	100-400-30FT	$\vartheta_{\max}, \delta, \mu$	30,700,100FL	$\vartheta_{\max}, \delta, \mu$	30,700,400FT
Parameter Sets	6,80,20	139661	6,80,20	61085	6,80,10	133822
	6,80,30	139805	6,80,30	61106	6,80,20	133889
	6,80,40	140503	6,80,40	61089	6,80,30	133861
	7,70,20	139995	7,70,20	60596	7,70,10	133245
	7,70,30	140878	7,70,30	61054	7,70,20	133478
	7,70,40	140976	7,70,40	61188	7,70,30	133966
	8,50,20	140878	8,50,20	61284	8,60,10	133596
	8,50,30	139943	8,50,30	61266	8,60,20	133954
	8,50,40	140760	8,50,40	61176	8,60,30	133870
Best	6,80,20	139661	7,70,20	60596	7,70,10	133245

The C and C+ benchmark instances are classified into 6 groups according to their size. The label for each group is a vector depicting the number of nodes, the number of arcs and the number of commodities. The problem instances within each group differ in the tightness of the arcs' capacity constraints and the relative importance of the fixed costs for the arcs and the costs per unit of flow. The calibration was conducted by using one problem from each group, shown in the headings of the three main columns of Table 1. A single run of the algorithm was used and the solution cost derived for each instance is indicated within the second column of the three main columns. The parameter set that produces the best result for each different group is chosen to be fixed and is applied to solve the rest of the problems in the group using a single run of the algorithm.

As expected, small to medium scale problems favour a smaller number of local search iterations and a large Reference Set, while large scale problems needed more local search iterations with a relatively small reference set to produce the best results. Furthermore, it is easily observed that there is no significant variation of the solution costs derived from different parameter sets, concerning the same problem.

4.3 Network efficiency vs total cost

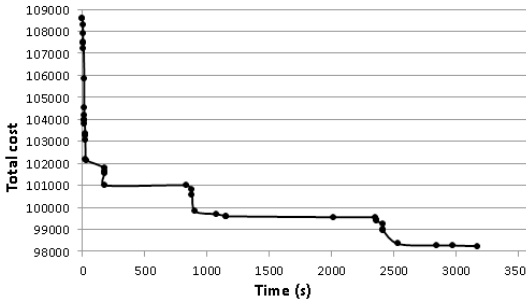
The local search introduced herein is guided by rerouting commodities flows from inefficient to more efficient chains. Nevertheless, the original objective function of the problem is used throughout the solution process and the solutions are evaluated according this objective function and to dissimilarity criteria. To illustrate the impact of the efficiency guidance on the solution cost, we present in Figure 2 the evolution of performance measures for two problem instances.

Figure 2 shows the relation between the efficiency of the network, that is the efficiency of the arcs that are open according to equation (7), and the total cost of the solution during the search. Two instances, 20,230,200VT and 30,700,400VL, were chosen for this analysis. The first two figures, (a) and (b), show the evolution of the total cost for the best solutions found for the 20,230,200VT and 30,700,400VL problem instances, respectively. The next two figures (c) and (d) give the maximum inefficiency of an open arc for the different solutions found during the algorithm's iterations. We observe that as the algorithm iterates, the maximum arc inefficiency is dramatically reduced and follows a logarithmic trend. On another note, the 20,230,200VT graph appears to be steeper than 30,700,400VL, which can be attributed to the different characteristics of the problem instances. Conversely, the last two subfigures (e) and (f) show the average efficiency of the arcs as the search progresses, which illustrate an increase in the efficiency of the network as the solution quality improves.

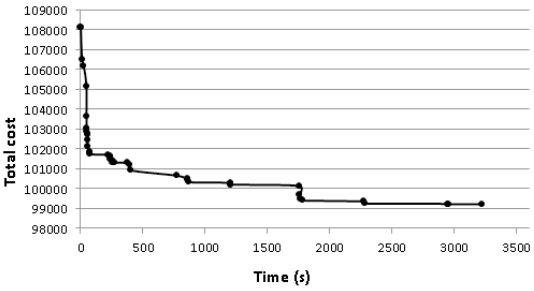
4.4 Solvency ratio vs random parent selection

The solvency ratio (introduced in Section 3.4.1) informs the parent selection procedure during the SS recombination, such that solutions producing offspring which have been subsequently "educated", (i.e., inserted into the Reference Set), are favoured during the parent selection procedure. To illustrate the effectiveness of the solvency ratio, tests were conducted and results are presented on two benchmark instances, which typically present the general behaviour of the algorithm, namely 100-400-30-FT and 20,300,200FT, considering solvency-based and random parent selection strategies.

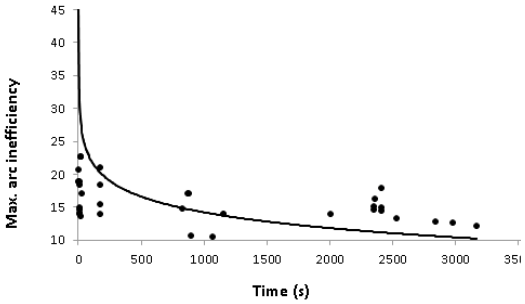
Figure 3 presents the comparisons between the two strategies. Note that the two first generations are used as a warm up for the solvency strategy, which is enabled from the third generation onwards and this is apparent from the subfigures. Subfigures (a) and (b) show how the best solution values evolve over time. For 100,400,40FT we can easily observe that solutions from the random-based strategy become prematurely trapped in a local optimum, whereas the solvency-based strategy is slower to improve the best solution initially, but a pay-off is observed in the middle of the horizon, and finally terminates with an overall better solution. The 20,300,200FT problem instance exhibits a similar



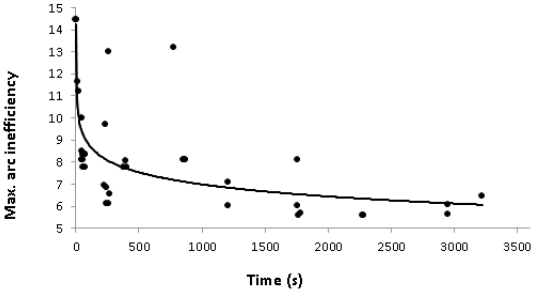
(a) Total cost vs time



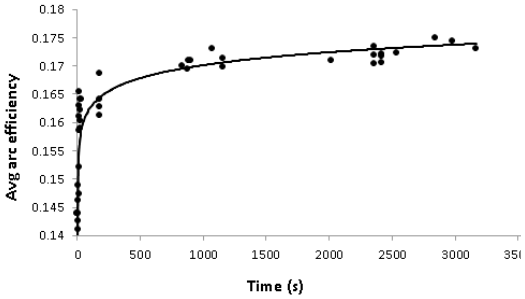
(b) Total cost vs time



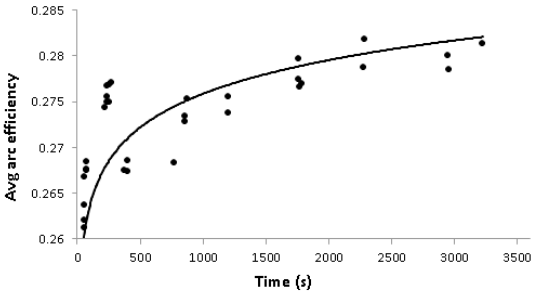
(c) Maximum arc inefficiency vs time



(d) Maximum arc inefficiency vs time



(e) Average arc efficiency vs time



(f) Average arc efficiency vs time

Figure 2: Arc efficiency and total cost tracked over time for instances 20,230,200VT and 30,700,400VL

pattern, i.e., the solvency strategy provides a large improvement in the early generations and then follows a less steep path as the algorithm improves the total cost. The random strategy is again trapped in a local optimum, in this case at the 12^{th} generation. The next two subfigures (c) and (d) display the average solution cost in the Reference Set throughout the generations. The main observations are similar to the first two subfigures; the solvency strategy exhibits a smooth descending path and ends up with a better population in the last generations.

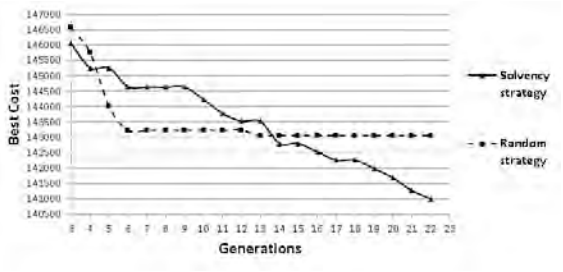
A "healthy" evolutionary process should typically produce a decent number of educated offspring per each generation. Figures 4(e) and 4(f) show this characteristic. In the middle of the evolution process, the random strategy has difficulties in producing educated offspring resulting in premature convergence. On the contrary, the solvency strategy keeps updating the Reference Set with educated offspring until the very last generations, and this is a desirable characteristic of our proposed evolutionary algorithm.

4.5 Ejection Cycles impact on the solution quality

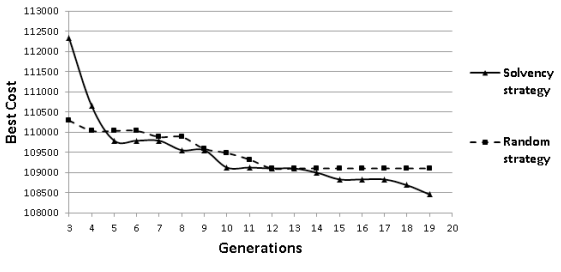
Experimentation was conducted on different versions of the proposed Cycle-based Evolutionary Algorithm (CEA) to investigate the effect of EC on the final solution quality. Two versions of CEA were thus considered. The first is a SS-ILS version, where the local search is iterated with a random perturbation strategy, according to which 25% of the commodities are selected at random, removed and rerouted via the construction mechanism discussed in Section 3.3, and the second is the full version of the CEA algorithm with EC. Table 2 presents the results of these experiments. The letters appended to the instance name are linked with the relative weightings of fixed (F) and variable (V) costs, and of tight (T) and loose (L) capacity constraints. The second column named OPT/LB reports either the optimal (OPT) or the best known lower bound (LB) as identified for each of the instances by Katayama, Chen and Kubo (2009). The entries in the % Dev columns are given by formula $100(v(\text{EC}) - v(\text{SSILS}))/v(\text{EC})$, where $v(\text{SSILS})$ is the cost obtained by using the SS+ILS version of the algorithm, while $v(\text{EC})$ indicates the solution cost obtained by the full version of the algorithm, including EC. It can be easily observed that the impact of the EC in the final solution quality is significant, as it achieves up to a 6.34% improvement in the solution cost.

4.6 Comparative analysis

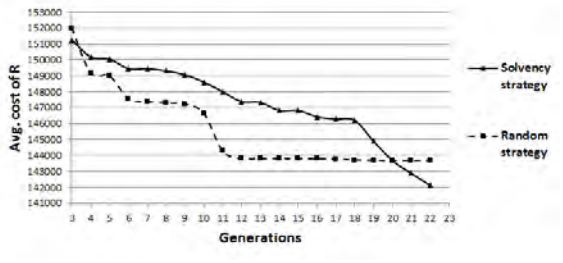
The performance of the proposed algorithm was compared with the solutions reported in Ghamlouche, Crainic and Gendreau (2003), Ghamlouche, Crainic and Gendreau (2004), Crainic, Li and Toulouse (2006), Katayama, Chen and Kubo (2009), Hewitt, Nemhauser and Savelsbergh (2010), Yaghini, Rahbar and Karimi (2012) and Rodrguez-Martin and



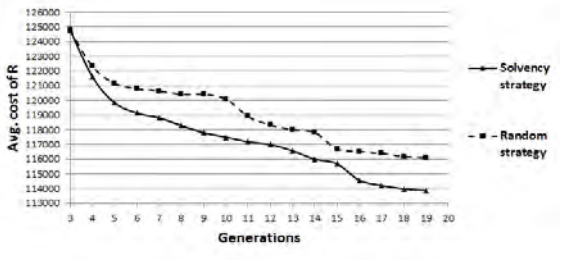
(a) Total cost: 100-400-30FT



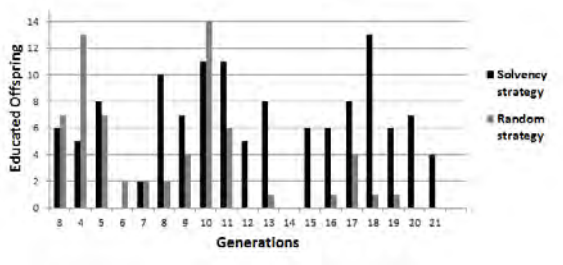
(b) Total cost: 20,300,200FT



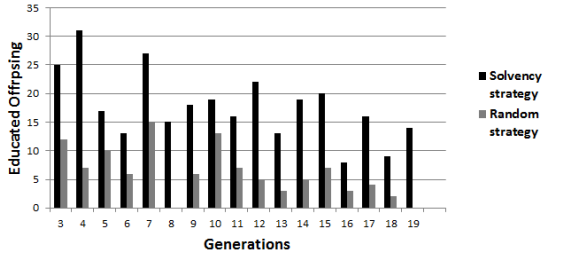
(c) Average cost for R: 100-400-30FT



(d) Average cost for R: 20,300,200FT



(e) Educated offspring: 100-400-30FT



(f) Educated offspring: 20,300,200FT

Figure 3: Solvency-based vs random-based strategies for instances 100-400-30FT and 20,300,200FT

Table 2: Comparative results among different versions of the proposed CEA

Instances	OPT/LB	CEA		% Dev	Instances	OPT/LB	CEA		% Dev
		SS+ILS	+EC				SS+ILS	+EC	
25,100,10VL	14712 O	14712	14712	0.00	100,400,10FL	23949 O	24244	23949	−1.23
25,100,10FL	14941 O	14941	14941	0.00	100,400,10FT	59470 L	70437	66240	−6.34
25,100,10FT	49899 O	49899	49899	0.00	100,400,30VT	384560 L	385916	385163	−0.20
25,100,30VT	365272 O	365272	365272	0.00	100,400,30FL	47459 L	51068	49577	−3.01
25,100,30FL	37055 O	37690	37324	−0.98	100,400,30FT	127825 L	142153	139661	−1.78
25,100,30FT	85530 O	86089	85530	−0.65	30,520,100VL	53958 L	54642	54109	−0.99
20,230,40VL	423848 O	424075	423848	−0.05	30,520,100FL	91285 L	95439	95302	−0.14
20,230,40VT	371475 O	375844	371475	−1.18	30,520,100VT	51825 L	52541	52284	−0.49
20,230,40FT	643036 O	644619	643187	−0.22	30,520,100FT	94646 L	99159	98525	−0.64
20,300,40VL	429398 O	430643	429398	−0.29	30,700,100VL	47603 O	47905	47619	−0.60
20,300,40FL	586077 O	588939	586077	−0.49	30,700,100FL	58772 L	61365	60596	−1.27
20,300,40VT	464509 O	466984	464509	−0.53	30,700,100VT	45552 L	46582	46084	−1.08
20,300,40FT	604198 O	605912	604198	−0.28	30,700,100FT	54233 L	55954	55271	−1.24
20,230,200VL	92598 L	95181	94468	−0.75	30,520,400VL	111992 L	115305	113694	−1.42
20,230,200FL	133512 L	141631	139002	−1.89	30,520,400FL	146809 L	154517	151688	−1.87
20,230,200VT	97344 L	99292	98209	−1.10	30,520,400VT	114237 L	117132	116322	−0.70
20,230,200FT	132432 L	142105	137131	−3.62	30,520,400FT	150009 L	157355	154425	−1.90
20,300,200VL	73759 L	76202	75288	−1.21	30,700,400VL	96741 L	99899	99222	−0.68
20,300,200FL	111655 L	118305	117320	−0.84	30,700,400FL	130724 L	139916	137112	−2.05
20,300,200VT	74991 O	76355	75607	−0.99	30,700,400VT	94118 L	97810	96388	−1.48
20,300,200FT	104334 L	108764	108459	−0.28	30,700,400FT	127666 L	134690	133245	−1.08
100,400,10VL	28423 O	28476	28426	−0.18					

Salazar-Gonzalez (2010). The algorithm of Alvarez, Gonzalez-Velarde and De-Alba (2005) could not be included in the comparisons as the authors do not report any results with the instances tested here.

Table 3 shows the results of the proposed algorithm in comparison with the aforementioned algorithms for solving the MCNDP. The first column of Table 3 shows the name of the instance as characterized by the number of nodes, the number of arcs and the number of commodities. The values of the solutions produced by the proposed algorithm are reported in column CEA. The remaining five columns report the relative percentage deviations of the solutions found by CEA from those of the quoted studies and is calculated as $100(v(\text{CEA}) - v(\text{Alg}))/v(\text{CEA})$, where $v(\text{Alg})$ indicates the solution value produced by the corresponding algorithm and $v(\text{CEA})$ the solution value produced by the proposed CEA. A negative value indicates that the solution found by the CEA is better. The following abbreviations are used:

- CTS: the Cycle-based Tabu Search of Ghamlouche, Crainic and Gendreau (2003);
- PR: the Path Relinking of Ghamlouche, Crainic and Gendreau (2004);
- MCA: the Multilevel Cooperative Algorithm of Crainic, Li and Toulouse (2006);
- CSH: the Capacity Scaling Heuristic of Katayama, Chen and Kubo (2009);
- IPS: the IP Search of Hewitt, Nemhauser and Savelsbergh (2010);

Table 3: Comparison results on the C and C+ sets of benchmark instances of Crainic, Gendreau and Farvolden (2000)

Instances	**OPT/LB**	**CEA**	**% Deviation**				
			CTS	**PR**	**MCA**	**CSH**	**IPS**
25,100,10VL	14712 O	14712	0.00	0.00	0.00	0.00	n/a
25,100,10FL	14941 O	14941	0.00	0.00	0.00	−0.64	n/a
25,100,10FT	49899 O	49899	0.00	0.00	−0.08	−1.75	n/a
25,100,30VT	365272 O	365272	−0.03	−0.03	−0.03	0.00	n/a
25,100,30FL	37055 O	37324	−0.69	−0.88	−0.76	−0.39	n/a
25,100,30FT	85530 O	85530	−0.90	−1.05	−1.09	−0.32	n/a
20,230,40VL	423848 O	423848	−0.22	−0.13	−0.67	−0.05	−0.13
20,230,40VT	371475 O	371475	−0.11	−0.09	0.00	−0.12	−0.08
20,230,40FT	643036 O	643187	−0.41	−0.37	−1.51	−0.20	0.00
20,300,40VL	429398 O	429398	−0.03	0.00	−0.10	0.00	0.00
20,300,40FL	586077 O	586077	−1.24	−0.74	−1.27	−0.29	0.00
20,300,40VT	464509 O	464509	−0.05	0.00	−0.32	−0.01	0.00
20,300,40FT	604198 O	604198	−0.48	−0.96	−2.48	0.00	0.00
20,230,200VL	92598 L	94468	−4.79	−6.28	−4.35	0.23	−0.67
20,230,200FL	133512 L	139002	−5.42	−6.46	−2.98	0.98	−1.62
20,230,200VT	97344 L	98209	−6.66	−6.60	−3.89	0.25	−1.22
20,230,200FT	132432 L	137131	−7.48	−7.60	−2.96	0.73	−2.29
20,300,200VL	73759 L	75288	−7.35	−3.85	−3.88	0.50	−0.04
20,300,200FL	111655 L	117320	−5.14	−5.25	−3.95	1.31	−0.19
20,300,200VT	74991 O	75607	−5.31	−4.31	−2.17	0.40	−0.78
20,300,200FT	104334 L	108459	−5.56	−4.73	−2.50	0.55	−1.74
100,400,10VL	28423 O	28426	−0.88	−0.21	−0.45	0.00	0.01
100,400,10FL	23949 O	23949	0.00	−0.30	−0.30	−2.13	0.00
100,400,10FT	59470 L	66240	−1.17	1.45	−0.07	−11.06	0.54
100,400,30VT	384560 L	385163	−0.09	0.06	−0.03	0.07	0.08
100,400,30FL	47459 L	49577	−3.98	−3.53	−1.77	−4.80	−0.24
100,400,30FT	127825 L	139661	−3.93	−1.22	−4.34	−3.33	−1.22
30,520,100VL	53958 L	54109	−1.57	−1.47	−3.04	0.04	−0.01
30,520,100FL	91285 L	95302	−4.50	−7.08	−4.74	0.53	0.96
30,520,100VT	51825 L	52284	−1.34	−1.40	−2.35	0.00	0.21
30,520,100FT	94646 L	98525	−7.10	−7.72	−4.01	−0.32	−0.36
30,700,100VL	47603 O	47619	−1.64	−2.32	−2.63	−0.03	0.01
30,700,100FL	58772 L	60596	−3.09	−4.12	−5.21	0.66	−0.17
30,700,100VT	45552 L	46084	−2.04	−2.44	−2.98	−0.18	0.08
30,700,100FT	54233 L	55271	−4.73	−2.36	−2.97	−0.16	−0.61
30,520,400VL	111992 L	113694	−6.12	−5.03	−1.74	0.75	−0.31
30,520,400FL	146809 L	151688	−6.20	−7.53	−3.24	1.48	−1.67
30,520,400VT	114237 L	116322	−4.53	−3.31	−4.00	1.45	1.20
30,520,400FT	150009 L	154425	−8.75	−5.99	−3.75	1.09	−0.12
30,700,400VL	96741 L	99222	−7.61	−5.94	−3.44	1.26	0.51
30,700,400FL	130724 L	137112	−8.63	−5.77	−5.01	1.49	−11.28
30,700,400VT	94118 L	96388	−5.48	−5.00	−2.91	1.12	0.23
30,700,400FT	127666 L	133245	−7.15	−5.83	−3.77	2.32	1.21
		MAX	0.00	1.45	0.00	2.32	1.21
		MIN	−8.75	−7.72	−5.21	−11.06	−11.28
		AVG	−3.31	−2.94	−2.27	−0.20	−0.53

- SACG: the Simulated Annealing Column Generation of Yaghini, Rahbar and Karimi (2012);
- LocB: the Local Branching approach of Rodrguez-Martin and Salazar-Gonzalez (2010).

As the results shown in Table 3 indicate, the proposed CEA is competitive relatively to

the state-of-the-art approaches. In particular, CEA is able to produce optimal solutions for the 25,100,10, and 20,300,40 instances as well as for 100,400,10FL. Furthermore, it produced optimal solutions for 25,100,30FT and 20,230,40VL which could not be found by any of the heuristics used for comparisons. Similarly, CEA produced a better solution than the state-of-the-art for the 25,100,30FL problem instance with total cost of 37324.

The performance of CEA as compared with the heuristics that form the basis of our comparison, is very competitive. The maximum improvement provided by CEA as compared to the CTS of Ghamlouche, Crainic and Gendreau (2003) is up to 8.75%, compared to the PR of Ghamlouche, Crainic and Gendreau (2004) is up to 7.72%, compared to the MCA of Crainic, Li and Toulouse (2006) is up to 5.21%, compared to the CSH of Katayama, Chen and Kubo (2009) is up to 11.06%, and finally compared to the IPS of Hewitt, Nemhauser and Savelsbergh (2010) is up to 11.28%. On average, CEA manages to perform better than all above heuristics. With regard to large scale problem instances 100,400,30FT and 30,520,100FT, new best solutions were obtained with values 139661 and 98525, respectively. These instances have up to 100 nodes, 520 arcs and 100 commodities, and the new best solutions improve by 1.22% and 0.32% over the previous best known solutions, respectively.

To be able to conduct objective comparisons with the state-of-the-art, CEA was run and compared to the SACG and the LocB under a time limit of 600 sec and to the IPS under a time limit of 900 sec, while these time limits are taken from the respective papers. The results are shown in Table 4. In particular, as in Table 3 the first column reports the instance name and the second the Optimum/Lower Bound as reported by Katayama, Chen and Kubo (2009). The next multicolumn compares CEA's performance on 600 sec running time to SACG's and LocB's in 600 sec running time. Note that, Yaghini, Rahbar and Karimi (2012) report two sets of results; one derived by 600 sec and one by 18000 sec. The latter column of results was not included here since it could not be compared neither to other studies nor to CEA, given that the results reported by Rodrguez-Martin and Salazar-Gonzalez (2010) are derived with a 600 sec time limit. The final three columns report on test with a 900 sec running time and CEA's performance was compared to IPS's 900 sec results.

As shown in Table 4, CEA remains competitive even when lower running time limits are considered, although as a typical evolutionary algorithm needs a considerable running time to converge and present its best performance. In particular, CEA is able to achieve improvements of up to 13.90%, 20.45% and 9.21% over SACG, LocB and IPS, respectively. On average, CEA is able to produce solutions that are 0.16% better than SACG and 0.89% better than LocB. As for IPS, the performance of CEA is competitive.

Table 5 presents the computation times in seconds reported for the best results obtained by CEA and by the comparator heuristics. The computation times for CTS and PR are for a SUN Enterprise 10000 with 400 MHz 64CPUs (one CPU use), while MCA

Table 4: Comparison results on 600 and 900 sec running time limits

Instances	OPT/LB	600 sec					900 sec		
		CEA	SACG	LocB	% Dev SACG	% Dev LocB	CEA	IPS	% Dev
25,100,10VL	14712 O	14712	14712	14712	0.00	0.00	14712	n/a	n/a
25,100,10FL	14941 O	14941	14941	14941	0.00	0.00	14941	n/a	n/a
25,100,10FT	49899 O	49899	49899	49899	0.00	0.00	49899	n/a	n/a
25,100,30VT	365272 O	365272	365272	365272	0.00	0.00	365272	n/a	n/a
25,100,30FL	37055 O	37522	37055	37326	1.23	0.52	37359	n/a	n/a
25,100,30FT	85530 O	85535	85530	85530	0.01	0.01	85535	n/a	n/a
20,230,40VL	423848 O	424075	423848	423848	0.05	0.05	424075	424385	−0.07
20,230,40VT	371475 O	371475	371475	371475	0.00	0.00	371475	371779	−0.08
20,230,40FT	643036 O	643253	643036	643036	0.03	0.03	643253	643187	0.01
20,300,40VL	429398 O	429398	429398	429398	0.00	0.00	429398	429398	0.00
20,300,40FL	586077 O	586077	586077	586077	0.00	0.00	586077	586077	0.00
20,300,40VT	464509 O	464885	464509	464509	0.06	0.08	464885	464509	0.08
20,300,40FT	604198 O	604198	604198	604198	0.00	0.00	604198	604198	0.00
20,230,200VL	92598 L	95213	95802	95295	−0.09	−0.09	95075	95097	−0.02
20,230,200FL	133512 L	140199	150489	143446	0.31	−2.32	139831	141253	−1.02
20,230,200VT	97344 L	99800	98291	98039	1.82	1.76	98933	99410	−0.48
20,230,200FT	132432 L	140008	149398	141128	−0.54	−0.80	139162	140273	−0.80
20,300,200VL	73759 L	76873	79084	76375	1.13	0.65	75723	75319	0.53
20,300,200FL	111655 L	118559	137387	119142	0.52	−0.49	117838	117543	0.25
20,300,200VT	74991 O	75928	75091	76167	0.67	−0.31	75879	76198	−0.42
20,300,200FT	104334 L	109889	114947	109808	0.16	0.07	109134	110344	−1.11
100,400,10VL	28423 O	28430	28423	28423	0.02	0.02	28430	28423	0.02
100,400,10FL	23949 O	24022	23949	24690	0.30	−2.78	24022	23949	0.30
100,400,10FT	59470 L	67497	65017	67357	3.44	0.21	67497	65885	2.39
100,400,30VT	384560 L	385971	384802	384809	0.30	0.30	385971	384836	0.29
100,400,30FL	47459 L	50567	49945	49872	1.83	1.37	49720	49694	0.05
100,400,30FT	127825 L	142234	141812	141633	0.49	0.42	142234	141365	0.61
30,520,100VL	53958 L	54412	53958	54026	0.79	0.71	54412	54113	0.55
30,520,100FL	91285 L	95829	94524	96255	1.59	−0.44	95495	94388	1.16
30,520,100VT	51825 L	52483	52046	52129	0.18	0.67	52416	52174	0.46
30,520,100FT	94646 L	99635	107807	101102	−0.55	−1.47	99129	98883	0.25
30,700,100VL	47603 O	47809	47603	47603	−0.40	0.43	47736	47612	0.26
30,700,100FL	58772 L	61278	60248	60272	1.14	1.64	61184	60700	0.79
30,700,100VT	45552 L	46264	45880	45905	0.25	0.78	46264	46046	0.47
30,700,100FT	54233 L	55523	54919	55104	0.92	0.75	55523	55609	−0.15
30,520,400VL	111992 L	114805	117052	114367	0.53	0.38	114761	114042	0.63
30,520,400FL	146809 L	154726	177653	157725	−0.72	−1.94	153954	154218	−0.17
30,520,400VT	114237 L	116768	117071	115240	0.99	1.31	116768	114922	1.58
30,520,400FT	150009 L	156263	177653	168561	−2.38	−7.87	156263	154606	1.06
30,700,400VL	96741 L	99914	102094	103787	−1.81	−3.88	99887	98718	1.17
30,700,400FL	130724 L	140934	200005	169759	−13.90	−20.45	139705	152576	−9.21
30,700,400VT	94118 L	97262	97441	96680	0.22	0.60	97106	96168	0.97
30,700,400FT	127666 L	134181	154630	144926	−5.45	−8.01	134181	131629	1.90
				MAX	3.44	1.76			2.39
				MIN	−13.90	−20.45			−9.21
				AVG	−0.16	−0.89			0.06

uses the same machine but takes advantage of the cluster architecture. CSH is run on a Pentium 3.2 GHz CPU, while CEA is run on an i-5 2.5 GHz PC. IPS is not included in this table since the authors do not report specific individual running times, but use a termination criterion of 900 seconds per instance. Ideally in computation times comparisons, appropriate adjustments should be made that account for the speed of the machines. Due to the lack of information regarding the computer architectures, no adjustments have been made in Table 5; the running times are reported as they appear in

Table 5: Computational times (in sec) and comparisons with the state-of-the-art

Instances	CTS	PR	MCA	CSH	CEA	Instances	CTS	PR	MCA	CSH	CEA
25,100,10VL	5.6	48.9	12.5	1.3	6.1	100,400,10FL	33.0	306.8	82.9	93.3	362.6
25,100,10FL	8.4	53.8	14.1	7.1	16.1	100,400,10FT	81.2	626.5	209.9	55.6	2640.9
25,100,10FT	17.1	51.2	24.1	3.7	587.3	100,400,30VT	277.5	1975.3	492.8	17.8	1388.3
25,100,30VT	16.6	223.7	101.4	3.2	74.6	100,400,30FL	100.2	1300.6	315.0	621.5	3445.2
25,100,30FL	33.0	215.4	75.2	14.7	704.6	100,400,30FT	215.7	1870.0	480.9	153.9	2048.4
25,100,30FT	71.8	224.6	97.0	10.0	27.5	30,520,100VL	995.6	3356.0	1194.1	26.9	3505.2
20,230,40VL	71.3	370.3	148.8	3.0	1938.3	30,520,100FL	939.2	4032.4	1460.0	406.5	3403.1
20,230,40VT	90.3	435.6	156.9	3.3	71.9	30,520,100VT	1218.5	3481.1	1513.7	37.3	3536.6
20,230,40FT	121.8	423.3	172.2	3.8	705.2	30,520,100FT	670.3	3927.4	1522.7	199.7	2915.7
20,300,40VL	71.1	611.5	224.9	3.2	66.4	30,700,100VL	1265.1	4396.4	1860.6	38.5	3313.9
20,300,40FL	113.4	581.9	228.3	6.2	726.1	30,700,100FL	1479.6	4755.0	1837.5	114.5	3456.3
20,300,40VT	145.3	589.6	247.9	3.8	3487.6	30,700,100VT	2426.0	4560.1	1894.1	46.3	3601.2
20,300,40FT	123.4	560.4	214.4	4.4	2856.5	30,700,100FT	1735.7	4866.1	1706.1	96.8	2235.7
20,230,200VL	504.5	2663.2	2494.9	442.1	2621.3	30,520,400VL	5789.3	36530.8	27477.4	568.0	3598.1
20,230,200FL	491.6	2718.3	2878.3	1658.0	1421.1	30,520,400FL	6406.6	42929.6	36669.3	2610.4	3423.7
20,230,200VT	548.4	2565.7	2210.9	523.5	2345.5	30,520,400VT	6522.2	28214.0	23089.1	230.0	3212.0
20,230,200FT	889.7	3120.1	3385.8	1943.8	3123.5	30,520,400FT	8415.2	40010.9	52173.2	1673.9	3521.4
20,300,200VL	982.2	4086.8	3566.0	347.7	3576.7	30,700,400VL	12636.2	24816.8	22314.8	474.8	3338.4
20,300,200FL	1316.8	4367.9	4012.6	1289.9	3521.2	30,700,400FL	11367.7	69540.1	75664.9	1782.9	3601.2
20,300,200VT	938.3	3807.9	3924.2	428.7	2290.1	30,700,400VT	15879.5	34974.9	24288.9	749.1	2458.5
20,300,200FT	1065.9	4657.5	3857.1	1721.7	3512.7	30,700,400FT	11660.4	51877.9	44936.4	1487.1	2854.1
100,400,10VL	32.7	336.3	89.2	5.8	65.1						

the original papers. Nevertheless, one could derive some general conclusions regarding the effectiveness of the proposed CEA. In general terms, CEA produces high quality solutions in reasonable computational times. It is worth noting that the computational times of Hewitt, Nemhauser and Savelsbergh (2010) and Katayama, Chen and Kubo (2009) algorithms are very competitive.

5 Conclusions and Further Research

This paper has presented an evolutionary algorithm for the Multi-Commodity Network Design Problem. The proposed methodology evolves a pool of solutions using Scatter Search principles, and an Iterated Local Search as an improvement method. The latter introduces new cycle-based neighbourhood structures, short and long term memory structures for guiding the search, and an efficient perturbation strategy, inspired by Ejection Chains, to enable the search escape from local optima. An efficient recombination strategy is introduced which dynamically adjusts the preferences for inherited solutions based on the search history.

Computational experiments on the benchmark instances of Crainic, Gendreau and Farvolden (2000) show that the proposed CEA is quite competitive compared to state-of-the-art approaches. In particular, CEA is able to reproduce the 12 out of 17 optimum solutions for 17 problem instances, which have previously been solved by exact algorithms. On average, CEA performs better than all the heuristics in the literature, while remaining

competitive when compared to the IPS algorithm by Hewitt, Nemhauser and Savelsbergh (2010). It is also able to produce two new best solutions, with regard to the best known solutions obtained by heuristics, which refer to large-scale problem instances. In general terms, CEA's performance is strong, thus placing it among the most efficient algorithms for solving the MCNDP.

In terms of further research, a promising research direction is the use of a knowledge base where favourable paths for the commodities would be stored not only for speeding up the algorithm but also for guiding the algorithm towards producing unexplored solution structures. Another direction is to look at decomposition techniques to solve the flow subproblems with a view to reducing the computational times. Finally, it is worthwhile to explore the proposed evolutionary algorithm for solving other variants of the MCNDP or even to other problems that share common features with MCNDP.

Acknowledgments

Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant program. With regard to the UK, the funding has been provided by the Engineering and Physical Sciences Research Council (EPSRC), through its Standard Grant Scheme. We also gratefully acknowledge the support of Fonds de recherche du Québec, Canada, through their infrastructure grants. Lastly, this research was completed when the first author was a member of the Centre for Operational Research, Management Science and Information Systems (CORMSIS) and the Schools of Management and Mathematics at the University of Southampton.

References

Alvarez, AM, JL Gonzalez-Velarde, K De-Alba. 2005. Scatter search for network design problem. *Annals Oper. Res.* **138** 159–178.

Balakrishnan, A, TL Magnanti, P Mirchandani. 1997. Network design. F Dell'Amico, M Maffioli, S Martello, eds., *Annotated Bibliographies in Combinatorial Optimization*. John Wiley and Sons, 311–334.

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

Crainic, TG, A Frangioni, B Gendron. 2001. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Appl. Math.* **112** 73–99.

Crainic, TG, M Gendreau. 2002. Cooperative parallel tabu search for capacitated network design. *J. Heur.* **8** 601–627.

Crainic, TG, M Gendreau. 2007. A scatter search heuristic for the fixed-charged capacitated network design problem. Doerner, KF, M Gendreau, P Greistorfer, WJ Gutjahr, RF Hartl and M Reimann, ed., *Metaheuristics - Progress in Complex Systems Optimization*. Springer, 25–40.

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

Feo, TA, MGC Resende. 1995. Greedy Randomized Adaptive Search Procedures. *J. Global Optim.* **6**(2) 109–133.

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

Frangioni, A, B Gendron. 2001. 0–1 Reformulations of the multicommodity network design problem. *Discrete Appl. Math.* **112** 73–99.

Gendron B, TG Crainic, A Frangioni. 1998. Multicommodity capacitated network design. P Soriano, B Sanso, eds., *Telecommunications Network Planning*. Kluwer, 1–19.

Ghamlouche, I, TG Crainic, M Gendreau. 2003. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Oper. Res.* **51** 655–667.

Ghamlouche, I, TG Crainic, M Gendreau. 2004. Path relinking, cycle-based neighborhoods and capacitated multicommodity network design. *Annals Oper. Res.* **131** 109–134.

Glover, F. 1996. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Appl. Math.* **65** 223–253.

Hewitt M, GL Nemhauser, MWP Savelsbergh. 2010. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS J. Comput.* **22**(2) 314–325.

Hewitt M, GL Nemhauser, MWP Savelsbergh. 2012. Branch and price guided search for integer programs with an application to the multicommodity fixed-charge network flow problem. *INFORMS J. Comput.* **Articles in advance** 1–15.

Holmberg, K, D Yuan. 2000. A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Oper. Res.* **48** 461–481.

Katayama, N, M Chen, M Kubo. 2009. A capacity scaling heuristic for the multicommodity capacitated network design problem. *J. Comput. Appl. Math.* **232** 90–101.

Lourenço, HR, O Martin, T Stützle. 2002. Iterated local search. Dorigo M Corne D, Glover F, eds., *Handbook of Metaheuristics: International Series in Operations Research & Management Science*. Kluwer, 321–353.

Magnanti, TL, RT Wong. 1986. Network design and transportation planning: models and algorithms. *Transportation Sci.* **1** 1–55.

Minoux, M. 1986. Network synthesis and optimum network design problems: models, solution methods and applications. *Networks* **19** 313–360.

Paraskevopoulos DC, CD Tarantilis, G Ioannou. 2012. Solving Project Scheduling Problems with Resource Constraints via an Event List-based Evolutionary Algorithm. *Expert Systems Appl.* **39**(4) 3983–3994.

Rodrguez-Martin I, Salazar-Gonzalez JJ. 2010. A local branching heuristic for the capacitated fixed-charge network design problem. *Comput. Oper. Res.* **37**(3) 575–581.

Yaghini M, M Rahbar, M Karimi. 2012. A hybrid simulated annealing and column generation approach for capacitated multicommodity network design. *J. Oper. Res. Soc.* **In press**.