Université de Montréal

Parallel Metaheuristics for Stochastic Capacitated Multicommodity Network Design

par

Xiaorui Fu

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en informatique

Avril, 2008

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé:

Parallel Metaheuristics
for Stochastic Capacitated Multicommodity Network Design

présenté par:

Xiaorui Fu

a été évalué par un jury composé des personnes suivantes:

Jean-Yves Potvin
(président-rapporteur)

Michel Gendreau
(directeur de recherche)

Teodor Gabriel Crainic
(co-directeur)

Bernard Gendron
(membre du jury)

Mémoire accepté le:

Résumé

Les problèmes de synthèse de réseau multiproduits avec coûts fixes et capacité s'appliquent dans différents domaines. À titre d'exemple, citons les problèmes de planification du transport, de la logistique, des télécommunications et de la fabrication. Ces problèmes sont extrêmement difficiles à résoudre. Des métaheuristiques du type recherche avec tabous, "scatter search" et "path relinking", utilisant la structure de voisinage par cycles, ont été élaborées pour résoudre ce problème de façon efficace.

Quand on étudie les problèmes de synthèse de réseau, on suppose habituellement que les demandes entre les paires de nœuds sont connues à l'avance. Or, les demandes futures sont généralement des paramètres inconnus au moment de prendre les décisions. Par conséquent, on doit considérer plutôt un modèle stochastique de synthèse de réseau.

Nous modélisons d'abord les demandes incertaines au moyen d'un ensemble de scénarios. En nous inspirant de la méthode dite de "progressive hedging", nous proposons ensuite un algorithme métaheuristique parallèle qui utilise l'architecture maître/esclave. Les esclaves résolvent de façon répétitive des sous-problèmes modifiés en utilisant la recherche avec tabous basée sur le voisinage par cycles, alors que le maître ramasse les solutions des scénarios et modifie les sous-problèmes en conséquence. Le maître et les esclaves collaborent dans le cadre de la structure métaheuristique jusqu'à ce qu'un certain niveau de saturation soit observé. Par la suite, une deuxième phase est accomplie sur le problème restreint afin d'obtenir les décisions finales.

Dans ce mémoire, nous réalisons d'abord la ré-ingénierie du code existant pour la recherche avec tabous basée sur le voisinage par cycles, puis nous implantons l'algorithme de "progressive hedging" proposé. Les résultats numériques indiquent que l'algorithme proposé est capable de trouver des solutions de bonne qualité. Cependant, le logiciel est conçu et réalisé en gardant à l'esprit la réutilisabilité, l'extensibilité et la maintenabilité.

Mots-clés : synthèse de réseau, programmation stochastique, méthode de recherche avec tabous, calcul parallèle, voisinage par cycles.

Abstract

Fixed-charge capacitated multicommodity network design (CMND) formulations have a wide range of applications in transportation, logistics, telecommunication and production planning. Metaheuristics such as tabu search, scatter search and path-relinking have been tailored to solve this problem using cycle-based neighborhood structures.

When studying network design problems it is customary to assume that point-to-point demands are given. However, future demands are generally unknown at the time when the design decisions are made. One should therefore consider the stochastic network design (SND) model instead.

We model the uncertain demands through scenario analysis, then propose a progressive hedging-inspired parallel metaheuristic algorithm using master/slave architecture. The slaves use the cycle-based tabu search algorithm to solve modified subproblems iteratively, while the master collects scenario solutions and modifies subproblems accordingly. Master and slaves cooperate under this metaheuristic framework until some level of saturation is observed, then a second phase is performed on the restricted problem in order to obtain the final design decisions.

We first performed a software re-engineering of the existing code of the cycle-based tabu search algorithm, and then implemented the proposed progressive hedging algorithm. Experimental results show that high quality designs can be obtained by the proposed algorithm. At the same time, the software is designed and implemented keeping in mind its re-usability, extensibility and mantainabil-

ity.

Key words: network design, stochastic programming, tabu search, parallel programming, cycle-based neighborhood.

Contents

List of Figures

List of Tables

List of Abrevations

CMCF	:	capacitated multicommodity minimum cost flow problem
CMND	:	capacitated multicommodity network design problem
EV	:	expected value solution
EEV	:	expected result of using the EV solution
EVPI	:	expected value of perfect information
LP	:	linear programing
MIP	:	mixed integer programing
MPI	:	message passing interface
PHA	:	progressive hedging algorithm
RP	:	recourse problem
SIP	:	stochastic integer programming
SND	:	stochastic network design problem
TS	:	tabu search
VSS	:	value of stochastic solution
WS	:	wait and see solution

List of Algorithms

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Michel Gendreau, and my co-supervisor, Teodor Gabriel Crainic, for their continuous inspiration, expertise and guidance, for having accepted me in their research group, for having given me the great opportunity to pursue my study in a famous research centre. With their support, this study has been an invaluable learning experience for me.

I would like to express my honest gratitude to Walter Rei, for his constant support, encouragement and patience. His help makes my integration to Canadian society much easier. His remarks and recommendations on my thesis have been invaluable.

I am also indebted to the professional staff and technical personnel of the former CRT (Centre de Recherche sur les Transports), actually CIRRELT (Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport), where I had the privilege to carry out all my research work. I would like to thank particularly François Guertin, Pierre Girard, Daniel Charbonneau, Serge Bisaillon, Geneviève Hernu, Luc Rocheleau and Denise Desjardins, for their generous help within their area of expertise.

I would like to thank Ilfat Ghamlouche, although I did not have the chance to contact her in person, for all previous research work she completed, which is an essential basis of my thesis. I would like to thank Stein William Wallace and Michal Kaut for generously sending us their scenario generation program and giving us detailed instructions, which makes our numerical experiments more

meaningful.

I would like to thank all my friends for the moral support that they always demonstrated, in the happy days as well as in the sad moments. I would like to thank Nadia Lahrichi for her help on LaTeX, Massimo Di Francesco for his help on mathematical models, Michel Toulouse for his listening and encouragement.

Finally, I would like to express my deepest gratitude to my husband Zhenfeng Li, for his love, understanding and support. Without him, I would not have accomplished my graduate studies.

Chapter 1

Introduction

The fixed-charge capacitated multicommodity network design (CMND) formulation represents a generic model that arises in various applications in telecommunication, transportation, logistic and production planning [5, 16, 12, 18, 34, 35]. In these applications, it is required to route multiple commodities over a given network with arc capacities in order to satisfy known demands between origin-destination pairs. In doing so, one pays not only a routing cost proportional to the number of units of each commodity transported over a network link, but also the fixed cost representing the construction or improvement of a link if it is used. The objective of CMND is to find the optimal design (selected links in the final network) that minimizes the total cost, computed as the sum of the fixed and routing costs.

Fixed-charge network design problems belong to the NP-hard complexity class [34], and capacitated ones are particularly difficult [4]. Not only is finding the optimal solution to a large problem instance a significant challenge, but even identifying efficiently good feasible solutions might be a difficult task. As a consequence, only specially tailored heuristics have been proven to be of any help.

In various heuristic or metaheuristic methods that search for good designs in the solution space, the neighborhood structure plays an important role in the efficiency of the search mechanisms. The cycle-based neighborhood proposed by Ghamlouche, Crainic and Gendreau [21] defines moves that deviate flow around

cycles which close and open arcs accordingly. Such a neighborhood is quite powerful since it modifies the flow distribution of several commodities simultaneously and takes explicitly into account the impact on the total cost. In [21], the authors tested this cycle-based neighborhood within a simple tabu search algorithm and got interesting numerical results.

In CMND models, commodities usually correspond to origin-destination pairs with given demands. Generally speaking, this is unsuitable in practice, since future demands are usually unknown at the time when the design decisions are made. In order to deal with the uncertainty of demands, we consider stochastic network design (SND) models, which can be formulated as two-stage stochastic integer programming problems. A common approach to address such problems is through scenario analysis, which represents future events as possible alternative scenarios. Then the question becomes how to work with different scenario solutions and consolidate them into an overall decision scheme.

The progressive hedging algorithm (PHA) of Rockafellar and Wets [37] is a general method for solving linear continuous stochastic programs. The idea behind PHA is that by solving individual scenario problems and insisting progressively more and more on the requirement that the solutions generated by the scenario problems must be implementable, one may identify "trends" and eventually come up with a "well hedged" solution. The algorithm is intuitively appealing and has been proven to converge to a global optimum in the linear stochastic case.

Inspired by PHA, our research strategy is to take PHA as a metaheuristic framework, and make use of the special structure of the network design problem. We propose a parallel metaheuristic algorithm using master/slave architecture to solve SND problems. First, several slaves solve in parallel the deterministic network design problem for different scenarios using the cycle-based tabu search

algorithm. The design vectors are then sent back to the master. The master modifies the subproblems by penalizing more and more the inconsistency of scenario solutions. The master also constructs and broadcasts a new design vector that is used later by slaves to guide their tabu search. This information exchange and cooperation continues until some level of saturation is observed. Finally, a second phase is performed on the restricted problem to achieve final design decisions. We propose two ways to modify the objective function of the subproblems in the PHA framework, one is an Augmented Lagrangian method and the other is a heuristic fixed cost adjustment.

In order to solve the scenario problems, we first performed a software re-engineering of existing code of the cycle-based tabu search algorithm. We tested the re-implemented software on the same instances than [21] and got comparable results. Then, we implemented the proposed parallel metaheuristic algorithm for the stochastic problem, and conducted experiments on two instance sets. Numerical results show that high quality designs can be achieved by the proposed algorithm. Meanwhile, the software is designed and implemented by keeping in mind its reusability, extensibility and maintainability. Design and implementation details are also reported in this thesis for further reference.

This thesis is organized as follows. The mathematical formulation and related works of CMND and SND are introduced in Chapter 2. The design and implementation details of the cycle-based tabu search algorithm, as well as the numerical results, are reported in Chapter 3. Chapter 4 presents the proposed parallel metaheuristic algorithm, whose design and implementation details are given in Chapter 5. Computational results and analysis for the SND problems follow in Chapter 6. Conclusions and suggestions for future work are outlined in Chapter 7.

Chapter 2

Fixed-charge capacitated multicommodity network design

This chapter consists of two sections. In the first section, we present the arc-based mathematical formulation of the fixed-charge capacitated multicommodity network design problem (CMND) and related works. In the second section, we examine the stochastic network design formulations, analyze the value of the stochastic model and review the related research in literature. We conclude this chapter with our methodological approach.

2.1 Deterministic network design

The CMND problem has a wide range of applications in telecommunication, transportation and logistics. In these applications, multiple commodities are required to be routed over a capacitated network. In addition to the transportation cost, one pays a construction or improvement cost for the first time a link is used. The goal is to find the design that minimizes the total cost, computed as the sum of the fixed and routing costs.

2.1.1 Mathematical formulation

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a network with a node set $\mathcal{N}$ and a directed design arc set $\mathcal{A}$. Let $\mathcal{K}$ denote the set of commodities to be routed using this network. Each commodity $k \in \mathcal{K}$ has a single origin $o(k)$, a single destination $s(k)$, and a given demand w^k. Let i and j be the node indices. The arc-based formulation of the CMND can then be written as follows:

$$\min z(y, x) = \sum_{(i,j)\in\mathcal{A}} f_{ij} y_{ij} + \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^k \tag{2.1a}$$

subject to:

$$\sum_{j\in\mathcal{N}^+(i)} x_{ij}^k - \sum_{j\in\mathcal{N}^-(i)} x_{ji}^k = d_i^k \qquad \forall i \in \mathcal{N}, \forall k \in \mathcal{K} \tag{2.1b}$$

$$\sum_{k\in\mathcal{K}} x_{ij}^k \leq u_{ij} y_{ij} \qquad \forall (i,j) \in \mathcal{A} \tag{2.1c}$$

$$y_{ij} \in \{0, 1\} \qquad \forall (i,j) \in \mathcal{A} \tag{2.1d}$$

$$x_{ij}^k \geq 0 \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K} \tag{2.1e}$$

where y_{ij} is a binary variable representing whether or not arc (i, j) is selected in the final design, x_{ij}^k is a non-negative continuous variable standing for the flow amount of commodity k on arc (i, j). The constant parameters in model (2.1)

are :

f_{ij} : fixed cost of arc (i, j)

c_{ij}^k : routing cost of one unit flow of commodity k on arc (i, j)

u_{ij} : capacity of arc (i, j)

$$d_i^k = \begin{cases} w^k & \text{if } i = o(k) \\ -w^k & \text{if } i = s(k) \\ 0 & \text{otherwise.} \end{cases}$$

The objective function (2.1a) accounts for the total system cost, including the fixed costs for the arcs selected in a given design and the routing costs for the distribution of commodities. As for the constraints, Equations (2.1b) represent the network flow conservation relations, requiring that the demands on all nodes be satisfied, where $\mathcal{N}^+(i)$ and $\mathcal{N}^-(i)$ are the set of outward and inward neighbors of node i, respectively. Equations (2.1c) are called arc capacity constraint, indicating that the total amount of flow for all commodities on an arc should not exceed its capacity if this arc is opened ($y_{ij} = 1$), and must be zero if this arc is closed ($y_{ij} = 0$). Equations (2.1d) and (2.1e) are the integrality and non-negativity constraints.

Recall that, for a given design vector $\hat{y}$ (an assignment of 0 and 1 to each design variable), the above CMND model becomes a capacitated multicommodity minimum cost flow problem (CMCF) whose formulation is given below:

$$\min\ z(x(\hat{y})) = \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^k \tag{2.2a}$$

subject to (2.1b) plus:

$$\sum_{k\in\mathcal{K}} x_{ij}^k \leq u_{ij}\hat{y}_{ij} \qquad \forall (i,j) \in \mathcal{A}(\hat{y}) \tag{2.2b}$$

$$x_{ij}^k \geq 0 \qquad \forall (i,j) \in \mathcal{A}(\hat{y}), \forall k \in \mathcal{K} \tag{2.2c}$$

where $\mathcal{A}(\hat{y})$ represents the set of arcs that are opened in the design vector $\hat{y}$. A feasible solution to model (2.1) thus can be viewed as $(\hat{y}, x^*(\hat{y}))$, where $x^*(\hat{y})$ stands for the optimal solution of model (2.2), i.e., the optimal flow distribution of the corresponding CMCF problem. The objective function value associated to a solution $(\hat{y}, x^*(\hat{y}))$ therefore can be calculated as the sum of the fixed costs for the arcs that are opened in $\hat{y}$ plus the optimal objective function value of (2.2):

$$z(\hat{y}, x^*(\hat{y})) = \sum_{(i,j)\in\mathcal{A}(\hat{y})} f_{ij}\hat{y}_{ij} + z(x^*(\hat{y})).$$

It should be noted here before calculating the total cost $z(\hat{y}, x^*(\hat{y}))$, a trim procedure should be performed, which closes the arcs that do not carry any flow, i.e., set $\hat{y}_{ij} = 0$ if $\sum_{k\in\mathcal{K}} x_{ij}^k = 0$.

Finally, in addition to the arc-based model, a number of applications might be modeled by using the path-based formulation [5, 14]. In this thesis, we concentrate on the arc-based formulation.

2.1.2 Addressing the CMND model

Some exact methods for CMND problems have been developed, such as the simplex-based cutting plane algorithm [11], the bundle-based relaxation method [13], the Lagrangian relaxation algorithm [18], the Lagrangian heuristic based branch-and-bound approach [24], and some combinations of the cutting, relaxing and variable fixing approaches [29, 38]. However, these methods can only solve

moderately sized instances. For reasonably large instances, only heuristics have proven to be effective.

Crainic, Gendreau and Farvolden [14] proposed a tabu search meta-heuristic in 2000 for the path-based formulation of the problem. This method explores the space of the path-flow variables by using pivot-like moves and column generation. Even though it produced very impressive results, its search efficiency may be limited since each move considers only the impact of changing the flow of one commodity. Moreover, there is no guarantee that all paths will be considered, which, in turn, impairs the capability of the algorithm to explore thoroughly the search space.

In 2003, Ghamlouche, Crainic and Gendreau [21] proposed a cycle-based neighborhood for the arc-based formulation. In this neighborhood structure, moves are defined as re-deviation of the flow around cycles and for which arcs are closed and opened accordingly. This neighborhood is quite powerful since it modifies the flow distribution of several commodities simultaneously and takes into account the impact on the total cost explicitly. The authors tested this neighborhood in a simple tabu search procedure. Numerical results showed that this approach appeared as the best known approximate solution method at that time in terms of robust performance, solution quality and computing efficiency. However, given the simplicity of the tabu search procedure used in [21], this approach seemed to produce a rather local exploration of the search space.

A path relinking procedure is proposed in [20] in order to refine the search method in [21]. The authors combined the path relinking framework and the cycle-based neighborhood structure into a meta-heuristic for the CMND problem. They compared several strategies to both build the reference set and to select initial and guiding solutions. Experimental results showed that the proposed approach offered the best performance among approximate solution meth-

ods for the CMND problem. Moreover, the algorithm appeared robust in terms of solution quality and computational effort.

In 2004, Crainic, Gendron and Hernu [15] proposed a slope scaling heuristic that integrates a Lagrangian perturbation scheme and intensification/diversification mechanisms based on a long-term memory. Computational results showed that the proposed method is competitive with the best known heuristic approaches for the problem. Moreover, it generally provides better solutions on larger, more difficult instances.

2.2 Stochastic network design

In the CMND models, the design parameters are assumed to be deterministic. Unfortunately, critical parameters such as demands, costs and capacities are quite uncertain. Future demands are usually unknown at the time when the design decisions are made. Moreover, the construction or improvement cost of an arc may be underestimated or overestimated at the planning time, the routing cost of delivering some commodities over an arc may change due to some unforeseen reason, and some ad-hoc events may cause the decrease of the capacity for an arc, or even make the arc totally unusable. In order to model these kinds of stochasticity, one should consider the stochastic network design (SND) model instead.

Among all the uncertainties mentioned above, uncertain demand is often considered in the first place in the literature [3]. Within this thesis, we address only the demand uncertainty of stochastic network design problem. Applications can be found in many fields such as telecommunication, hub location, transportation and production system.

2.2.1 Mathematical formulations

Two-stage stochastic model

To capture the stochastic feature of the demands in the CMND problem, we extend the deterministic model to a two-stage stochastic model. The network design decisions are made in the first stage while future demands are unknown. In the second stage, the stochastic demands become known, then the flow distribution decisions can be made. The decisions that are made in the second stage can be viewed as a recourse action, which introduces a recourse cost to the objective function. The recourse cost is defined here as the expectation of the routing costs over all demand scenarios.

Most of the notations in the following models are from the book of Birge and Louveaux [6]. We made a slight change in order to be coherent with the notation used for the network design problem. Using the matrix form, the SND model can be stated as follows:

$$\min\ z(y,\xi) = f^T y + E_\xi[\min\ c^T x(y,\omega)] \tag{2.3a}$$

subject to:

$$W_1 x(y,\omega) = d(\omega) \tag{2.3b}$$

$$Ty + W_2 x(y,\omega) \leq 0 \tag{2.3c}$$

$$y \in \{0,1\}^{|\mathcal{A}|} \tag{2.3d}$$

$$x(y,\omega) \geq 0. \tag{2.3e}$$

where y represents the first stage variable vector containing $|\mathcal{A}|$ binary design variables, ξ is the random variable vector for which $\omega \in \Omega$ defines a particular realization, x stands for the second stage variable vector containing $|\mathcal{A}||\mathcal{K}|$ non-

negative continuous flow variables, which depends both on the network design decision y and the random demand realization ω. f and c are constant coefficients representing respectively the fixed costs and the routing costs.

The objective function (2.3a) accounts for the total system cost, including the fixed costs for the opened arcs and the expectation of the routing costs taken over all realizations of the random demands.

Equations (2.3b) represent the network flow conservation relations, where W_1 is a matrix of size $|\mathcal{N}||\mathcal{K}| \times |\mathcal{A}||\mathcal{K}|$, which includes $|\mathcal{K}|$ node-arc incidence matrices of network $\mathcal{G}$ on its diagonal and where all other entries are zeros, and $d(\omega)$ is a column vector of size $|\mathcal{N}||\mathcal{K}|$ with entries $d_i^k(\omega)$ defined as

$$d_i^k(\omega) = \begin{cases} w^k(\omega) & \text{if } i = o(k) \\ -w^k(\omega) & \text{if } i = s(k) \\ 0 & \text{otherwise.} \end{cases}$$

Equations (2.3c) are the arc capacity constraints, where T is a diagonal matrix of size $|\mathcal{A}| \times |\mathcal{A}|$ with $-u_{ij}$ on its diagonal, W_2 is a matrix of size $|\mathcal{A}| \times |\mathcal{A}||\mathcal{K}|$ which includes K identity matrices of size $|\mathcal{A}| \times |\mathcal{A}|$. Constraints (2.3c) are also called linking constraints, since they involve both the first and second stage variables.

It should be noted that in model (2.3), the first stage variables are all required to be binary, while the second stage ones are all continuous. The only constraints in the first stage are the integrality constraints (Equation 2.3d). In the second stage, the objective coefficients c, recourse matrices W_1 and W_2, and the matrix T that links the first and second stage are all fixed.

Scenario model

In many situations, only limited statistical information about future demands is available. Even when a probabilistic description of the unknown demands is at hand, it is usually of doubtful quality. Besides, the associated optimization problem can be notoriously hard to solve. A common approach to deal with this situation is scenario analysis, where the uncertain parameters are modeled by a number of representative scenarios. In the SND problem we addressed, each scenario stands for a possible realization of the random demands. Given a set of scenarios $\mathcal{S}$ with index s, the stochastic model (2.3) can be rewritten as follows:

$$\min \sum_{(i,j)\in\mathcal{A}} f_{ij}y_{ij} + \sum_{s\in\mathcal{S}} p^s \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^{ks} \tag{2.4a}$$

subject to:

$$\sum_{j\in\mathcal{N}^+(i)} x_{ij}^{ks} - \sum_{j\in\mathcal{N}^-(i)} x_{ji}^{ks} = d_i^{ks} \qquad \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \forall s \in \mathcal{S} \tag{2.4b}$$

$$\sum_{k\in\mathcal{K}} x_{ij}^{ks} \le u_{ij}y_{ij} \qquad \forall (i,j) \in \mathcal{A}, \forall s \in \mathcal{S} \tag{2.4c}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A} \tag{2.4d}$$

$$x_{ij}^{ks} \ge 0 \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K}, \forall s \in \mathcal{S} \tag{2.4e}$$

where p^s is the probability of scenario s ($p^s \le 1$, $\sum_{s\in\mathcal{S}} p^s = 1$), x_{ij}^{ks} represents the flow amount of commodity k on arc (i,j) if scenario s is observed, and d_i^{ks} is defined as

$$d_i^{ks} = \begin{cases} w^{ks} & \text{if } i = o(k) \\ -w^{ks} & \text{if } i = s(k) \\ 0 & \text{otherwise} \end{cases}$$

for each scenario s.

By modeling uncertainty through scenarios, problem (2.4) becomes a deterministic mixed integer linear program, for which different solution methods have been proposed in the literature. It should be noted that the CMND problem in itself is already NP-hard in the strong sense. The stochastic version, which includes multiple demands scenarios, is even more difficult to solve, since the number of variables and constraints are multiplied by $|\mathcal{S}|$. As a consequence, for SND problems of any interesting size, decomposition methods are of great interest.

Scenario decomposition model

The block structure in model (2.4) naturally leads us to consider scenario decomposition approaches. By introducing S copies of the first stage decision variables y, model (2.4) can be rewritten as

$$\min \sum_{s\in\mathcal{S}} p^s \Big(\sum_{(i,j)\in\mathcal{A}} f_{ij} y^s_{ij} + \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c^k_{ij} x^{ks}_{ij} \Big) \tag{2.5a}$$

subject to (2.4b), (2.4e) plus:

$$\sum_{k\in\mathcal{K}} x^{ks}_{ij} \le u_{ij} y^s_{ij} \qquad \forall (i,j)\in\mathcal{A}, \forall s\in\mathcal{S} \tag{2.5b}$$

$$y^s_{ij} \in \{0,1\} \qquad \forall (i,j)\in\mathcal{A}, \forall s\in\mathcal{S} \tag{2.5c}$$

$$y^s_{ij} = y^t_{ij} \qquad \forall (i,j)\in\mathcal{A}, \forall s,t\in\mathcal{S}, s\neq t \tag{2.5d}$$

where y^s_{ij} is the design variable for arc (i,j) in scenario s, and relations (2.5d) are called the non-anticipativity constraints. Constraints (2.5d) make sure that

the solution of (2.5) is implementable, i.e., the network design decisions are taken globally and do not depend on particular scenarios.

Model (2.5) is equivalent to (2.4). The only difference is the number of variables and constraints. It should be noted that model (2.5) has a desirable property: if the non-anticipativity constraints (2.5d) are relaxed, the problem becomes scenario-separable. There are several ways to rewrite then relax these constraints. We will discuss this in detail in Section 4.2.

2.2.2 Value of the stochastic model

The CMND problem is NP-hard in the strong sense, and the SND problem is even harder, since it combines the difficulty of stochastic and integer programming. There should be a good incentive to consider the stochastic version instead of assigning "reasonable" values to the unknown demands. In this section, we present two ways to measure the value of uncertainty for the network design problem. Through this analysis, we define both lower bounds for model (2.5), as well as upper bounds, from which some valuable initial solutions may be obtained. We refer to [6] for a complete introduction of the concepts presented in this section.

Expected value of perfect information

The concept of expected value of perfect information (EVPI) was first developed in the context of decision analysis [2]. It measures the potential value of having perfect (complete and accurate) information about the future. In other words, it represents the maximum amount a decision maker would be willing to pay to be able to predict with certainty which scenario will be observed prior to making a decision.

In the stochastic programming setting, EVPI is defined as the difference be-

tween the wait-and-see solution and the here-and-now solution [6]. Models (2.3), (2.4), (2.5) are recourse problems (RP) whose solutions are the so-called here-and-now solutions, where decisions must be taken before the demand realization is known. In turn, if one could predict with certainty which of the scenarios will be observed, the problem to be solved, if scenario s is to occur, is the CMND problem defined as follows:

$$\min \sum_{(i,j)\in\mathcal{A}} f_{ij}y_{ij}^{s} + \sum_{k\in\mathcal{K}}\sum_{(i,j)\in\mathcal{A}} c_{ij}^{k}x_{ij}^{ks} \tag{2.6a}$$

subject to:

$$\sum_{j\in\mathcal{N}^{+}(i)} x_{ij}^{ks} - \sum_{j\in\mathcal{N}^{-}(i)} x_{ji}^{ks} = d_{i}^{ks} \qquad \forall i \in \mathcal{N}, \forall k \in \mathcal{K} \tag{2.6b}$$

$$\sum_{k\in\mathcal{K}} x_{ij}^{ks} \leq u_{ij}y_{ij}^{s} \qquad \forall (i,j) \in \mathcal{A} \tag{2.6c}$$

$$y_{ij}^{s} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A} \tag{2.6d}$$

$$x_{ij}^{ks} \geq 0 \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K}. \tag{2.6e}$$

Model (2.6) will be refered to as a *subproblem*, or a *scenario problem* for scenario s. Without loss of generality, we can assume that there exists at least one feasible solution for each subproblem, which, in turn, implies the existence of at least one optimal solution for the subproblem. Let y^{s*} denote the optimal design of (2.6) and $z(y^{s*})$ the related optimal objective function value. The wait-and-see solution (WS) can then be defined as the expected value of the optimal

solutions taken over all scenario problems:

$$WS = \sum_{s \in \mathcal{S}} p^s z(y^{s*}).$$

Here follows the definition of $EVPI$,

$$EVPI = RP - WS$$

where RP is the optimal objective function value of the recourse problem. The nonnegative property of $EVPI$ makes WS a lower bound for the SND problem. In fact, this is quite intuitive, since for each scenario s, we have

$$z(y^{s*}) \leq z(y^*), \tag{2.7}$$

where y^{s*} represents the optimal solution of the scenario problem, y^* stands for the optimal solution to the recourse problem, and $z(\cdot)$ is the related objective function value.

Equation (2.7) can be explained as follows: Given scenario s, the scenario solution y^{s*} is the best design which can be found. However, this scenario solution may turn out to be poor (or even infeasible) for a different scenario. On the other hand, the stochastic optimal solution (y^*), which may be inferior in quality to any scenario solution (y^{s*}) once the realization of the random event is known, provides a good compromise among all possible scenarios. For the SND problem addressed within this thesis, we aim at high quality, if not optimal, stochastic solutions.

It should be noted that the wait-and-see solution is not a feasible solution for the recourse problem. It provides only a lower bound. The procedure of finding this lower bound can be viewed as an approach of relaxing the non-anticipativity

constraints. If at least one feasible solution is found for each scenario problem, a global design can be constructed by opening all arcs that carry flow in at least one scenario solution. An upper bound can then be derived from this global design, which can also be used as an initial solution for various heuristic search strategies.

For the SND problem, the computation of WS requires solving each CMND to optimality, which rapidly becomes very expensive whenever the network size and the number of commodities become too large. For the instances whose subproblems cannot be solved to optimality within a reasonable time limit, their lower bounds can be obtained. Denote $\underline{z}(y^s)$ the lower bound of the subproblem, the lower bound of wait-and-see solution is then defined as

$$\underline{WS} = \sum_{s \in \mathcal{S}} p^s \underline{z}(y^s),$$

which can be used in place of WS to estimate the EVPI value.

Value of stochastic solution

For practical purposes, finding the wait-and-see solution is sometimes too difficult. On the one hand, solving to optimality each scenario problem is computationally expensive; on the other hand, the wait-and-see approach often delivers a set of solutions that conflict with each other, while what we are looking for is an implementable global design.

To obtain a global design, a natural temptation is to solve the corresponding expected value (EV) problem obtained by replacing the random demands with their expected values. The EV problem can be expressed as

$$EV = \min\, z(y, \bar{\xi}),$$

where $\overline{\xi} = E(\xi)$ represents the expectation of random demands ξ taken over all scenarios. Let us denote by $\hat{y}$ the optimal solution to EV problem. Using $\hat{y}$ as the global design, one can solve for each scenario s the second-stage CMCF problem, then get the *expected result of using the EV solution*:

$$EEV = \sum_{(i,j)\in\mathcal{A}} f_{ij}\hat{y}_{ij} + \sum_{s\in\mathcal{S}} p^s[\min\, z(x(\hat{y},\xi))],$$

where $\min\, z(x(\hat{y},\xi))$ is the related CMCF problem using design $\hat{y}$. It should be noted here a trim procedure should be performed before calculating EEV, which closes the arcs that do not carry any flow in any scenario.

The value of stochastic solution (VSS) is then defined as

$$VSS = EEV - RP,$$

which precisely measures how good or, more frequently, how bad the decision $\hat{y}$ is in terms of the recourse problem. This measure can also be interpreted as the cost of ignoring uncertainty, or, the gain in using the stochastic model.

The nonnegative property of VSS makes EEV an upper bound of the SND problem. It should be noted that the EV solution $\hat{y}$ is not always feasible for the associated CMCF problems. In fact, in the context of the network design problem, if there is some kind of backup transportation methods that can be used when $\hat{y}$ turned out to be infeasible for some scenarios, the EV solution is guaranteed to be feasible. On the contrary, if there is no available backup transportation method, an infeasible EV solution $\hat{y}$ will lead to an infinite EEV, thus an infinite VSS.

For all instances where both $EVPI$ and VSS are equal to zero, it is unnecessary to solve a recourse problem. In such situations, the optimal solutions are insensitive to the realization of the random event, thus finding the optimal solu-

tion for one scenario would yield the same result as solving the recourse problem. The stochastic network design model should only be solved on those instances where the EVPI or VSS are large.

2.2.3 Related works

The stochastic network design problem belongs to the group of stochastic integer programming (SIP) problems, which are challenging from both computational and theoretical point of views. They combine the difficulty associated with stochastic programming and integer programming.

In the situations where it is reasonable to assume that the random variables are all discretely distributed and that their joint distribution has a finite number of possible realizations, scenario analysis can be used. Otherwise, a careful discretization procedure is needed to carry on the scenario analysis.

Given a set of scenarios $\mathcal{S}$, the stochastic model becomes a large-scale, dual block-angular mixed-integer problem, as in the equations (2.4) and (2.5). Decomposition is a natural way to solve this large-scale formulation.

Decomposition methods for stochastic integer programs generally fall into two groups: primal methods that work with subproblems assigned to time stages and dual methods that work with subproblems assigned to scenarios [8]. The L-shaped procedure in [9, 31, 36] belongs to the first group. According to [9], however, the master problem obtained from this kind of decomposition is not, in general, computationally attractive.

As for the second group, Carøe and Schultz [8] proposed a branch and bound algorithm that uses Lagrangian relaxation of the non-anticipativity constraints as bounding procedure. Rounding heuristics are applied on the average over all scenario solutions in order to fulfill the integrality restrictions. The numerical

results show that this algorithm can provide very good feasible solutions (within 0.2% of the optimun). However, the instances used in this paper have very low VSS values (less than 0.8%), which means that randomness has little influence on the optimal first-stage solution. This is generally not the case for the stochastic network design problems.

Rockafellar and Wets proposed a progressive hedging algorithm (PHA) in [37] for solving stochastic programs. This progressive hedging algorithm will generate a sequence of estimates of the optimal solution, obtained by insisting more and more on the requirement that the solutions generated by the scenario problems be globally implementable. The algorithm is intuitively appealing and has the desirable property of converging to a global optimum in the linear stochastic case.

Løkketangen and Woodruff [33] introduced a general purpose algorithm for finding good solutions to mixed integer multistage stochastic programming problems. In their algorithm, the progressive hedging algorithm in [37] was used as a framework to blend scenario solutions. Tabu search was used to solve the induced quadratic mixed-integer subproblems, which consists of the deterministic function and the Lagrangian terms that penalize the lack of global implementability. Computational experiments on a family of problems with 3, 5, and 10 scenarios verified that the proposed algorithm is effective in finding good solutions.

2.2.4 Problem addressed and methodological approach

Inspired by the progressive hedging algorithm [37], our research strategy is to combine the progressive hedging framework and the cycle-based tabu search algorithm into a metaheuristic algorithm. We aim at an algorithm that is able to find good solutions efficiently for the stochastic network design problem with uncertain demands.

The challenge is twofold. First, the difficulty inherited from the CMND problem makes it a formidable task to identify good feasible solutions for the stochastic network design problem. Second, it is hard to make a tradeoff between the effort put in finding good scenario solutions and that put in consolidating the scenario solutions into one global design.

The implementation of the proposed algorithm is performed in two steps. The first step is to do the software re-engineering work on the existing research code of the cycle-based tabu search algorithm proposed in [21], so that it can be efficiently used in various contexts. Moreover, we also established a good basis for future software evolution. Chapter 3 reports on the re-engineering work for the cycle-based tabu search algorithm. In the second step, we implement the progressive hedging algorithm using a master/slave architecture, and reuse the cycle-based tabu search algorithm to solve the scenario problems. The algorithm is proposed in Chapter 4, while the implementation and the experimental results are reported in Chapters 5 and 6, respectively. We summarize our contribution and point out future research avenues in Chapter 7.

Chapter 3

Re-engineering of the cycle-based tabu search for deterministic network design

This chapter can be viewed as a technical report of the software re-engineering work to the cycle-based tabu search algorithm proposed in [21]. We first present the cycle-based neighborhood structure and the tabu search algorithm, then give out the requirement analysis, design and implementation details of the new code. We report at last the numerical results.

3.1 Cycle-based tabu search algorithm

First of all, let us recall the fundamental idea of the cycle-based neighborhood structure proposed in [21]: *One may move from one solution to another by*

1. *Identifying two points in the network together with two paths connecting these points, thus closing a cycle;*
2. *Deviating the total flow from one path to another such that at least one currently open arc becomes empty;*
3. *Closing all previously open arcs in the cycle that are empty following the flow deviation and, symmetrically, opening all previously closed arcs that now have flow.*

The general neighbourhood structure is written as

$\mathcal{V}(\hat{y}) = \{$ y: obtained from $\hat{y}$ by complementing the status of a number of arcs following the deviation of flow in a given cycle in $\mathcal{A}(\hat{y})$ $\}$.

Since such neighbourhoods are huge and their explicit exploration is not practical, a label-correcting based heuristic is proposed in [21] in order to identify low cost cycles in the residual network. Section 3.1.1 explains how to build a residual network from a given solution, and how to find the low cost cycle in a residual network. The tabu search procedures, including the local search, the intensification and the restoration, are described in Section 3.1.2.

3.1.1 Cycle-based neighborhood

Residual network

The concept of *residual network* is based on the following idea. Suppose that arc (i, j) carries x_{ij} units of flow. Then we can send an additional $u_{ij} - x_{ij}$ units of flow from node i to node j along arc (i, j), which will increase the routing cost by $c_{ij}x_{ij}$. Also notice that we can send up to x_{ij} units of flow from node j to node i over the arc (j, i), which amounts to canceling the existing flow on the arc (i, j), thus causing a routing cost decrement of $c_{ij}x_{ij}$. When using this idea in fixed-charge network, the fixed cost f_{ij} should also be considered, because sending more flow or canceling some may cause the arc (i, j) to be opened or closed.

For a given solution $\hat{y}$ and a specified flow value γ, a residual network $\mathcal{G}(\gamma, \hat{y})$ can be constructed by replacing each arc (i, j) of the original network by at most two residual arcs $(i, j)^+$ and $(j, i)^-$. A residual arc $(i, j)^+$ is included if its *residual capacity* $r_{ij} = u_{ij} - \sum_{k \in \mathcal{K}} x_{ij}^k$ is greater than or equal to γ, which means we can send γ additional units of flow on (i, j). The cost c_{ij}^+ associated to $(i, j)^+$

approximates the cost of routing γ units of additional flow on arc (i,j). It is equal to the average commodity routing costs on the arc, plus the fixed cost if it is currently closed:

$$c_{ij}^{+} = \frac{\sum_{k\in\mathcal{K}} c_{ij}^{k}}{|\mathcal{K}|}\gamma + f_{ij}(1 - (\lceil \min(1, \sum_{k\in\mathcal{K}} x_{ij}^{k}) \rceil)). \tag{3.1}$$

Symmetrically, a residual arc $(j,i)^{-}$ is included if its *residual capacity* $r_{ji} = \sum_{k\in\mathcal{K}} x_{ij}^{k}$ is equal or greater than γ. The cost c_{ji}^{-} associated to $(j,i)^{-}$ approximates the value of a reduction of γ units of the total flow (all commodities) currently using arc (i,j). This approximation is computed as the weighted average of the routing costs of the commodities currently using arc (i,j). The fixed cost f_{ij} is then subtracted if the reduction of γ units of flow leaves the arc empty:

$$c_{ji}^{-} = \begin{cases} -\frac{\sum_{k\in\mathcal{K}} c_{ij}^{k} x_{ij}^{k}}{\sum_{k\in\mathcal{K}} x_{ij}^{k}}\gamma - f_{ij} & \text{if } \sum_{k\in\mathcal{K}} x_{ij}^{k} = \gamma \\ -\frac{\sum_{k\in\mathcal{K}} c_{ij}^{k} x_{ij}^{k}}{\sum_{k\in\mathcal{K}} x_{ij}^{k}}\gamma & \text{if } \sum_{k\in\mathcal{K}} x_{ij}^{k} > \gamma \end{cases} \tag{3.2}$$

In the data files used by [21] and also by our re-implementation, the routing costs on an arc are the same for all commodities, i.e., $c_{ij}^{k} = c_{ij}\ \forall k \in \mathcal{K}, \forall (i,j) \in \mathcal{A}$. As a result, Equations (3.1) and (3.2) can be simplified as follows:

$$c_{ij}^{+} = \begin{cases} c_{ij}\gamma & \text{if } y_{ij} = 1 \\ c_{ij}\gamma + f_{ij} & \text{if } y_{ij} = 0. \end{cases} \tag{3.3}$$

$$c_{ji}^{+} = \begin{cases} -c_{ij}\gamma & \text{if } \sum_{k\in\mathcal{K}} x_{ij}^{k} > \gamma \\ -c_{ij}\gamma - f_{ij} & \text{if } \sum_{k\in\mathcal{K}} x_{ij}^{k} = \gamma. \end{cases} \tag{3.4}$$

Heuristic to identify low-cost cycles

In order to identify low cost cycles in the residual network, a heuristic is proposed in [21]. This heuristic is based on the label-correcting algorithm [1] that finds the shortest path between two nodes. In the presence of negative directed cycles, as in the case of the residual network, the original label-correcting algorithm gets trapped and keeps cycling. To avoid this, an explicit verification of cycles is added: if a scanned node j already belongs to the current shortest path from the source node to its predecessor node i, the label of node j will not be modified. Although the path generated by this heuristic is not necessarily the shortest path, it does produce very good cycles. Algorithm 1 describes this in detail.

Algorithm 1 Heuristic label correcting algorithm

$d(s) \leftarrow 0, pred(s) \leftarrow 0$

$d(j) \leftarrow \infty$ for each node $j \in \mathcal{N} - \{s\}$

LIST $\leftarrow \{s\}$

while LIST $\neq \phi$ **do**

remove an element i from LIST

for each node $j \in \mathcal{N}^+(i)$ **do**

if $(d(j) > d(i) + c_{ij})$ **and** (j is not in the path from s to i) **then**

$d(j) \leftarrow d(i) + c_{ij}$;

$pred(j) = i$;

if $j \notin$ LIST **then** add j to LIST.

In our implementation, a heap is used to maintain the LIST, so that each time the node with smallest label in the LIST is removed and evaluated.

3.1.2 Tabu search procedure

The initial solution is obtained by opening all arcs, solving the CMCF problem and triming the unused arcs. After that, the tabu search procedure explores the

solution space iteratively until a predefined stopping criterion is met.

At each iteration, the local search procedure described in Algorithm 2 determines and implements the best non-tabu move regardless of whether it improves the overall solution or not. When a particularly good solution is encountered, the search is intensified using Algorithm 3. If an infeasible solution is produced by a local search move, the restoration phase described in Algorithm 4 is undertaken.

The tabu search algorithm in [21] was quite simple, since the authors aimed only to test the performance of the cycle-based neighborhood structure. We complete the tabu search procedure with aspiration and diversification mechanisms.

Tabu status

A tabu list is a short-term memory that records characteristics of visited solutions to avoid cycling. At each time a residual network is built for a given solution, the tabu status of the residual arcs are determined. The rules used to determine a residual arc's tabu status are given in Figure 3.1.

As shown in Figure 3.1, a residual arc is made tabu if its corresponding arc in the original network is in the tabu list AND one of the following two cases occur: First, it has the same direction as its reference arc AND its reference arc is closed in the current solution, which means we want to route flow on a recently closed arc; Second, its direction is the opposite of its reference arc AND its reference arc carries exactly γ units of flow, which means we are trying to close this recently opened arc by canceling γ units of flow on it.

The tabu status of the residual arcs is checked during the exploration of the neighborhood. More specifically, it is checked when searching for the low cost path between two nodes using Algorithm 1. In this way, the low cost cycle identified will not close or open the arcs whose status have recently changed.

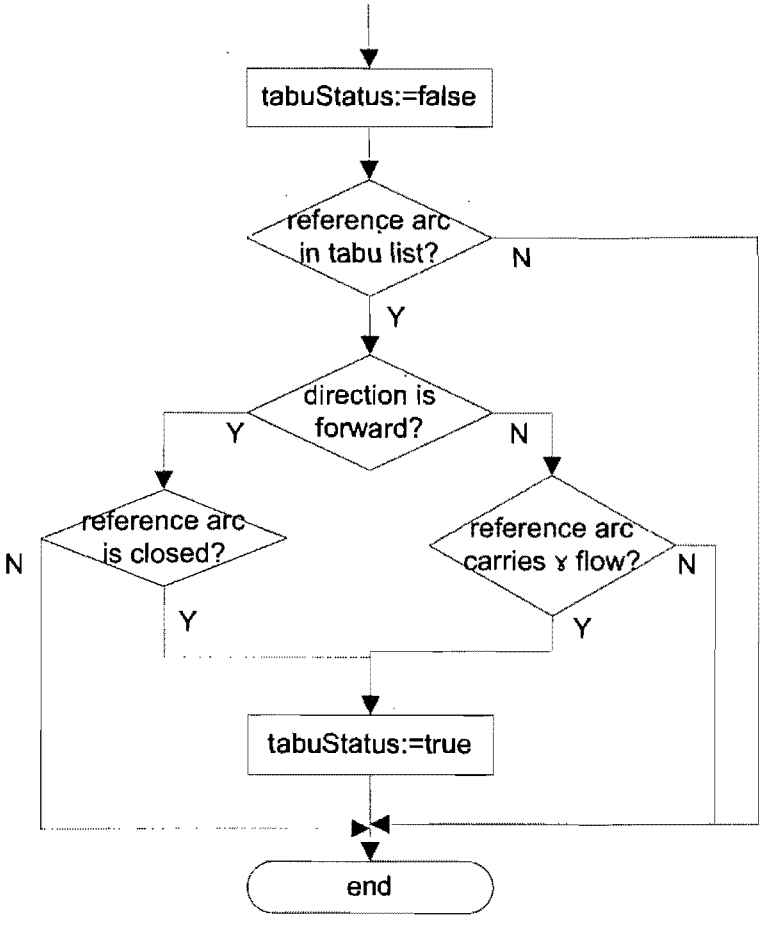

Figure 3.1: Determine residual arc's tabu status

Local search procedure

The local search procedure builds the neighborhood from the current solution, finds the best cycle, then carries out a local search move to modify the current design, and solves the corresponding CMCF problem using Cplex [26], as shown in Algorithm 2.

Algorithm 2 Local search procedure of tabu search

```
bestCycle ← NULL
build residual value set Γ(ŷ)
for each γ ∈ Γ(ŷ) do
    build residual network G(γ, ŷ)
    build candidate link set C(γ)
    find the best cycle for C(γ) in G(γ, ŷ)
    update bestCycle if needed
if bestCycle ≠ NULL then
    carry out a local search move to modify ŷ using bestCycle
    solve the associated CMCF using Cplex
    if the solution is infeasible then
        restoration
    if ŷ is good then
        intensification
else
    diversification
```

The residual value set $\Gamma(\hat{y})$ in Algorithm 2 is defined as the set of the total strictly positive volumes on the open arcs of the corresponding network:

$$\Gamma(\hat{y}) = \{\sum_{k \in \mathcal{K}} x_{ij}^k > 0 : (i,j) \in \mathcal{A}(\hat{y})\}.$$

The candidate link set $\mathcal{C}(\gamma)$ is built by choosing the residual arcs that meet

one of the following rules: residual arc $(i,j)^+$ is selected if arc (i,j) is closed and $u_{ij} \geq \gamma$; residual arc $(j,i)^-$ is added to the set if arc (i,j) is opened and $\sum_{k\in\mathcal{K}} x_{ij}^k = \gamma$. In the first case, routing flows on the cycle including $(i,j)^+$ will open (i,j), while in the second one, routing flows on the cycle including $(j,i)^-$ will close (i,j). In both cases, we are interested only in the moves that will change the network configuration. However, it is very time-consuming to consider the whole neighborhood, thus a percentage parameter is used to control the neighborhood size.

For each combination of $\gamma \in \Gamma(\hat{y})$ and $(i,j) \in \mathcal{C}(\gamma)$, a low-cost cycle is identified using the label-correcting based heuristic. The non-tabu cycle with lowest cost is identified as the best cycle. A local search move is then carried out by traversing the residual arcs of this best cycle and modifying the current network design as follows: for a residual arc $(i,j)^+$, open (i,j) if it is closed in the current config, because we want to route flow on it; for a residual arc $(j,i)^-$, close (i,j) if $\sum_{k\in\mathcal{K}} x_{ij}^k = \gamma$, since the flow on (i,j) will be canceled. The arcs that changed their status are added to the tabu list.

After the local search move, the associated CMCF problem is solved using Cplex. If the solution becomes infeasible, call the restoration procedure to deviate flows on artificial arcs in order to get a feasible solution. If the solution is "good", call the intensification procedure to search using the per-commodity cycle-based neighborhood. Here, by "good" solution, we mean the solution is within `IntensGap` percentage of the best solution, where `IntensGap` is a parameter. If no cycle is found for the current solution, which means a "cul-de-sac" is encountered, call the diversification procedure to jump out.

Intensification phase

When a local search move yields a solution that improves the overall best solution or is close to it, the intensification procedure is called. This procedure searches to improve the solution further by iteratively modifying the flow distribution of one commodity at a time. Algorithm 3 describes this in detail.

Algorithm 3 Intensification procedure of tabu search

```
improveFound ← TRUE
while improveFound do
    improveFound ← FALSE
    for each k ∈ K do
        bestCycle ← NULL
        build residual value set Γ(ŷ, k)
        for each γ ∈ Γ(ŷ, k) do
            build residual network G(γ, ŷ, k)
            build candidate link set C(γ, k)
            find the best cycle for C(γ, k) in the residual network
            update bestCycle if needed
        if bestCycle has negative cost then
            improveFound ← TRUE
            carry out an intensive move to modify ŷ and current solution using
              bestCycle
solve ŷ using Cplex
```

In the intensification phase, the residual value set is built for each commodity k to contain the strictly positive flow of k:

$$\Gamma(\hat{y}, k) = \{x_{ij}^k > 0 : (i, j) \in \mathcal{A}(\hat{y})\}.$$

Building the residual network $\mathcal{G}(\gamma, \hat{y}, k)$ in the intensification phase for a specific commodity k is a little bit different from that in the local search phase: when considering canceling flows, we cancel only the flow of commodity k and keep the

flow for other commodities fixed, that is, a residual arc $(j,i)^-$ is included in the residual network if $x_{ij}^k \geq \gamma$.

Compared to the local search procedure, only closed arcs are considered when building the candidate link set $\mathcal{C}(\gamma, k)$, since canceling the flow of commodity k on arc (i,j) has little help in closing arc (i,j).

An intensification move for commodity k is carried out only after a negative cost cycle is found. In the intensification move, γ units of flow of commodity k are deviated around the given cycle on the current solution, and the arcs are opened or closed correspondingly in the current configuration. The arcs that changed their status by this flow deviation are added to the tabu list.

At the end of the intensification phase, the CMCF associated to the current configuration is solved by Cplex, in order to make sure that the current solution represents its optimal flow distribution.

Restoration phase

A local search move may cause an infeasible solution, that is why a restoration phase is necessary. Algorithm 4 describes the procedure.

Algorithm 4 Restoration procedure of tabu search

for each artificial arc (i,j) **do**

  if (i,j) carries positive flow for commodity k **then**

    $\gamma \leftarrow$ the flow amount on (i,j)

    build residual network $\mathcal{G}(\gamma, \hat{y}, k)$

    find the shortest cycle for $(j,i)^-$ in $\mathcal{G}(\gamma, \hat{y}, k)$

    carry out an intensive move using the above cycle

solve the modified config

In the restoration phase, the flows on artificial arcs are deviated around the

shortest cycle by an intensification move, as in the intensification phase. At the end of the restoration phase, Cplex is called in order to have optimal flow distribution.

It is possible that there is no non-tabu cycle found for a specified artificial flow in the current solution. In such a case, we try to find a cycle without considering the tabu status. If there is still no cycle to use, we use the path in the current best solution to deliver the commodity on this artificial arc.

Diversification

Ensuring proper diversification is possibly the most critical issue in the design of tabu search heuristics [17]. By forcing the search into previously unexplored areas of the search memory, the diversification technique tries to alleviate the "local-optimal" problem, i.e., the local search procedure tends to spend most of its time in a restricted portion of the search space.

Our diversification mechanism is based on a frequency memory, where one records for each design arc the total number of times it is opened during the search process.

A diversification phase is triggered in two situations: when the overall best solution has not been improved for a predefined number of iterations, or when there is no cycle found for the current solution thus nowhere to move. In the diversification procedure, the most often opened arcs are closed and the least often opened ones are opened on the basis of the current configuration, then the modified configuration is solved using Cplex. The percentages of total arcs that will be opened or closed in this diversification phase can be specified as parameters.

Aspiration

The tabus are sometimes too powerful and they may prohibit attractive moves, or they may lead to an overall stagnation of the search process [17]. It is thus necessary to allow some tabu moves when there is no danger of cycling.

In the proposed cycle-based neighborhood, the cost of a cycle is only an approximation of the change to the objective function value. It is not practical to evaluate them explicitly and then allow the one that results in an overall best solution, as the most commonly used aspiration criterion in almost all tabu search implementations.

In our implementation, a cycle is made tabu if it includes a tabu residual arc. There are two cases where we need to consider the tabu cycles. First, in the local search process, if there is no non-tabu cycle found for a given γ, we try to find the shortest cycle without considering their tabu status, but we penalize this cycle by a ratio less than 1. By doing so, the tabu cycle is allowed to enter in the competition for the overall best cycle.

The second case of using aspiration is during the restoration phase. If for an artificial flow amount γ of commodity k, there is no non-tabu cycle found in its residual network, try to find the shortest cycle without considering tabu status, and use this cycle to deviate the artificial flow.

3.2 Requirement analysis of the new implementation

The available code prior to this work was a super set of the tabu search algorithm proposed in [21], which includes tabu search, path-relinking and learning

mechanisms (see Ghamlouche's Ph.D. thesis [19]). The main problems are: it is a mix of C and C++ style, there are too many global functions and variables, there are too few comments, and there are about 100 parameters used to control the search procedure. It is almost impossible to take out the tabu search part and modify it. So we decided to re-design and re-implement it according to [21].

The main function of this software is to solve the CMND problem described in a given data file. Several parameters can be specified by the user to control the tabu search procedure. So we keep the command line format as:

```
./main dataFileName parameterFileName
```

The output of the existing research code consists of the tabu search trace and the overall best solution. We keep the format of the output too.

3.2.1 Data file format

We use the same two data sets as in [21]. Problems in these sets are general transshipment networks with no parallel arcs, single origin-destination pair for each commodity, and unique but arc-specific commodity routing costs. The format is described as follows:

```
MULTIGEN.DAT:
number_of_nodes  number_of_arcs  number_of_commodities
(for each arc:) from to routing_cost capacity fixed_cost dump1 dump2
(for each commodity:) from to demand
```

where dump1 and dump2 can be any number.

3.2.2 Parameter file format

The tabu search process can be controlled by many parameters. The original parameter file format is not user-friendly, so we created a new format. Comment line begins with #, the parameter file ends with the line "END". Unspecified parameters will be assigned to their default values which are given in the header file *Parameters.h*. An example of parameter file is given below. Most parameters are self explainable or explained by the comments line.

```
#Stopping criteria
TabuTotalIteration = 400
TabuNonImproveIteration = 400
TabuTotalRunSec = 36000

#The percentage of closed arcs randomly selected
NeighbourhoodPercentage = 0.5
#Tabu tenure
TabuTenure = 2
#The threshold used to determine a good solution
IntensGap = 0.09
#Whether or not to do intensification on the initial solution
IntensInitialSolution = 1

#Whether or not use aspiration
Aspiration = 0;
#Aspiration rate: If no NON-TABU cycle found, try to find
#a cycle with TABU but penalize the cycle cost with this rate
TabuPenalty = 0.8;

#Whether or not do diversification
Diversification = 0
```

```
#How many iterations without improvement will cause diversification
DiversBeginIters = 9
#The percentage of arcs to open or close in diversification
DiversOpenArcs = 0.03
DiversCloseArcs = 0.03

RandomSeed = 1
END
```

3.3 Architecture design

The project can be divided into several relatively independent packages as shown in Figure 3.2.

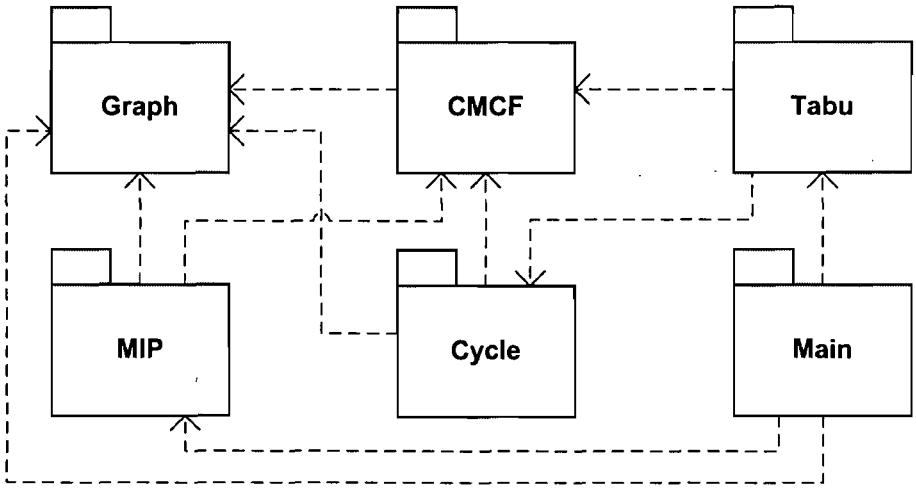

Figure 3.2: Architecture design for the cycle-based tabu search

The `Graph` package reads the data file and creates a graph object, which contains all data information about the problem instance. Once created, the graph becomes read-only.

The `CMCF` package is able to solve the capacitated multicommocity minimum

cost flow problem. Given a configuration (a design) of the graph, Cplex is called to solve the CMCF problem, and solution information is then recorded. Here a solution should contain not only the flow distribution and routing cost, but also the arc status and fixed cost. It is a solution to the CMND problem.

The `Cycle` package deals with the cycle-based neighborhood. For a given solution of the graph, a residual network can be created for a specified flow value, from which the low-cost cycle can be identified.

The `Tabu` package provides the function of tabu search. At each search step, it uses the `CMCF` package to solve a configuration, creates a neighborhood for the current solution, then asks the `Cycle` package to give the best cycle in the neighborhood, and takes a move using the best cycle. Intensification, restoration from infeasible solution, aspiration and diversification should be provided in this package.

In order to evaluate the tabu search performance, the `MIP` package defines a solver that solves the CMND problem as a mixed integer program using Cplex. At last, the `Main` package defines global constants, reads parameter files and tests the software.

3.4 Class design and implementation

In this section, we present class design for each package, and descibe some important data structures.

3.4.1 Graph package

Graph and its helper classes, including *NetworkArc*, *Node* and *Prod*, are defined in this package, as shown in Figure 3.3.

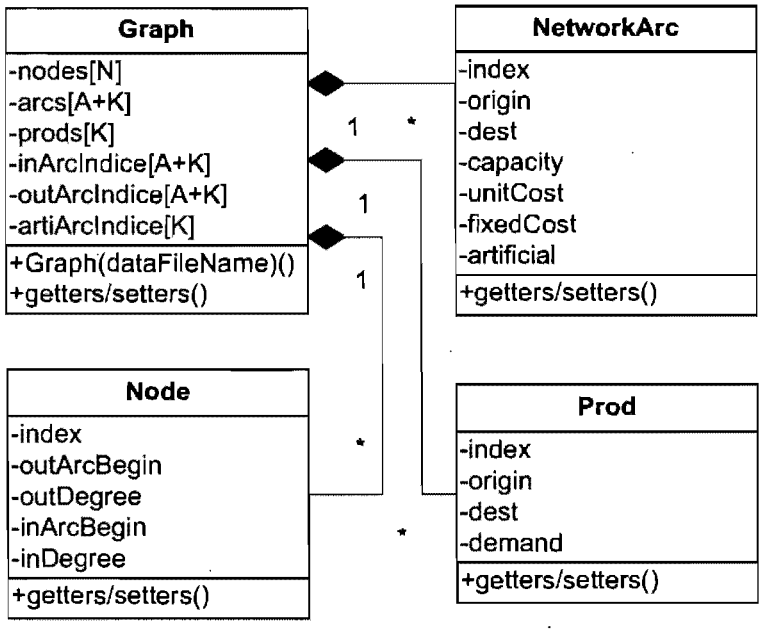

Figure 3.3: Graph package design

The *NetworkArc* class represents a directed design arc, its data fields are origin i, destination j, fixed cost f_{ij}, routing cost c_{ij}, and arc capacity u_{ij}. The *Prod* class stands for a transportation requirement (i.e., a commodity), and its data fields are origin $o(k)$, destination $s(k)$ and demand w^k. For each commodity k, there is also an artificial arc from $o(k)$ to $s(k)$ with high routing cost, zero fixed cost and a capacity of w^k. Artificial arcs are added in order to avoid infeasible solutions. The *Node* class has information about its in-degree, out-degree, as well as the index of its first outgoing arc and first incoming arc in order to traverse its neighbors.

The *Graph* class is a container class. Three vectors are used to hold the elements: nodes[N], arcs[$A + K$] and prods[K]. In order to traverse the out-

ward and inward neighbors ($\mathcal{N}^+(i)$ and $\mathcal{N}^-(i)$) of node i, two help vectors $inArcIndice[A+K]$ and $outArcIndice[A+K]$ are used to record the incoming and outgoing arc indices for each node. Together with the information in *Node*, this data structure is the so-called *forward and reverse star* representation. It provides an efficient way to traverse a node's neighbours [1]. Besides, vector $artiArcIndice[K]$ is used to specify the artificial arc's index for each commodity.

The constructor of the *Graph* class reads the data file, creates and fills the above three vectors, and throws *DataFileReadingException* if the data file is not in good format or an error occurred during reading the data file.

3.4.2 CMCF package

In this package, *CplexSolver* solves the CMCF problem for a given design vector (a *Config*) of the graph, and returns a *Solution*, as displayed in Figure 3.4.

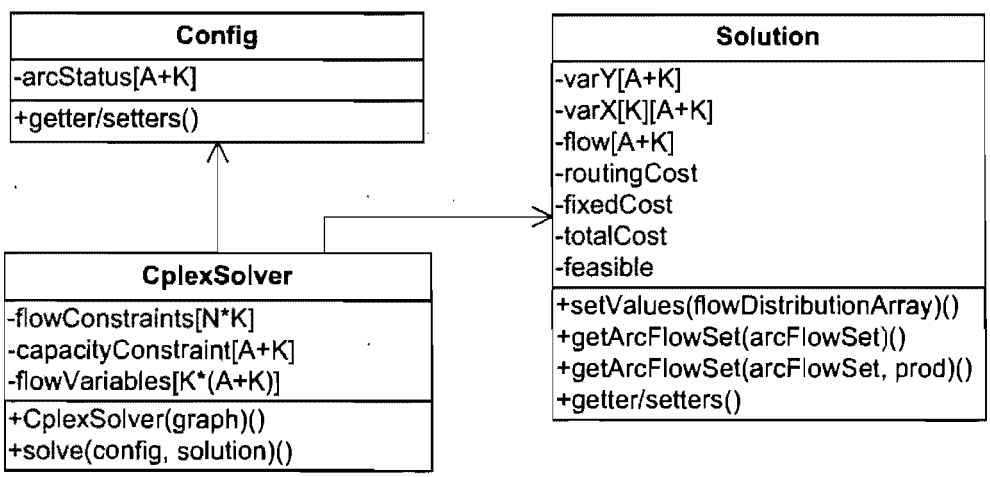

Figure 3.4: CMCF package design

The *Config* class represents a design vector $\hat{y}$ of the graph to be solved. Its data field contains simply a boolean vector of size $A+K$, where "true" means an arc is open and "false" means it is closed. It also provides functions to manipulate

the arc status.

The *CplexSolver* class solves the corresponding CMCF problem for a given design of the graph as a linear program. There are $K(A+K)$ decision variables that represent the flow distribution on each arc for each product, including artificial arcs. There are two constraint arrays, the first contains NK expressions representing the network flow conservation relations, and the second contains $A+K$ expressions that stand for the arc capacity constraints.

The *CplexSolver* class is constructed using the *Graph*. During the construction, the model is created, which means the objective function and all constraints are populated. In the initial model, all arcs are considered opened, so the right-hand side of the arc capacity constraints are set to arc capacities. The function *solve()* requires a *Config* as input and a *Solution* as output parameters. Inside this function, the right hand side of arc capacity constraints are reset to u_{ij} or 0 according to the arc status specified in the given *Config*. Then Cplex is called to solve this model as a linear program, and the resulting variable value array is used to specify the values in the *Solution*. The arcs that are opened in the *Config* but carry zero flow are then closed by a so-called *trim* procedure.

A *Solution* to the CMND problem holds information about the design decision, the flow decision and the resulting costs. Vector $varY[A+K]$ contains the design variables y_{ij}, the two-dimensioned vector $varX[K][A+K]$ holds the flow distribution variables x_{ij}^k. From these variables, the routing cost, fixed cost and total cost are calculated. For computation convenience, a helper vector $flow[A+K]$ is used to store the amount of flow on each arc, i.e., $x_{ij} = \sum_{k \in K} x_{ij}^k$. If there is any flow on some artificial arc, the solution is considered infeasible.

The *setValue()* function of *Solution* takes as input an array of $K(A+K)$ numbers representing the flow distribution, fill the corresponding vectors and calculate the costs. *Solution* class also provides some functions to modify the flows, the

arc status, and recalculate the costs. The overloaded function *getArcFlowSet()* is used in the tabu search process to construct residual value set $\Gamma(\hat{y})$.

3.4.3 `Cycle` package

This package deals with the idea of cycle-based neighborhood. A *ResidualGraph* can be constructed for a given solution and a specified flow value γ. It contains N *ResidualNodes* and at most $2(A+K)$ *ResidualArcs*. A low-cost *Cycle* then can be identified from the *ResidualGraph*, which consists of a sequence of *ResidualArcs*. See Figure 3.5 for their interfaces and relationshipss.

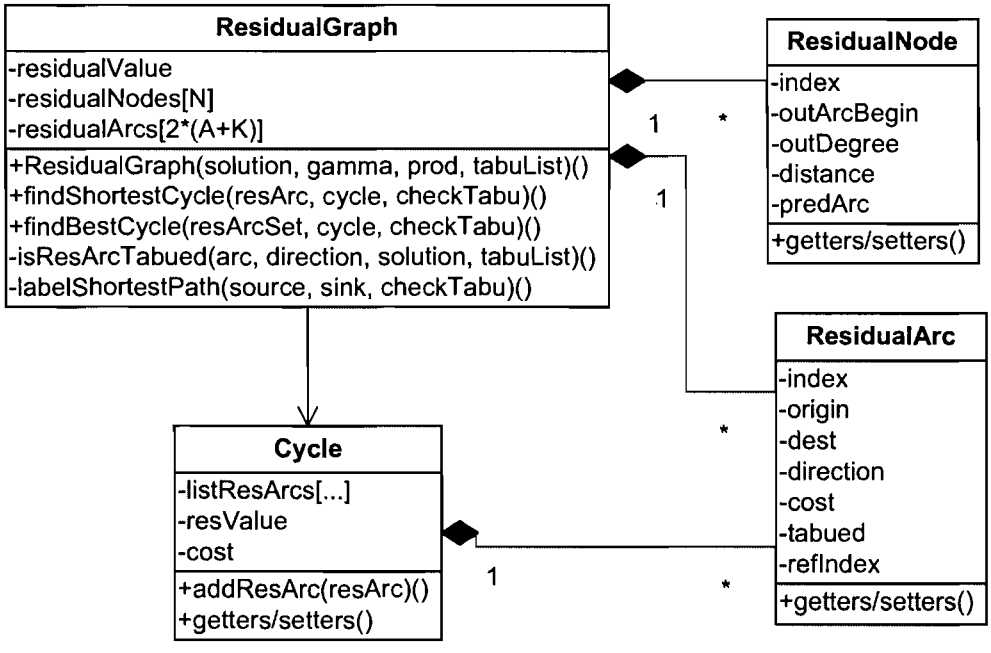

Figure 3.5: `Cycle` package design

The constructor of the *ResidualGraph* class takes a *Solution* $\hat{y}$ and a residual value γ as parameters to build the residual network $\mathcal{G}(\gamma, \hat{y})$.

The *ResidualNode* class has information about a node's out-degree and the index of its first out-going arc in the residual network. Both are used to traverse

its outward neighbors efficiently. Besides, in order to find a shortest path using the label-correcting algorithm, each node i has a distance label $d(i)$ representing the length of the current directed path from the source, and the predecessor node $pred(i)$ in the current directed path.

The *ResidualArc* class records the origin, destination, direction, cost and tabu status of an arc in the residual network. The cost of a residual arc is given in equations (3.3) and (3.4). The data field *direction* means this residual arc has the same or the opposite direction compared to the original arc (i, j), that is, it is $(i, j)^+$ or $(j, i)^-$. The data field $refIndex$ refers to the arc (i, j) in the original network. A residual arc' s tabu status is decided during the construction of the residual network. The rules used to determine a residual arc's tabu status are given in Figure 3.1.

Cycle class contains a vector of *ResidualArcs*, its *residual capacity*, and the cost, which is calculated as the sum of its residual arcs' cost. The *residual capacity* of a cycle denotes the maximum flow one can deviate around the cycle.

The function *findShortestCycle()* of *ResidualGraph* is used to find the low-cost path from j to i for a given *ResidualArc* (i, j) using Algorithm 1, and thus construct a cycle. For a given set of residual arcs, the function *findBestCycle()* identifies the best one among all the low-cost cycles constructed using the former function. The caller of these functions even has the flexibility of specifying whether or not to consider the tabu status of residual arcs when looking for such cycles.

3.4.4 Tabu package

In this package, as shown in Figure 3.6, the *TabuSolver* class solves the CMND problem. The *TabuList* class is a short-term memory used to prevent cycling,

and *FreqMemory* class is a long-term memory used for diversification.

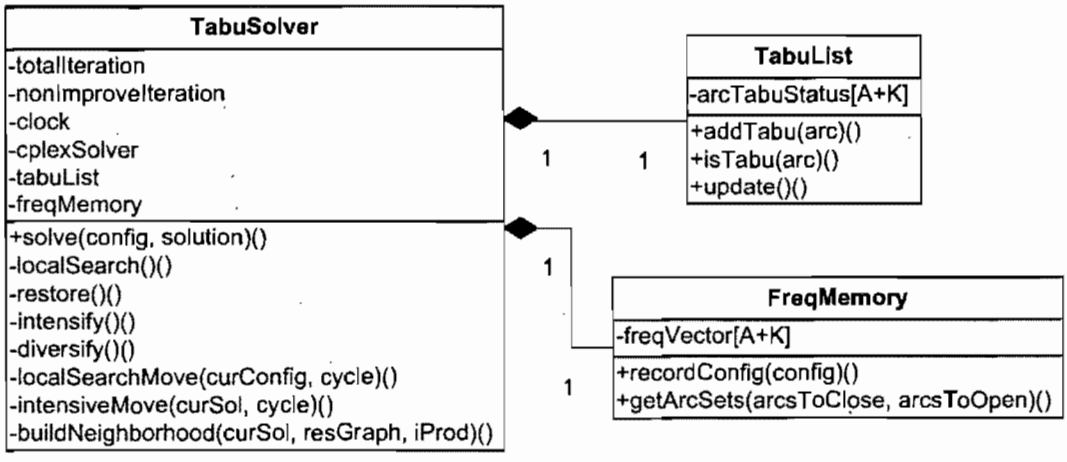

Figure 3.6: `Tabu` package design

The *TabuList* class is very simple. It contains a vector of $A + K$ integer elements representing the tabu status of each arc. Initially, all elements are set to 0 which means no arc is made tabu. When an arc is added to the tabu list, its tabu status is set to `TabuTenure`, a constant specified in the parameter file. A nonzero tabu status t for arc (i, j) means the arc is made tabu for the next t iterations. Tabu list is updated at each tabu search iteration by decreasing the non-zero tabu status.

FreqMemory is a long-term memory where we record for each design arc the total number of times it is opened during the search process. Function *recordConfig()* records the arc status of a *Config* after it is evaluated (the corresponding CMCF problem is solved). Another function *getArcSets()* sorts the elements according to their frequency, and returns two sets of arc indices: the most often opened arcs and the least often opened arcs. The number of arcs in these two sets are specified by `DiversOpenArcs` and `DiversCloseArcs` in the parameter file.

The *TabuSolver* class holds the total iteration number and non-improvement iteration number as its data fields, as well as a *TabuList*, a *CplexSolver* and a

FreqMemory as its object data fields. After construction using *Graph*, its public function *solve()* takes a *Config* as input parameter, and writes results to its output parameter, the *solution*.

In the public function *solve()*, an initial solution is obtained at first by solving the given *Config* by Cplex and triming the unused arcs. If `IntensInitialSolution` in the parameter file is turned on, an intensification procedure is carried out on this initial solution. Then, the search proceeds until one of the stopping criteria is met. Three criteria are given in the parameter file by `TabuTotalIteration`, `TabuNonImproveIteration` and `TabuTotalRunSec`. At each search iteration, the private function *localSearch()* is called to explore the neighborhood and follow the best non-tabu move. If `Diversification` is enabled in the parameter file and the non improvement iteration number reaches `DiversBeginIters`, *diversify()* is triggered to jump out of the "valley". During the search process, the current config, the current solution and the current best solution are maintained.

The procedures *localSearch()*, *restore()*, *intensify()* and *diversify()* implement the algorithms described in Section 3.1.2.

3.4.5 `MIP` and `Main` package

The `MIP` package contains only one class *MIPCplexSolver*, which inherits from *CplexSolver* and modifies the model by adding binary design variables. The objective function and capacity constraints are adjusted too by adding y_{ij} terms, as shown in Equation (2.1) in Chapter 2.

The `Main` package consists of a *Clock* class used to time the process, a static class *Parameters* used to read the parameter file, a header file *Global* that defines global constant numbers, some utility functions, and, of course, a *main()* function to run the algorithm.

3.5 Numerical results

To evaluate our implementation, we experiment with the same 43 problem instances of set C used by [21]. The computer code is written in C++, Cplex version is 10.0. Tests were conducted on one 3.00GHz processor with 1 Gigabyte of RAM, operating under Linux CentOs 4.2. Since Cplex solver's version and hardware have changed, the only meaningful way to compare the two implementations is to use the same number of total iterations. We set this number to 400, and use the same search parameters as those used in [21], that is, `NeighborhoodPercentage` = 0.5, `TabuTenure` = 2 and `IntensGap` = 0.09. As for the diversification and aspiration, we turned them off in order to simulate the existing research code.

Table 3.1: Computational results for tabu search

No.	Prob	Opt	Tc(400)	Min	Gap	Avg	Gap
25	25.100.10.V.L	14712	14712	14712	0.00%	14746	0.23%
26	25.100.10.F.L	14941	14941	15037	0.64%	15216	1.84%
28	25.100.10.F.T	49899	49899	50460	1.12%	50948	2.10%
31	25.100.30.V.T	365272	365385	365385	0.00%	365385	0.00%
30	25.100.30.F.L	37055	37583	38093	1.36%	38379	2.12%
32	25.100.30.F.T	85530	86296	87000	0.82%	87189	1.03%
33	20.230.40.V.L	423848	424778	424277	-0.12%	425264	0.11%
35	20.230.40.V.T	371475	371893	371778	-0.03%	371778	-0.03%
36	20.230.40.F.T	643036	645812	648392	0.40%	648949	0.49%
41	20.300.40.V.L	429398	429535	429535	0.00%	429672	0.03%
42	20.300.40.F.L	586077	593322	597711	0.74%	598510	0.87%
43	20.300.40.V.T	464509	464724	466102	0.30%	466904	0.47%
44	20.300.40.F.T	604198	607100	611899	0.79%	614680	1.25%

Continued on next page

No.	Prob	Opt	Tc(400)	Min	Gap	Avg	Gap
37	20.230.200.V.L	94247	98995	100496	1.52%	100992	2.02%
38	20.230.200.F.L	138182	146535	148804	1.55%	151260	3.22%
39	20.230.200.V.T	98013	104752	105016	0.25%	106098	1.28%
40	20.230.200.F.T	136072	147385	150098	1.84%	150774	2.30%
45	20.300.200.V.L	74929	80819	78820	-2.47%	81034	0.27%
46	20.300.200.F.L	116489	123347	124777	1.16%	125855	2.03%
47	20.300.200.V.T	74991	79619	79183	-0.55%	79733	0.14%
48	20.300.200.F.T	108037	114484	115563	0.94%	116247	1.54%
65	100.400.10.V.L	28423	28677	28744	0.23%	28823	0.51%
66	100.400.10.F.L	23949	23949	24022	0.30%	24096	0.61%
68	100.400.10.F.T	64757	67014	68142	1.68%	69008	2.98%
71	100.400.30.V.T	384802	385508	385508	0.00%	385510	0.00%
70	100.400.30.F.L	50573	51552	51850	0.58%	52076	1.02%
72	100.400.30.F.T	138107	145144	145276	0.09%	147563	1.67%
49	30.520.100.V.L	53964	54958	54959	0.00%	55106	0.27%
50	30.520.100.F.L	94747	99586	102156	2.58%	102644	3.07%
51	30.520.100.V.T	52062	52985	52876	-0.21%	53098	0.21%
52	30.520.100.F.T	98286	105523	104079	-1.37%	104899	-0.59%
57	30.700.100.V.L	47603	48398	48787	0.80%	49066	1.38%
58	30.700.100.F.L	60324	62471	62520	0.08%	62844	0.60%
59	30.700.100.V.T	45949	47025	47085	0.13%	47218	0.41%
60	30.700.100.F.T	55365	57886	56534	-2.34%	57092	-1.37%
53	30.520.400.V.L	112774	120652	120241	-0.34%	121129	0.40%
54	30.520.400.F.L	149759	161098	163334	1.39%	163867	1.72%
55	30.520.400.V.T	114687	121588	121358	-0.19%	122477	0.73%
56	30.520.400.F.T	153262	167939	167168	-0.46%	167680	-0.15%
61	30.700.400.V.L	98268	106777	105605	-1.10%	106337	-0.41%

Continued on next page

No.	Prob	Opt	Tc(400)	Min	Gap	Avg	Gap
62	30.700.400.F.L	136123	148950	147420	-1.03%	149077	0.09%
63	30.700.400.V.T	95530	101672	101343	-0.32%	102043	0.37%
64	30.700.400.F.T	130698	142778	141532	-0.87%	143454	0.47%

The numerical results are displayed in Table 3.1. The first two columns give the number of the instance and its characteristics, which includes the number of nodes, arcs and commodities, as well as two letters summarizing the fixed cost and capacity information: a relatively high or low fixed cost relative to the routing cost is signaled by the letter **F** or **V**, respectively, while letters **T** and **L** indicate whether the problem is tightly or somewhat loosely capacitated compared to the total demand.

The `Opt` column corresponds to the best integer solution found by *MIPCplexSolver* within a time limit of 10 hours. Column `Tc(400)` corresponds to the results displayed in [21]. Then, we have column `Min` the best solution over three runs of our implementation, column `Avg` the average value of three runs. The two columns `Gap` following column `Min` and `Avg` are the relative gap between our implementations and the result of `Tc(400)`, respectively.

The highest gap between `Min` and `Tc(400)` is 2.58%, the lowest one is -2.47%, and the average is 0.23%. As for the gap between `Avg` and `Tc(400)`, these numbers are 3.22%, -1.37% and 0.87%, respectively. Because of the existence of random factors (when constructing the neighborhood), it is reasonable to believe that our re-implementation achieved the same performance as that in [21].

To see further the similarity between the two implementations, the distribution of the gap between `Min` and `Ttc(400)` is given in Table 3.2.

From the numerical results, we can see that the gaps between the two implementations are quite small (0.23% for best solution and 0.87% for average

Table 3.2: Gap between two implementations

≤ -2	$(-2, -1.5]$	$(-1.5, -1]$	$(-1, -0.5]$	$(-0.5, 0]$
2	0	3	2	11
$(0, 0.5]$	$(0.5, 1]$	$(1, 1.5]$	$(1.5, 2]$	> 2
9	7	4	4	1

solution). They are even smaller than the average random difference over three runs (the random difference is 1.07% in average). As for the gap distribution, it is almost normally distributed, from which we can say that the two implementations reach the same performance.

Table 3.3: Computational results of diversification

No.	Off	20-0.01	20-0.02	20-0.03	40-0.01	40-0.02	40-0.03
25	14712	14742	14750	14729	14712	14814	14712
		0.20%	0.26%	0.12%	0.00%	0.69%	0.00%
30	39088	38660	38289	38266	39025	39062	38384
		-1.09%	-2.04%	-2.10%	-0.16%	-0.07%	-1.80%
35	372142	371778	371612	371778	372097	371893	371893
		-0.10%	-0.14%	-0.10%	-0.01%	-0.07%	-0.07%
44	612267	613468	607725	610353	615060	615972	612093
		0.20%	-0.74%	-0.31%	0.46%	0.61%	-0.03%
37	102508	102702	104086	102277	102782	102532	102656
		0.19%	1.54%	-0.23%	0.27%	0.02%	0.14%
46	125528	125357	126711	127692	124982	123998	127692
		-0.14%	0.94%	1.72%	-0.43%	-1.22%	1.72%
72	146139	144444	147942	147672	148433	147570	148174
		-1.16%	1.23%	1.05%	1.57%	0.98%	1.39%
49	55053	55503	55664	55539	55175	55175	54910
		0.82%	1.11%	0.88%	0.22%	0.22%	-0.26%
58	64235.3	63750	63430	62218	63101	63626	62825
		-0.76%	-1.25%	-3.14%	-1.77%	-0.95%	-2.20%
55	120564	122902	123178	122032	122044	122044	122044
		1.94%	2.17%	1.22%	1.23%	1.23%	1.23%
64	144053	144053	144053	143066	144053	141209	144053
		0.00%	0.00%	-0.69%	0.00%	-1.97%	0.00%
total		0.10%	3.07%	-1.58%	1.37%	-0.52%	0.14%

In order to test the aspiration and diversification procedures, we turn on the relative parameters and experiment with 11 instances selected from the instance set C. For the aspiration operation, we tried three different values (0.8, 0.9 and

1.0) of `TabuPenalty`. The numerical results are almost identical to that without aspiration. This can be explained as follows: the aspiration is carried out when there is no non-tabu cycle found in the residual network $\mathcal{G}(\gamma, \hat{y})$ for a given combination of $\gamma \in \Gamma$ and $(i, j) \in \mathcal{C}(\gamma)$. It is quite rare that this situation occurs. Even when an aspiration is carried out, the resulting cycle has to enter the competition for the overall best cycle, which reduced further its potential impacts.

As for the diversification option, there are three parameters to be ajusted. The `DiversBeginIters` parameter represents the frequency of diversification, while the `DiversOpenArcs` and `DiversCloseArcs` parameters represent the strength of the diversification operation. We tried two values (20, 40) for `DiversBeginIters` and three values (0.01, 002 and 0.03) for `DiversOpenArcs` and `DiversCloseArcs`. All other parameters are kept same. The numerical results are shown in Table 3.3.

In Table 3.3, the `Off` column corresponds to the solution obtained without diversification. The percentage under each solution is the relative difference between the solution and that in the column `Off`. The last row of this table is the sum of the relative difference. From the numerical results, we can see that with careful calibration, the diversification procedure is able to improve the performance of the cycle-based tabu algorithm.

3.6 Conclusion

The cycle-based neighborhood structure is known to be powerful in addressing CMND problems. By re-implementing the algorithm, we got a profound understanding about tabu search, the CMND problem and the cycle-based neighborhood. From the software engineering point of view, after the re-design and re-implementation, this valuable research work now can be reused, maintained

and extended.

Two useful extensions should be considered. One is to change the data file format so that the routing costs may be specified for each commodity on each arc. On the current software basis, the constuctor of the class *Graph* has to be changed to read the new data file, and the *NetworkArc* class needs to be changed to hold the different routing costs for different commodities. Another slight modification is to change the calculation of the residual arc cost in the constructor of the class *ResidualGraph*.

The other extension is to use other algorithms than Cplex to solve the related CMCF problems. An intermediate function is needed to transform the *Graph* defined in the current implementation to that used by other solvers.

The implementation and numerical experimentation raised several points that need to be mentioned.

First, intensifying on the initial solution can make a big difference, especially for larger instances. It can be explained as follows: there is a lot of room to improve the initial solution, and we can begin from a better point after the initial intensification. But from this point, we can also infer that the whole search process does not diversify sufficiently. It is the intensification phase that contributes mostly to finding good solutions. Maybe the search is captive in the local valley since we moved down from the very first solution. Performance improvements then can be expected after turning on the diversification. More tests should be done for tuning the diversification parameters.

Second, when constructing the candidate link set $\mathcal{C}(\gamma, k)$ in the intensification phase, we tried two approaches. The first way is to choose the arcs that carry flow of product k at each iteration, thinking that maybe we can deviate the flow, then close the arc (hopefully). We tried 100%, 70% and 50% of such arcs. The second way is to randomly choose the closed arcs as in the local search step. By

comparing the results, we found that the second way works better, which is quite out of our expectation. Here we can explain it as follows: since the intensification phase is very powerful in moving down to the local minimum, the first way does not help much in doing so, while the second way introduces some random factor that plays a diversification-like role to some degree. Once again, we feel the need for a good diversification.

The last observation is about when to check the tabu status. At the beginning, we just tried to find the shortest cycle for each combination of γ and candidate link (i, j), then ignore the tabu moves, but the performance on larger instances was inferior to `Tc(400)` in [21]. We thus found the first version of the existing code, traced the program line by line, and finally realized the reason. In the existing research code, the tabu status is checked during the identification of the shortest cycle in the residual network. As a consequence, only non-tabu cycles take part in the competition of the overall best cycle, which is then used as a local search move. While in our first implementation, the shortest cycle for the pair $(\gamma, (i, j))$ was dropped if it is unfortunately made tabu. This trival change made a big difference: by considering tabu status in a profound level (in the label-correcting algorithm), the number of candidate moves from which we choose the best one is indeed increased, and better solutions can be found. This shows again the importance of having a well documented software project.

Chapter 4

Progressive hedging algorithms for stochastic network design

In this chapter, we first present at first the progressive hedging algorithm (PHA) of Rockafellar and Wets [37]. We then cast PHA into a meta-heuristic framework, where the sub-problems are modified for each scenario and solved heuristically. Two ways to modify the sub-problems are proposed, the first one uses augmented Lagrangian to relax the non-anticipativity constraints, while the second one modifies the subproblems heuristically in order to drive all scenarios toward one global design. Last, we present how to transform the proposed idea into an actual algorithm by specifying the stopping criteria, subproblem solver and parallel computation structure.

4.1 Progressive hedging algorithm

The progressive hedging algorithm is proposed in [37] and reviewed in [39]. It enforces the implementability constraint algorithmically. It can be interpreted as a scenario decomposition method for stochastic programming problems.

Let x^* be the optimal solution to the following stochastic problem

$$\min E\{f(x,\xi)\}$$
$$\text{subject to } x \in C \subset R^n.$$

Let $\mathcal{S}$ denote the scenario set. For each scenario $s \in \mathcal{S}$, let the *scenario solution* x^s represents an optimal solution to the deterministic scenario problem:

$$\min f(x,s)$$
$$\text{subject to } x \in C_s \subset R^n.$$

Define the *average solution* $\overline{x}$ as the expectation taken over the scenario solutions:

$$\overline{x} = \sum_{s\in\mathcal{S}} p^s x^s.$$

The progressive hedging algorithm generates a sequence of estimates of the optimal solution x^*, obtained by insisting progressively more and more on the requirement that the solutions generated by the scenario problems must be implementable.

At each iteration t, the *average solution* is calculated as

$$\overline{x}^t = \sum_{s\in\mathcal{S}} p^s x^{st} \tag{4.1}$$

where x^{st} is an optimal solution to the modified scenario problem

$$\min f^t(x,s) = f(x,s) + \lambda^{t-1}(s)x + \frac{1}{2}\rho|x - \overline{x}^{t-1}|^2. \tag{4.2}$$
$$\text{subject to } x \in C_s$$

The vectors λ^t are adjusted as follows:

$$\lambda^t(s) = \lambda^{t-1}(s) + \rho[x^{st} - \overline{x}^t]. \tag{4.3}$$

The progressive hedging algorithm is described in Algorithm 5.

Algorithm 5 Progressive hedging algorithm

Initialize:

$f^0(x,s) \leftarrow f(x,s)$, $\overline{x}^0 \leftarrow 0$,
$\lambda^0(s) \leftarrow 0$, choose $\rho > 0$
$t \leftarrow 1$

repeat

for each $s \in \mathcal{S}$ **do**
solve equation (4.2) to get scenario solution x^{st}
calculate $\overline{x}^t$ using equation (4.1)
update $\lambda^t(s)$ using equation (4.3)
$t \leftarrow t+1$

until $x^{st} = \overline{x}^t$, $\forall s \in \mathcal{S}$

This algorithm only requires the capability of solving individual scenario problems (and linear-quadratic perturbations thereof). Its terminating criterion is that for all $s \in \mathcal{S}$, the scenario solution x^{st} converges to the average solution $\overline{x}^t$, which is implementable and feasible. At each iteration t, the distance between the scenario solution vector $\{x^{st}, \forall s \in \mathcal{S}\}$ and the average solution $\overline{x}^t$ can be defined as follows:

$$\theta^t = \sum_{s \in \mathcal{S}} p^s |x^{st} - \overline{x}^t|^2.$$

This distance converges to 0, at least in the convex case. Even if the functions are nonconvex, $\overline{x}^t$ is still the global minimum provided the functions $f(\cdot, s)$ satisfy certain minimum growth conditions [39].

For linear stochastic problems without integer requirements, Rockafellar and Wets proved that PHA converges to the global optimum in a linear rate. Another advantage is that at each iteration, we can always have at hand a solution estimate that is better than that of all previous iterations. Unfortunately, this is not the case in stochastic integer programming. The convergence to optimality of this algorithm has not been formally proved for stochastic integer programming, and nonconvergence is possible [33].

4.2 Research methodology

We repeat the scenario-decomposition model (Equation 2.5 in Chapter 2) here for clarity.

$$\min \sum_{s\in\mathcal{S}} p^s \Big(\sum_{(i,j)\in\mathcal{A}} f_{ij} y_{ij}^s + \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^{ks} \Big) \tag{4.4a}$$

subject to

$$\sum_{j\in\mathcal{N}^+(i)} x_{ij}^{ks} - \sum_{j\in\mathcal{N}^-(i)} x_{ji}^{ks} = d_i^{ks} \qquad \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \forall s \in \mathcal{S} \tag{4.4b}$$

$$\sum_{k\in\mathcal{K}} x_{ij}^{ks} \le u_{ij} y_{ij}^s \qquad \forall (i,j) \in \mathcal{A}, \forall s \in \mathcal{S} \tag{4.4c}$$

$$y_{ij}^s \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A} \tag{4.4d}$$

$$x_{ij}^{ks} \ge 0 \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K}, \forall s \in \mathcal{S} \tag{4.4e}$$

$$y_{ij}^s = y_{ij}^t \qquad \forall (i,j) \in \mathcal{A}, \forall s,t \in \mathcal{S}, s \ne t \tag{4.4f}$$

This model is scenario-separable except for the non-anticipativity constraints (4.4f), which imply that the design decision taken for each scenario should be the same. In other words, the global design decision should not depend on scenarios.

There are several possible ways to solve this mixed integer program, among which the traditional way is to relax the integrality constraints (4.4d) to get a lower bound, then do branching, bounding and cutting. But the CMND problem itself is already notoriously difficult to solve, this stochastic network design model is even harder because its dimension is multiplied by S, the size of the scenario set.

The scenario decomposition structure of model (4.4) leads us naturally to the relaxation of the non-anticipativity constraints.

4.2.1 Augmented Lagrangian relaxation

Since all design decisions y_{ij}^s are required to be boolean, the non-anticipativity constraints can be aggregated as follows:

$$\overline{y}_{ij} = \sum_{s \in \mathcal{S}} p^s y_{ij}^s \qquad \forall (i,j) \in \mathcal{A} \tag{4.5a}$$

$$\overline{y}_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A} \tag{4.5b}$$

Notice that without equation (4.5b), this aggregated constraint is not correct, since equation (4.5a) itself cannot prevent the scenario solutions y_{ij}^s from being different with each other.

The constraints (4.5a) can be relaxed by augmented Lagrangian relaxation, where a Lagrangian multiplier λ_{ij} is associated to each constraint. Once again, $\overline{y}_{ij}$ should be kept binary during the relaxation. Some rounding heuristics can be used to fullfill this binary requirement, but a more systematic method must

be carried out afterwards, like the branch and bound used in [8]. On the other hand, a binary $\overline{y}$ carries less information than a continuous one.

Here is another way to represent the non-anticipativity constraints:

$$y_{ij}^s = \overline{y}_{ij}, \forall (i,j) \in \mathcal{A}, \forall s \in \mathcal{S}, \tag{4.6}$$

where $\overline{y}_{ij} \in [0,1]$ is defined as the expected value of the design variables over all scenarios, as in Equation (4.5a).

Equations (4.6) imply that for each scenario s, its decision on arc (i,j) should be the same as that in the global decision. Equations (4.6) are different from Equation (4.5) in two points: first, $\overline{y}_{ij}$ in Equation (4.6) is not required to be binary; second, Equation (4.6) is the disaggregated version of Equation (4.5). Using Equation (4.6), the model (4.4) contains more constraints. However, when a scenario decomposition method like PHA is used to solve this model, the total computation complexity remains the same. The advantage of Equations (4.6) is that these constraints can be relaxed individually in the corresponding scenario problems, which makes it possible to penalize the difference between the scenario solutions and the global expectation individually in order to consolidate them into a global solution.

After relaxing the constraints (4.6) using the Augmented Lagrangian, the objective function of model (4.4) becomes

$$\min \sum_{s\in\mathcal{S}} p^s \Big(\sum_{(i,j)\in\mathcal{A}} f_{ij} y_{ij}^s + \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^{ks} + \sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^s (y_{ij}^s - \overline{y}_{ij}) + \frac{\rho}{2} \sum_{(i,j)\in\mathcal{A}} (y_{ij}^s - \overline{y}_{ij})^2 \Big). \tag{4.7}$$

However, the above equation is not very useful since the problem is still inseparable because of the existence of $\overline{y}_{ij}$. In the progressive hedging algorithm

proposed by Rockafellar and Wets [37], the average solution is taken from the previous iteration ($\overline{x}^{t-1}$ in Equation 4.2). If we replace $\overline{y}$ by the same value from the previous iteration, this value becomes a constant in the current iteration, which makes this model completely separable by scenarios.

Let t denote the iteration index of PHA, the modified objective function for each scenario s at iteration t can be written as

$$\min \sum_{(i,j)\in\mathcal{A}} f_{ij}y_{ij}^{st} + \sum_{k\in\mathcal{K}}\sum_{(i,j)\in\mathcal{A}} c_{ij}^{k}x_{ij}^{kst} + \sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^{st-1}(y_{ij}^{st} - \overline{y}_{ij}^{t-1}) + \frac{\rho^{t-1}}{2}\sum_{(i,j)\in\mathcal{A}}(y_{ij}^{st} - \overline{y}_{ij}^{t-1})^2. \quad (4.8)$$

The Lagrangian multipliers λ_{ij}^{st} and penalty ratio ρ^t are updated as follows:

$$\lambda_{ij}^{st} \leftarrow \lambda_{ij}^{st-1} + \rho^{t-1}(y_{ij}^{st} - \overline{y}_{ij}^{t-1}) \quad (4.9)$$

$$\rho^{t} \leftarrow \alpha\rho^{t-1} \quad (4.10)$$

where α is a constant greater than 1, and ρ^0 should be positive in order to make sure that $\rho^t \to \infty$.

Comparing equation (4.8) to its deterministic version, a linear and a quadratic term have been added. After opening the square term and dropping the constant terms in (4.8), we have

$$\min \sum_{k\in\mathcal{K}}\sum_{(i,j)\in\mathcal{A}} c_{ij}^{k}x_{ij}^{kst} + \sum_{(i,j)\in\mathcal{A}} (f_{ij} + \lambda_{ij}^{st-1} - \rho^{t-1}\overline{y}_{ij}^{t-1})y_{ij}^{st} + \frac{\rho^{t-1}}{2}\sum_{(i,j)\in\mathcal{A}}(y_{ij}^{st})^2 \quad (4.11)$$

Now we can take advantage of the integrality constraint on y_{ij}^{st}. Since they are required to be binary variables, we can drop the square sign and rearrange the

above equation as

$$\min \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^{kst} + \sum_{(i,j)\in\mathcal{A}} (f_{ij} + \lambda_{ij}^{st-1} - \rho^{t-1}\overline{y}_{ij}^{t-1} + \frac{\rho^{t-1}}{2}) y_{ij}^{st} \tag{4.12}$$

The above equation is in fact a CMND problem with modified fixed costs, and all previous research works on CMND problems can be used.

To clarify the idea, we focus on the fixed cost term and repeat it here. At iteration t the fixed costs are modified for scenario s as follows

$$f_{ij}^{st} = f_{ij} + \lambda_{ij}^{st-1} - \rho^{t-1}\overline{y}_{ij}^{t-1} + \frac{\rho^{t-1}}{2}, \forall (i,j) \in \mathcal{A} \tag{4.13}$$

Now, the subproblem for each scenario s at iteration t has its objective as follows

$$\min \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^{kst} + \sum_{(i,j)\in\mathcal{A}} f_{ij}^{st} y_{ij}^{st}. \tag{4.14}$$

At each iteration t, we can construct a feasible global design by opening all arcs that are used by at least one scenario. Formally speaking, denote *max design* for arc (i,j) at iteration t as

$$y_{ij}^{Mt} = \cup_{s\in\mathcal{S}} \{y_{ij}^{st}\}, \tag{4.15}$$

where $\cup$ is "or". As the hedging proceeds, the overall best integer solution should be recorded and updated, since it is possible that y^{Mt} is inferior to that of its previous iterations.

The progressive hedging algorithm combined with the Augmented Lagrangian relaxation is described in Algorithm 6.

Algorithm 6 Progressive hedging algorithm with Augmented Lagrangian

Initialization

$t \leftarrow 0$

$\lambda_{ij}^{st} \leftarrow 0 \ \forall (i,j) \in \mathcal{A}, \forall s \in \mathcal{S}$

$\rho^t \leftarrow \rho^0$

for all $s \in \mathcal{S}$ **do**

$f_{ij}^{st} \leftarrow f_{ij}, \ \forall (i,j) \in \mathcal{A}$

solve the corresponding CMND subproblem

$\overline{y}_{ij}^t \leftarrow \sum_{s \in \mathcal{S}} p^s y_{ij}^{st} \ , \ \forall (i,j) \in \mathcal{A}$

calculate and evaluate y^{Mt}

$bestSolution \leftarrow y^{Mt}$

while stopping criteria are not met **do**

$t \leftarrow t+1$

for all $s \in \mathcal{S}$ **do**

$f_{ij}^{st} \leftarrow f_{ij} + \lambda_{ij}^{st-1} - \rho^{t-1}\overline{y}_{ij}^{t-1} + \frac{\rho^{t-1}}{2}, \ \forall (i,j) \in \mathcal{A}$

solve the corresponding CMND subproblem

Update

$\overline{y}_{ij}^t \leftarrow \sum_{s \in \mathcal{S}} p^s y_{ij}^{st}$

$\lambda_{ij}^{st} \leftarrow \lambda_{ij}^{st-1} + \rho^{t-1}(y_{ij}^{st} - \overline{y}_{ij}^{t-1})$

$\rho^t \leftarrow \alpha \rho^{t-1}$

calculate and evaluate y^{Mt}

update $bestSolution$ if y^{Mt} gives current best

4.2.2 Heuristic fixed cost adjustment

By applying the Augmented Lagrangian relaxation to the non-anticipativity constraints of SND problems, and integrating this into the progressive hedging algorithm framework, we got in fact for each scenario a CMND problem with modified arc fixed costs. It is very natural to think about modifying the fixed costs more "intelligently" instead of using λ and ρ blindly.

At the end of each iteration t, we have $\overline{y}_{ij}^t \in [0,1]$, the expected value of binary design variables for arc (i,j) over all scenarios. This value can be viewed as a "trend" of opening or closing arc (i,j). A lower value of $\overline{y}_{ij}^t$ means that only a small portion of scenarios chose to open this arc, while a higher value means that the majority prefers to open it. In the case that $\overline{y}_{ij}^t$ is less than a threshold c^{low}, increasing the fixed cost of arc (i,j) has the potential of driving the subproblems to avoid using that arc. On the contrary, when $\overline{y}_{ij}^t$ is higher than a threshold c^{high}, in order to attract the subproblems to use arc (i,j), its fixed cost should be lowered.

Here comes the formal presentation of the heuristic fixed costs adjustment method:

$$f_{ij}^t = \begin{cases} \beta f_{ij}^{t-1} & \text{if } \overline{y}_{ij}^{t-1} < c^{low} \\ \frac{1}{\beta} f_{ij}^{t-1} & \text{if } \overline{y}_{ij}^{t-1} > c^{high} \\ f_{ij}^{t-1} & \text{otherwise,} \end{cases} \tag{4.16}$$

where β is a constant larger than 1, c^{low} and c^{high} are two constants such that $0 < c^{low} < 0.5$ and $0.5 < c^{high} < 1$, and f_{ij}^t represents the modified fixed cost of arc (i,j) at iteration t.

The above adjustment can be called "global", since the fixed cost modifications are made for all scenarios. Keeping in mind the aim of an unanimous design, this heuristic modification can be pushed even further. For each scenario s, if it

is too far from the average (for example, $y^{st} = 0$ and $\overline{y}_{ij} = 0.9$, or $y^{st} = 1$ and $\overline{y}_{ij} = 0.1$), we can enforce the adjustment to increase or decrease the fixed cost temporarily even more just for this scenario. This modification is done locally in the sense that it only affects scenario s at the current iteration. Formally speaking,

$$f_{ij}^{st} = \begin{cases} \beta f_{ij}^{t} & \text{if } |y_{ij}^{st-1} - \overline{y}_{ij}^{t-1}| \geq c^{far} \text{ and } y_{ij}^{st-1} = 1 \\ \frac{1}{\beta} f_{ij}^{t} & \text{if } |y_{ij}^{st-1} - \overline{y}_{ij}^{t-1}| \geq c^{far} \text{ and } y_{ij}^{st-1} = 0 \\ f_{ij}^{t} & \text{otherwise,} \end{cases} \tag{4.17}$$

where $0.5 < c^{far} < 1$ and $\beta > 1$ are two given constant parameters, and f_{ij}^{st} stands for the modified local fixed cost of (i, j) for scenario s at iteration t.

There is another possible heuristic local modification to the subproblems: if for current scenario s, the decision on arc (i, j) is in the majority, that is, $|y_{ij}^{st-1} - \overline{y}_{ij}^{t-1}| \leq c^{near}$, this decision should be kept by fixing y_{ij}^{st} to y_{ij}^{st-1}, where c^{near} is a constant between 0 and 0.5.

The progressive hedging algorithm with heuristic fixed cost adjustment is described in Algorithm 7.

4.3 Specification of the algorithm

Before we can implement Algorithms 6 and 7, there are several things that need to be defined, such as the stopping criteria and the subproblem solver. Also, the nature of this algorithm drives us to consider its parallel implementation.

Algorithm 7 Progressive hedging algorithm with heuristic cost adjustment

Initialization

$t \leftarrow 0$

for all $s \in \mathcal{S}$ **do**

$f_{ij}^{st} \leftarrow f_{ij}, \forall (i,j) \in \mathcal{A}$

solve the corresponding CMND subproblem

$\overline{y}_{ij}^{t} \leftarrow \sum_{s \in \mathcal{S}} p^s y_{ij}^{st}, \forall (i,j) \in \mathcal{A}$

calculate and evaluate y^{Mt}

$bestSolution \leftarrow y^{Mt}$

while stopping criteria are not met **do**

$t \leftarrow t+1$

$\forall (i,j) \in A$, modify f_{ij}^{t} globally using equation (4.16)

for all $s \in \mathcal{S}$ **do**

$\forall (i,j) \in \mathcal{A}$, modify f_{ij}^{st} locally using equation (4.17)

fix some y_{ij}^{st} if needed

solve the corresponding CMND subproblem

Update

$\overline{y}_{ij}^{t} \leftarrow \sum_{s \in \mathcal{S}} p^s y_{ij}^{st}, \forall (i,j) \in \mathcal{A}$

calculate and evaluate y^{Mt}

update $bestSolution$ if y^{Mt} gives current best

4.3.1 Stopping criteria and second phase

The stopping criteria in the PHA for linear stochastic problems (Algorithm 5) is that all scenario solutions converge to the average solution. But in stochastic integer problems like SND problem, this is not formally justified and nonconvergence is possible. We need therefore other stopping criteria such as total running time, total iteration number or non-improvement iteration number, often used in metaheuristic procedures.

In order to evaluate the convergence of the PHA for SND problems, we define for iteration t of PHA

max design for arc (i,j):	$y_{ij}^{Mt} = \cup_{s\in\mathcal{S}}\{y_{ij}^{st}\}$
minimum design for arc (i,j):	$y_{ij}^{mt} = \cap_{s\in\mathcal{S}}\{y_{ij}^{st}\}$
inconsistency of arc (i,j):	$D_{ij}^{t} = y_{ij}^{Mt} - y_{i,j}^{mt}$
inconsistency level:	$D^t = \sum_{(i,j)\in\mathcal{A}} D_{ij}^t$
active arc set:	$A^t = \{(i,j) \mid D_{ij}^t = 1\}$,

where $\cup$ is "or" and $\cap$ is "and". y_{ij}^{Mt} is equal to 1 if at least one scenario chooses to open arc (i,j), while y_{ij}^{mt} is equal to 1 only if all scenarios choose to open arc (i,j). D_{ij}^t represents whether the status of arc (i,j) is the same for all scenarios: $D_{ij}^t = 0$ means y_{ij}^{st} is the same, $D_{ij}^t = 1$ means they are not the same. The inconsistency level D^t is a non-negative integer whose maximum value is the total number of arcs. It stands for the number of arcs whose status have not been agreed upon by all scenarios. This level can be seen as a measure of convergence for iteration t. The active arc set A^t contains the arcs whose status is not yet the same.

Convergence based only on integer variables in stochastic integer programming is called *integer convergence* [33]. In our case, $D^t = 0$, $A^t = \emptyset$, or $y^{Mt} = y^{mt}$ all mean that an integer convergence is achieved, i.e., a network design that is agreed by all scenarios. However, this integer convergence can not be guaranteed

by the progressive hedging algorithm.

Imagine a simple example where two commodities k_1 and k_2 need to be transported on a network, and there are only two extreme scenarios s_1 and s_2 in the scenario set: scenario s_1 has $d_1 = 100$ and $d_2 = 0$, while scenario s_2 has $d_1 = 0$ and $d_2 = 100$. For each commodity, there is only one possible path to deliver it. No matter how we modify the fixed costs for each scenario, it is still impossible to convince s_1 to open the path for k_2, then pay the construction for it, and vice versa, unless negative costs are permitted in our model. Since each subproblem itself is an optimization problem solved toward its own optimality, and the global design is required to be integer, it is quite possible that PHA is unable to achieve integer convergence.

Furthermore, even when integer convergence is somehow achieved, there is no theoretical guarantee that it is the optimal design for the SND problem. The convergence rate is related to the intensity of the penalization on the difference between scenario solutions and average solution. If this punishment is too harsh, a compromise may be reached out quickly but with poor quality. From this point of view, Algorithm 6 has the advantage that it needs less parameters.

By the end of the progressive hedging algorithm, if the active arc set $A^t \neq \emptyset$, which means an integer convergence is not achieved, a second phase can be carried out. The problem becomes a restricted SND problem, where the "active" design variables are those in A^t that need to be decided, and all other arc status are fixed.

There are several ways to solve the restricted model in the second phase, such as enumeration or branch and bound on the active arc set. We can also consider tabu search with cycle-based neighborhood. The neighborhood can be restricted to the active arc set, and a move can be defined as deviating flows on the active arcs. As for the cost of a cycle, it can be defined as an expectation

of the estimated cycle costs over all scenarios. For the time being, we use the branch and bound algorithm of Cplex to solve the restricted SND problem.

4.3.2 Solving subproblems

PHA does not need exact optimal solutions to subproblems at each iteration t. On the contrary, one *should* solve them approximately [27]. So the tabu search algorithm proposed in [21] and re-implemented in Chapter 3 is a good choice.

Another choice for solving the subproblems is the path relinking algorithm proposed in [20]. By generating a path from y^{st-1} towards $\overline{y}^t$, good solutions could be found for subproblems. In order to use the continuous vector $\overline{y}$ as a guiding solution, some adaption of the original algorithm must be made.

We use the cycle-based tabu search algorithm of [21] and adapt it in the following three aspects: the initial solution, the candidate arc set and the initial tabu list.

Initial solution

There are two steps in the PHA algorithm where we need to consider the initial solution.

In the initialization step of PHA algorithm, each subproblem is solved from scratch, which means the initial solution for each subproblem is obtained by opening all design arcs, then solving the CMCF problem and closing the unused arcs. There are some other initial solutions that can be used.

The linear relaxation of the SND model gives us S identical continuous variables vector y^s with $y^s_{ij} \in [0,1]$, because we kept the non-anticipativity constraints during the relaxation. An initial feasible solution can be achieved by rounding up

all non-zero design variables to 1. But one cannot expect too much regarding its quality. Meanwhile, sometimes even the linear relaxation is too big to be solved exactly within a reasonable time limit.

The *wait-and-see* solution defined in Section 2.2.2 can give us a lower bound. This lower bound is achieved by solving for each scenario the deterministic network design problem, and taking the expectation of the total costs. The non-anticipativity constraints (4.4f) are ignored temporarily. This lower bound should be better than the linear relaxation, at the cost of more computational effort. An initial feasible solution can be obtained by constructing the max design vector y^{Mt}. However, this initial solution is computationally expensive since it solves to optimality each subproblem. Furthermore, the scenario solutions usually conflict with each other, which makes the initial solution quite poor.

Another initial solution can be constructed using the EEV solution defined in Section 2.2.2. The EEV solution can be obtained by solving one CMND problem with average demands. This may speed up the convergence of PHA, but at the risk of losing diversity of the scenario problems. It should be noted that this EEV solution is not necessarily feasible for the scenario problems. A restoration phase is needed before the tabu search process begins.

Another step that uses the initial solution in the PHA algorithm is at the beginning of each iteration t. The subproblems are solved by tabu search on the basis of the solution from the previous iteration, that is, y^{st-1} is used as the initial solution of tabu search at iteration t. In the PHA framework, we can take advantage of the global information in $\overline{y}^{t-1}$ and make a heuristic modification on the best solution found at the previous iteration. For example, open y^s_{ij} if the average is greater than a threshold c^{high}, and close it if the average is lower than c^{low}; or be even more strict by opening y^s_{ij} if $\overline{y}^{t-1}_{ij} \geq 0.5$, which makes all tabu search procedures solve each subproblem from the same initial point. The option

of how to modify the initial solution can be specified in the parameter file.

Candidate arc set

The candidate set $\mathcal{C}(\gamma)$ contains the arcs that are used to begin the construction of cycles. The idea behind this is that by redelivering γ amount of flow around a selected cycle, at least one arc in this cycle will change its status. In order to accelerate the integer convergence, we amend the way to build $\mathcal{C}(\gamma)$ as follows:

$$\mathcal{C}(\gamma) = \mathcal{C}_1(\gamma) \cup \mathcal{C}_2(\gamma),$$

where

$$\mathcal{C}_1(\gamma) = \{(i,j)^+ \mid y_{ij}^s = 0 \text{ and } u_{ij} \geq \gamma \text{ and } \overline{y}_{ij}^{t-1} \geq c^{low}\}$$
$$\mathcal{C}_2(\gamma) = \{(j,i)^- \mid y_{ij}^s = 1 \text{ and } x_{ij} = \gamma \text{ and } \overline{y}_{ij}^{t-1} \leq c^{high}\},$$

and x_{ij} means the total amount of flow on arc (i,j).

The first subset $\mathcal{C}_1(\gamma)$ consists of the residual arcs $(i,j)^+$ that meet the following three criteria simultaneously: (i,j) is closed in the current solution; its capacity u_{ij} is greater than γ; and the average decision on this arc $\overline{y}_{ij}$ is greater than a threshold c^{low} $(0 < c^{low} < 0.5)$. We would like to open the arcs that have a higher average design value, since the cycle that begins with $(i,j)^+$ will put flow on arc (i,j) and thus open the arc if this cycle is selected as the next move in the tabu search procedure. If $\overline{y}_{ij}$ is low, which means only a few scenarios opened this arc, we would rather keep it closed for this scenario.

As for the second subset $\mathcal{C}_2(\gamma)$, if the flow amount on a currently open arc (i,j) equals γ, and the average decision on this arc $\overline{y}_{ij}$ is less than a threshold c^{high} $(0.5 < c^{high} < 1)$, add $(j,i)^-$ to this subset, because the cycle that begins

with $(j,i)^-$ will close (i,j) if it is selected as the next move. If $\overline{y}_{ij}$ is high, which means the majority of scenarios chose to open this arc, we would rather keep it open for this scenario.

Initial tabu list

In Section 4.2.2, we mentioned that the "good" decisions can be fixed. One way to fix them in the tabu search procedure is to add these arcs to the initial tabu list. Their tabu status expires after `TabuTenure` iterations, which makes this fix operation a little bit "softer".

4.3.3 Parallel computation structure

The structure of Algorithms 6 and 7 leads us naturally to consider parallel computation, because solving each subproblem is an independent task. A master/slave structure for the proposed algorithms is given in Figure 4.1.

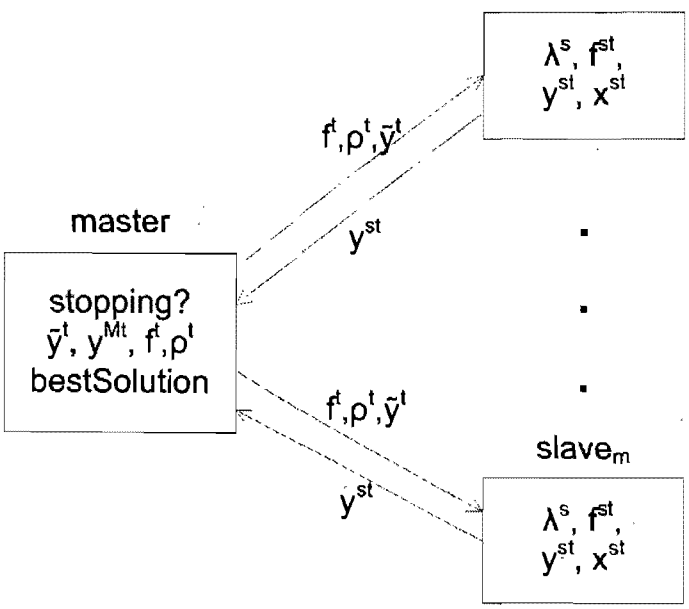

Figure 4.1: Parallel computation structure

As shown in Figure 4.1, the master process is responsible for testing the stopping criterion, calculating the average design $\overline{y}^t$, constructing and evaluating the max design y^{Mt}, and updating the best solution. In Algorithm 6, the master process updates ρ^t, while in Algorithm 7, it updates f_{ij}^t.

The slave processes are in charge of modifying the fixed costs locally and solving each CMND problem. A slave process holds the modified fixed costs f^{st}, the scenario solution y^{st} and x^{st}, and the Lagrangian multipliers λ^{st} when Algorithm 6 is used.

The information exchanges between master and slaves are also shown in figure 4.1. The master broadcasts $\overline{y}^t$, ρ^t and f^t, the slaves return y^{st}.

The synchronization is done at the end of each iteration t. Within each iteration, scheduling can be done statically or dynamically. For example, if there are 16 scenarios and 4 slave processes, static scheduling will assign scenarios 1 to 4 to process 1, scenarios 5 to 8 to process 2, etc. Dynamic scheduling will assign to each process one scenario, then the process that finished its task will be assigned another scenario, until there is no more scenario to solve. Dynamic scheduling should offer the advantage of a shorter total time and a better workload balance, since it takes different time to solve each subproblem.

Several variations can be done on this basis. It is possible to update the global information within an iteration t, instead of waiting until all scenarios are finished. Another possibility is not to solve all scenarios within each iteration, but only scenarios that have arc status in the minority group.

Scenario clustering is another furture research avenue. By grouping multiple scenarios into clusters, the size of subproblems gets bigger, but the quality of the global design will hopefully improve. The criterion for grouping the scenarios, as well as the subsolver that solves a scenario cluster need to be defined in the progressive hedging framework.

Chapter 5

Implementation of progressive hedging algorithm for stochastic network design

This chapter is similar to Chapter 3. It is a technical report for the implementation of the algorithms proposed in Chapter 4. We analyze the requirements, draw out the architecture design, and describe the class design. Numerical results will be given in Chapter 6.

5.1 Requirement analysis

The main function of this software is to solve the SND problem using the progressive hedging algorithm proposed in Chapter 4.

5.1.1 Scenario generation

In our stochastic network design model, demands are described by a scenario tree. The scenario tree must represent the underlying distribution of future demands in a good way, because the quality of the solution is directly linked to the quality of the scenario tree [28]: *garbage in, garbage out.* On the other hand, the number of scenarios must be limited for the stochastic program to be solvable. Pure sampling will not work in our case, since the size of the scenario tree grows exponentially with the number of commodities.

We derive the testing problems from the instances used in [23] for stochastic service network design (SSND) problems. These instances are generated for the time-dependent stochastic service network design problem. Each instance consists of two data files: one presents the network structure and commodity O-D pairs, the other provides scenarios of demands.

The network structure has T time periods, N nodes for each period, and N^2T arcs that connect all nodes of period t to those in period $(t+1)$ mod T. A commodity k is represented by an O-D pair in this time-dependent network structure, specified by $(o(k), t^{o(k)}) \rightarrow (s(k), t^{s(k)})$ where $o(k)$ is the origin node, $t^{o(k)}$ is the commodity available time period, $s(k)$ is the destination node, and $t^{s(k)}$ is the required delivery time period. Its format is given below:

```
NAME : file_name
NODES : number_of_nodes
PERIODS: number_of_time_periods
OUTSOURCING : outsourcing_cost
CAPACITY : vehicle_capacity
COMMODITIES : number_of_commodities
VEHICLES : number_of_vehicles
SCENARIOS : number_of_scenarios
RELOADCOST : reload_cost
MARGINALTRUCKCOST: marginal_truck_cost
COST :
(for each node i: ) (for each node j:) cost_i_j
DISTANCE :
(for each node i: ) (for each node j:) distance_i_j
ORIGIN :
(for each commodity:) commodity_origin
DESTINATION :
(for each commodity:) commodity_destination
```

```
AVAILABLE :
(for each commodity:) available_time_period
DELIVERED :
(for each commodity:) deliver_time_period
EOF
```

The scenario data file consists of S rows, each stands for one scenario containing its probability p^s followed by K demand values. Its format is given below:

```
number_of_scenarios
(for each scenario:) probability (for each commodity :) demand
```

We used this data set as a starting point to derive our problems in the following way: We ignored the time periods and interpret an instance as a network with N nodes and N^2 arcs. The value of `vehicle_capacity` is used as the unique arc capacity, the `cost_i_j` is used as the routing cost of arc (i, j). We assigned a unique value of 100 to the arc fixed cost. For each commodity, we use its origin and destination node information, but ignore its available time period and the time period it is required to be delivered. An artificial arc is added for each commodity with `outsourcing_cost` as its routing cost and zero fixed cost, which means if the current network design is not able to fulfill the commodity's demand, some ad-hoc capacity increasing is needed, see [32] for the definition of this capacity increase. The addition of artificial arcs make sure that any network design is feasible.

After the re-interpretation, however, some problems arise. First, the network structure becomes a complete network with identical arc fixed costs and capacities, which is not common in practice. Secondly, in the original instances, there exists *self circle commodity* whose origin and destination are the same nodes $(o(k) = s(k))$, but in different time periods $(t^{o(k)} \neq t^{s(k)})$. It is meaningful in the time-dependent network, which simply means to keep the commodity at the

node. But in our model, this kind of commodity violates the flow conservation constraints and thus makes the instance infeasible. Last but not least, the current implementation of our tabu search algorithm does not support *parallel commodities* ($o(k_1) = o(k_2)$ and $s(k_1) = s(k_2)$), since the restoration from an infeasible solution is done on the basis of artificial arcs, while two artificial arcs are not distinguishable if they are constructed for parallel commodities. Some modification must be done to solve this kind of instances. Because we use these SSND instances temporarily, instead of modifying our implementation, we chose to slightly modify the instances to avoid self circles and parallel commodities. Denote this data set **S**.

Thanks to the generosity of Doctor Kaut and Professor Wallace, we could generate scenarios for the CMND problems using their scenario generation program based on [25]. We use the deterministic instances from the data set R of [21] as network structure, and generate the scenario tree for these instances. In order to generate the scenario tree, we must specify for each random variable its distribution and the correlation between each pair of random variables.

We assume the future demands follow a triangular distribution, as often used in business decision making when little information is available. Three values need to be specified for such a distribution: min a, max b and mode c. In the deterministic instances, we have a given demand value $d(k)$ for each commodity k, which can be used as the mode c, representing the most likely outcome. We set the min $a = 0$ and the max $b = 1.25c$, but this risks of yielding infeasible scenarios. So we need to check the feasibility of each scenario and decrease some demand value for infeasible scenarios before we solve the stochastic design problem.

We assume that the correlation between two demands is linear, which means the correlation coefficient r_{ij} between demand d_i and d_j should always lies between -1 and 1. $r_{ij} = 1$ if $i = j$, $r_{ij} = 0$ if d_i and d_j are independent. In real life, these

demands are not independent. The commodities that share the same origin or the same "origin area" are often correlated. We would like to see how the behavior of progressive hedging algorithm is influenced by different levels of correlations. Thus, another task of our implementation is to group the demands for a given network structure and prepare the required distribution and correlation input files for scenario generation.

Since the executable program of scenario generation provided by the authors of [25] is compiled under Windows 32 system, while we are using Linux system in our implementation, we cannot run the scenario generation program within our code. Text files are used to exchange the distribution and correlation requirements, as well as the resulting scenario instances. Figure 5.1 shows how our code interacts with the scenario generation program.

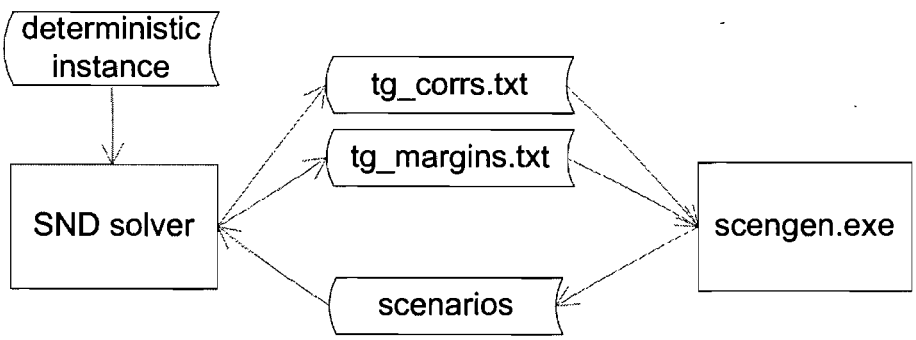

Figure 5.1: System interface for stochastic network design

5.1.2 Command line format

In the command line, we provide the flexibility of specifying which progressive hedging algorithm to use (Augmented Lagrangian or heuristic fixed cost adjusting), and how to solve the subproblems (Cplex or tabu search). Besides this main work, to generate scenarios, the program is able to output the distribution and correlation files and do feasibility checking. Furthermore, in order to evaluate the

proposed algorithms, the following methods are implemented: solving the model as a mixed integer problem using Cplex; finding the lower bound using the linear relaxation; computing the lower bound and initial solution of the wait-and-see solution; calculating the EEV solution (*expected result of using the EV solution*). We include also the deterministic methods implemented in Chapter 3, so that each scenario can be solved individually. The command line has the following format

```
./main option paramFileName dataFileName [scenFileName]
```

and the options are given below:

```
List of options:
-df : solve CMCF using cplex to check feasibility
-dc : solve CMND using cplex
-dt : solve CMND using tabu
-sc : solve SCMND using cplex
-sl : solve SCMND's linear relaxation using cplex
-se : solve SCMND's expected value solution using cplex
-pc : solve SCMND using Lagrangian PHA with cplex as subsolver
-pt : solve SCMND using Lagrangian PHA with tabu as subsolver
-hc : solve SCMND using heuristic PHA with cplex as subsolver
-ht : solve SCMND using heuristic PHA with tabu as subsolver
-sgN : generate correlation and distribution files for scenGen.exe
       N in -sgN is the correlation level
```

The wait-and-see solution can be obtained using option `-pc` with iteration number set to 1.

5.1.3 Parameter file format

There are several parameters that need to be specified by the user to control the algorithms, including tabu search parameters and progressive hedging parameters. We use the same parameter file format as in Chapter 3, and add the following parameter settings.

```
#================= Stochastic Strategy Parameters
PHATotalIter = 30
PHANonImproveIter = 5
PHATotalSec = 36000

#====parameters in PHA with augmented Lagrangian
# Increasing rate of rho
Alpha = 1.1;

#====parameters in heuristic PHA for fix cost adjusting
LocalFixCostAdjust = 1;

# Upper and lower thresholds used to decide arc status
CHigh = 0.8;
CLow = 0.2;

# Increasing or decreasing factor of fix cost
Beta = 1.1;

# The constant used to compare local design to average
#CNear is used to fix good local design decisions
CNear = 0.2
#CFar is used to modify local fix cost in heuristic PHA
CFar = 0.7
```

```
#=====parameters for the modified tabu search
#Initial config of tabu search
#0 means using old config; 1 means using Upper/Lower thresholds;
#2 means using 0.5 to guess
InitialConfigGuess = 1;

#Initial tabu list
#0 means empty list; 1 means put agreed arcs into tabu list
InitialTabuList = 0;
```

5.1.4 Parallel implementation

The parallel computation structure specified in 4.3.3 can be implemented in different ways. Our first issue is how to develop the parallel program: explicit or implicit parallel programming? Explicit parallel programming requires the programmer to explicitly specify how the processors will cooperate, also called hand threading, such as POSIX Threads (PThreads) [7], while implicit parallel programming asks the compiler to create and manage threads, such as OpenMP (*Open Multi-Processing*)[30]. Comparing to PThreads, OpenMP takes parallel programming to a higher level. Using OpenMP, we can produce elegant code that is easier to understand and maintain. Besides, OpenMP scales well with the number of processors. The program will scale the number of threads when running on a platform where more processors are available.

The next decision is how to share information among processors: shared-address-space or message-passing? In the shared-address-space programming paradigm, such as OpenMP [10], programs can be viewed as a collection of processes accessing a central pool of shared variables. While in the message-passing programming paradigm such as MPI (*Message Passing Interface*)[22], there are

no shared variables, each processor uses its local variables, and occasionally sends or receives data from other processors [30]. Using OpenMP, the parallel programs are limited to run on a group of computers that have a shared address space, which is usually a multi-CPU computer; while using MPI, the parallel programs can be run on a larger cluster, or even on heterogenous networks.

Although OpenMP limits the parallel level, it does help the programmers to focus on the algorithm itself, because the information exchange is done through the shared-address-space, which is accessible to all threads, just like in the sequential programs. Furthermore, the OpenMP parallel code can be run as a sequential one without any modification. This makes the debugging much more easier, because in our algorithm, each slave process needs a Cplex license during the tabu search procedure to solve the corresponding CMCF problem, and sometimes the requirement of several licenses at the same time is too difficult to meet.

Last but not least, the nature of our algorithm is suitable for OpenMP. In the progressive hedging algorithms, the most time-consuming part is to solve the modified subproblem for each scenario. This can be done by several slave processes independently. At the end of the *for each* $s \in \mathcal{S}$ loop, control is given back to the master process. This is the classical OpenMP parallel loop structure. It can be parallelized easily with a compiler directive.

5.2 Architecture design

On the basis of the implementation work in Chapter 3, we need to add a `Stoch` package to specify the progressive hedging algorithms. Also, scenario information should be added to the `Graph` package, and the EEV solver, linear relaxation and MIP solver should be added to the `MIP` package. The `Main` package deals with

command line options, reads and constructs *Graph* and reads parameters. The other packages are untouched.

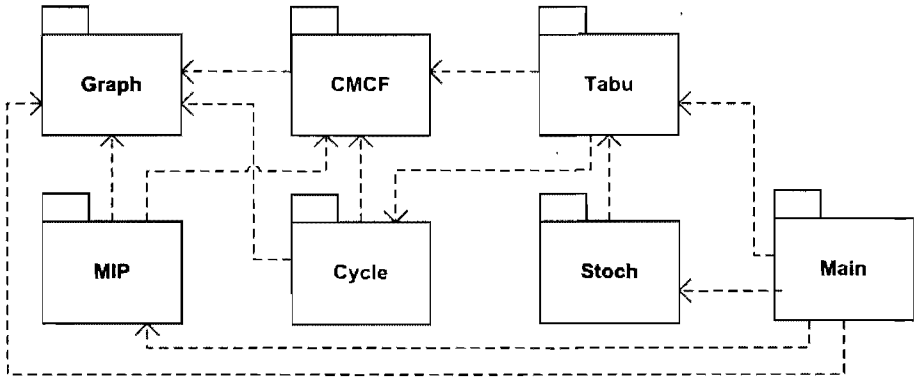

Figure 5.2: Modified architecture for stochastic network design

5.3 Class design and implementation

5.3.1 Modified `Graph` package

As shown in figure 5.3, a *Scenario* class is added to this package. The *Scenario* class contains its probability and a vector of demands. The *Graph* class is modified by adding a vector *scenarios[S]* and a function *createScens()*. The latter reads a given scenario data file and stores the data in the former.

In the proposed algorithm, the arc fixed costs are modified globally and locally. This modification should not be done directly on *NetworkArcs*, since we need the original arc fixed costs to evaluate a design. So another vector *fixedCosts[A + K]* is added to the *Graph* class, in order to hold the modified fixed costs.

Furthermore, in order to allow slave threads to modify the *Graph* in parallel without conflicting each other, a copy constructor should be provided. Each slave thread modifies and solves its local copy.

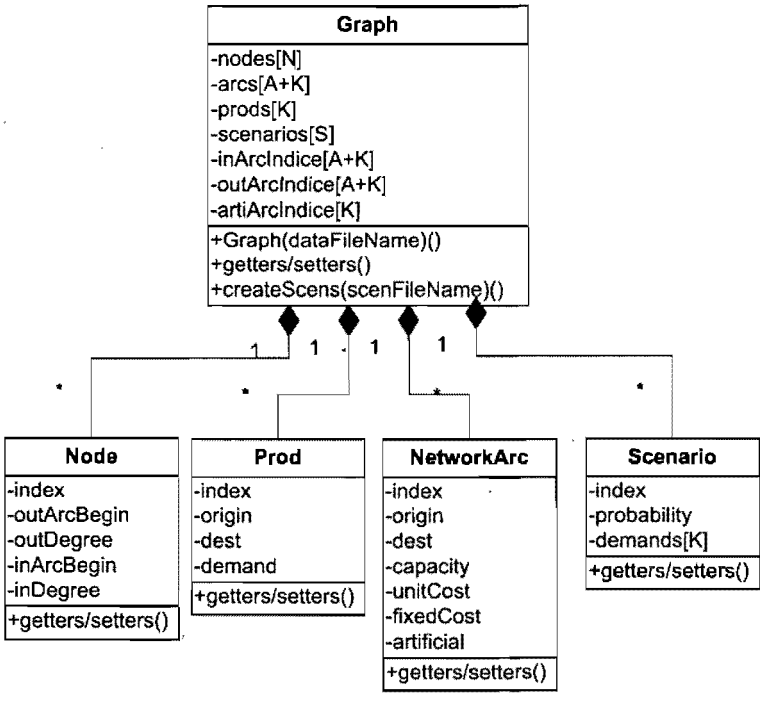

Figure 5.3: Modified Graph package design

5.3.2 Stoch package

Figure 5.4 presents the classes in Stoch package. The *ModifiedTabuSolver* class derives from *TabuSolver*. It contains two vectors as data fields: *avgY* holds the average design value $\overline{y}_{ij}$, *activeStatus* keeps the arc inconsistency information D_{ij}^{t}. It overwrites three *virtual* functions of its father class: *buildNeighborhood()* creates a set of residual arcs that are used to begin cycle construction; *createGamma()* builds the residual value set Γ; *initialModification()* modifies the initial config and initial tabu list as specified by the parameter file.

Here we used the polymorphism technique to reuse the *TabuSolver* class. By claiming these three functions as *virtual* functions, only the modified parts need to be implemented, the other functions like *localSearch()*, *restore()*, *intensify()* stay untouched. Notice that we need to change the *private* functions in *TabuSolver* into *protected*, in order to allow the inherited class to access them.

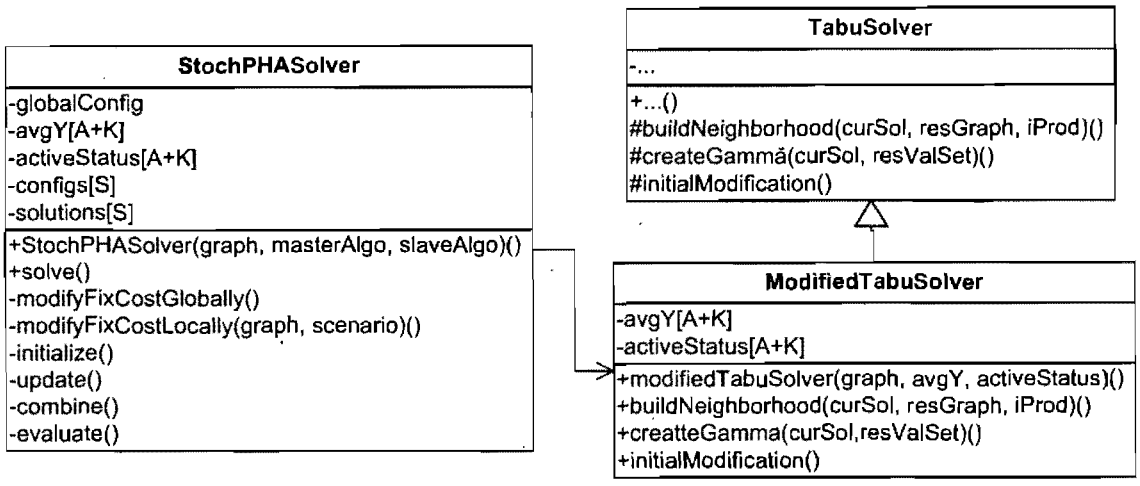

Figure 5.4: `Stoch` package design

The *StochPHASolver* class is the kernel of this implementation. As the master thread, it keeps the global *Config*, the average design vector $avgY$ and the inconsistency status vector *activeStatus*. The vectors *configs* and *solutions* are shared by slave threads to store for each scenario the design and flow decisions.

The constructor of *StochPHASolver* takes the *Graph*, *masterAlgo* and *slaveAlgo* as parameters. *masterAlgo* can be 'p' for augmented Lagrangian PHA or 'h' for heuristic adjusting PHA; *slaveAlgo* can be 'c' for Cplex or 't' for tabu search. Its public function *solve()* does the progressive hedging until some stopping criterion is met.

There are six private functions defined in this class. *modifyFixCostGlobally()* modifies f_{it}^t at each iteration t according to the specified method; *modifyFixCostLocally()* modifies f_{ij}^{st} for scenario s on the given copy graph; *combine()* constructs the feasible global design y^{Mt}; *initialize()* solves each original subproblems in parallel; *update()* solves the modified subproblems in parallel. *evaluate()* calculates the objective function for a given design, which is the sum of fixed costs and the expectation of routing costs over all scenarios. Notice that the routing cost for scenario s is obtained by solving the corresponding CMCF problem using original

arc fixed costs, and this step can also be parallelized.

5.3.3 Modified `MIP` and `Main` package

We add several independent solvers to the `MIP` package. *StochEEVSolver* solves at first the EV solution (the CMND problem with expected demands), then evaluate this design and output the EEV solution. *StochMIPSolver0* and *StochMIPSolver1* represent models (2.4) and (2.5) respectively. They solve the model as a mixed integer program using Cplex. *StochLPSolver0* and *StochLPSolver1* inherit from *StochMIPSolver0* and *StochMIPSolver1* respectively, they relax the integrality constraints on the given model and output the lower bound.

In the *Main* package, the static class *Parameter* is modified to deal with more parameters. Class *ScenGenerator* groups the demands of the *Graph*, then outputs two text files to specify the distribution and correlation of the random demands. These two text files are then used by the scenario generation program [28]. Feasibility checking for each scenario s is done by simply opening all arcs and solving the corresponding CMCF problem by Cplex. A scenario is infeasible if the solution has positive flow on artificial arcs.

5.4 Conclusion

From the software engineering point of view, the implementation of the proposed metaheuristic algorithm tested the reusability of the re-implemented software of the cycle-based tabu search. In fact, we found that it is quite easy to extend the functions of the current software.

There is an important remark from our debugging process that we need to mention here. We chose OpenMP to implement this parallel algorithm because

it is convenient, it allowed us to develop and test the proposed algorithm easily. Unfortunately, we found that the Concert technology of ILOG Cplex does not work well with OpenMP. We tested the following simple program, and it crashed as soon as the number of threads was set to more than 1.

```
#include <ilcplex/ilocplex.h>
#include <omp.h>
const int bignumber=4;

int main(int argc, char** argv)
{
#pragma omp parallel for
for (int i=0; i<bignumber; i++)
   {
        IloEnv env;
        IloModel model(env);
        IloCplex cplex(model);
        cplex.end();
        model.end();
        env.end();
   }

return 0;
}
```

We contacted the technical support of ILOG Cplex, they confirmed our observation and promised more parallel support in the future. For the time being, we simulated the shared memory in the master class, created the models sequentially, and kept all pointers in the master class. A future implementation using

MPI is needed in order to run the algorithm in parallel on larger clusters or on heterogenous networks.

Chapter 6

Experimental results and analysis

In this chapter, we report the experimental results and analyze the performance of the proposed algorithms. Experiments were conducted on a Sun Fire X4100 cluster of 16 computers, each has two 2.6 GHz Dual-Core AMD Opteron processors and 8192 Megabytes of RAM, operating under Solaris 2.10. For the parallel experiments, the number of parallel threads is set to 4. Computation times are wall-clock time in seconds. The Cplex solver version is 10.1.1.

Experiments are performed on two instance sets, denoted **S** and **R** respectively.

The instance set **S** consists of 16 problems derived from the instances used in [23] for the time-dependent stochastic service network design problems. As described in Section 5.1.1, each instance is a complete graph with unique arc fixed cost and arc capacity. The routing costs are arc-specific for each commodity. An artificial arc is added for each commodity with much higher routing cost and zero fixed cost, which means that if the current network design is not able to fulfill the demands, some other high-cost backup transportation methods have to be used. Detailed numerical results are reported in this chapter for the 16 instances in this set, including the results of solving the problems as mixed integer programs using the branch and bound algorithm of Cplex, their linear relaxations, the measures of EVPI and VSS, as well as the performance of the two progressive hedging algorithms proposed in Chapter 4.

The instance set **R** contains 180 problems. These problems are generated for 3 groups of deterministic network design problems with 5 different combined levels of fixed cost and capacity ratios selected from the instance set **R** used in [21]. For each deterministic instance, we generated 16, 32 and 64 scenarios for 3 different levels of positive correlations using the program provided by the authors of [23]. Thus, we have a total of $3 * 5 * 3 * 3 = 135$ instances in this set. We compared the best solutions obtained by the branch and bound algorithm of Cplex and those obtained by using our progressive hedging algorithms. Results for this set are listed in the Appendix and summarized in this chapter.

6.1 MIP and LP results

Table 6.1 displays the results of solving the instances in set S as mixed integer programs and their linear relaxations.

Table 6.1: Problem set S, solving as MIP and LP

	Branch & Bound				Linear Relaxation			
Prob	Opt	Time	LB	Gap	Linear	Time	Int	Diff
16N14C10S	4909.3	11.5	4909.3	0.00%	4228.7	0.4	5225.2	6.43%
16N14C20S	4990.1	47.3	4990.1	0.00%	4376.0	1.3	6414.8	28.55%
30N14C10S	5198.6	10.8	5198.6	0.00%	4523.5	1.2	5199.9	0.03%
30N14C20S	5218.6	25.7	5218.6	0.00%	4612.9	4.1	5630.2	7.89%
16N40C20S	15184.9	4569.6	15184.9	0.00%	13709.1	21.5	19611.6	29.15%
16N40C60S	15244.7	36006.4	15112.8	0.87%	13767.5	206.0	19798.0	29.87%
16N40C90S	15204.8	36019.3	15103.9	0.66%	13762.6	302.8	19896.6	30.86%
30N40C20S	14301.0	36009.9	14056.5	1.71%	12739.6	92.8	18508.2	29.42%
30N40C60S	14723.1	36029.0	13409.3	8.92%	12788.3	1304.7	19003.1	29.07%
30N40C90S	14723.0	36048.6	12787.0	13.15%	12787.0	2720.8	19110.5	29.80%
16N80C20S	27167.5	36013.6	26773.0	1.45%	24576.4	100.2	33208.7	22.24%
16N80C60S	28621.4	36030.8	26330.8	8.00%	24652.1	1028.5	34018.7	18.86%
16N80C90S	28621.1	36032.7	25709.6	10.17%	24659.4	1863.3	33531.5	17.16%
30N80C20S	31408.3	36027.3	29303.9	6.70%	27385.7	1911.4	40872.6	30.13%
30N80C60S	31412.7	36051.4	27491.8	12.48%	27491.8	22128.2	43592.4	38.77%
30N80C90S	31412.4	72106.8	27473.0	12.54%	27473.0	34870.4	42401.0	34.98%

The problems are identified in the first column by the number of nodes, commodities and scenarios. Since the instances in this set are complete graphs, the number of arcs is N^2.

The results in the columns under `Branch&Bound` are obtained by solving the instances as mixed integer programs. The `Opt` column corresponds to the best integer solution found by the branch&bound algorithm of Cplex within a time limit of 10 hours. For the last instance *30N80C90S*, the procedure failed to produce a feasible solution within this time limit. In order to facilitate further calculations of relative difference, we increase the time limit to 20 hours for this instance, and obtained a feasible solution. The `LB` column represents the lower bound found by Cplex, and the following column `Gap` indicates the optimality gap between the best integer solution and the lower bound.

The figures in the columns under `Linear Relaxation` are obtained by solving the linear relaxation of the problems. The `Linear` column corresponds to the value of the linear relaxation of the SND problem. The column `Int` represents the integer solution obtained by rounding up all non-zero design variables. The `Diff` column indicates the relative difference between this integer solution and the best integer solution found by Cplex.

A first conclusion that emerges from Table 6.1 is that stochastic network design problems are indeed difficult to solve. As indicated by the performance of a state-of-the-art mixed integer programming solver, most of the instances cannot be solved to optimality within 10 hours, except for the instances in the first group, which are relatively trivial. For some larger instances, the optimality gaps are more than 10% after 10 hours of computation.

The experimental results also indicate clearly that the linear relaxation can provide little help. On the one hand, the relaxation values cannot be used as good lower bounds; on the other hand, the integer solutions obtained from this

relaxation have very poor quality.

The same experiments are conducted on the 135 instances of set **R** and the same conclusions can be derived. The best integer solutions found by Cplex are given in Appendix in order to compare with the results obtained by our progressive hedging algorithms.

Another observation is the impact of the correlation levels on the results. Basically speaking, the higher the correlation level, the longer it takes Cplex to solve the problem, and the higher the optimal solution value. Three different correlation coefficients (r=0, r=0.2, r=0.8) are used to generate the scenarios, thus we have 3 groups of instances, each containing 45 problems. Compared to the results of the first group (r=0), 31 instances in the second group (r=0.2) have higher `Opt` values, while 35 instances in the third group (r=0.8) have higher `Opt` values. This trend is more clear when the problem becomes harder to solve.

6.2 EVPI and VSS results

Before solving these stochastic network design problems, we would like to measure their value of stochasticity. The EVPI and VSS values for the instance set S are reported in Tables 6.2 and 6.3, respectively.

In Table 6.2, the column `WS` represents the *wait-and-see* value, which is calculated as the expectation of total costs over all scenario problems. Each subproblem is solved by Cplex within a time limit of 1 hour. If the subproblem cannot be solved to optimality within the time limit, the value of `WS` is calculated using the best integer solution, while the `WS` value is computed as the expectation of the lower bounds found by Cplex.

The column `Time` corresponds to the wall-clock time when the subproblems

Table 6.2: Problem set S, EVPI

Prob	WS	$\underline{WS}$	Time	EVPI	Int	Diff	D
16N14C10S	4896.5	4896.5	5.2	12.8	5209.3	6.11%	8
16N14C20S	4982.0	4982.0	10.3	8.1	5290.1	6.01%	8
30N14C10S	5198.6	5198.6	19.5	0.0	5198.6	0.00%	0
30N14C20S	5218.6	5218.6	37.7	0.0	5218.6	0.00%	0
16N40C20S	15086.5	15086.5	3006.9	98.4	16435.7	8.24%	35
16N40C60S	15062.1	15062.1	5518.3	182.6	16725.8	9.72%	43
16N40C90S	15064.7	15064.7	9224.5	140.1	17137.0	12.71%	48
30N40C20S	14083.0	13829.1	18149.1	471.9	15724.4	9.95%	34
30N40C60S	14084.2	13828.0	54568.1	895.1	15735.8	6.88%	35
30N40C90S	14079.8	13830.1	83882.3	892.9	16023.0	8.83%	40
16N80C20S	26773.1	26416.1	18136.7	751.4	31351.9	15.40%	90
16N80C60S	26790.0	26405.9	54204.0	2215.5	32926.6	15.04%	113
16N80C90S	26785.6	26407.7	83241.0	2213.4	33421.4	16.77%	116
30N80C20S	30528.9	29311.0	18415.1	2097.3	39153.4	24.66%	142
30N80C60S	30741.3	29301.3	55081.6	2111.4	41412.7	31.83%	170
30N80C90S	30635.2	29311.8	84041.1	2100.6	46312.4	47.43%	222

are solved in parallel using 4 processors. The EVPI column holds the difference between the best solution found by Cplex (the Opt column of Table 6.1) and the value of $\underline{WS}$. For the instances whose subproblems are not solved to optimality, the value in the column EVPI is an estimation of the *expected value of perfect information.*

The column Int reports the integer solution value obtained by opening all arcs that carry flows in at least one scenario solution. The next Diff column displays the relative difference between this integer solution and the best solution found by Cplex. The last column D represents the inconsistency level defined as the number of arcs that are not agreed upon by all scenarios.

The results displayed in Table 6.2 indicate that the stochastic information has a relatively high value, except for some small instances. Another conclusion is that the integer solution constructed is time consuming and has very poor quality. The inconsistency level reported in column D reveals the challenge of finding a

compromise between different scenario solutions into one global design.

Table 6.3: Problem set S, VSS

Prob	EV	EEV	Time	Diff	VSS
16N14C10S	4909.3	4909.3	1.5	0.00%	0.0
16N14C20S	4990.1	4990.1	0.9	0.00%	0.0
30N14C10S	5198.6	5198.6	2.5	0.00%	0.0
30N14C20S	5218.6	5218.6	3.4	0.00%	0.0
16N40C20S	15061.0	15796.7	10.7	4.03%	611.8
16N40C60S	15107.5	15360.8	24.7	0.76%	248.0
16N40C90S	15063.3	15944.9	20.0	4.87%	841.0
30N40C20S	14108.1	14346.6	36034.5	0.32%	290.1
30N40C60S	14110.3	14367.2	19258.8	-2.42%	957.9
30N40C90S	14110.2	14338.6	3915.9	-2.61%	1551.6
16N80C20S	26770.4	27812.7	36096.0	2.37%	1039.7
16N80C60S	26783.4	28242.0	36130.0	-1.33%	1911.2
16N80C90S	26773.8	28267.8	36161.4	-1.23%	2558.2
30N80C20S	30286.2	31464.9	36185.1	0.18%	2161.0
30N80C60S	30224.5	31372.8	36100.1	-0.13%	3881.0
30N80C90S	30187.8	31061.0	36123.7	-1.12%	3588.0

Table 6.3 reports the measures of the value of the stochastic solution for the instance set S. The column EV represents the optimal solution of the *expected value problem*, which is obtained by replacing the random demands with its expected value, and solved by Cplex within a time limit of 10 hours. The EEV column corresponds to the *expected result of using the EV solution*, which is obtained by using the EV solution as a global design and solving for each scenario the related CMCF problem. The Time column shows the computation time in seconds.

The Diff column illustrates the relative difference between the EEV value and the best solution found by Cplex. The last column VSS indicates the difference between the EEV value and the lower bound found by Cplex (the column LB of Table 6.1). For the instances whose EV problem is not solved to optimality, the value in the column EEV is an estimation of the *value of the stochastic solution.*

Our first observation from Table 6.3 is that some EEV solutions are better

than the best solution found by Cplex, which is quite out of intuition. A further research on the flow distribution for scenario problems revealed the reason: for some scenarios, it is possible that the given *expected value design* is incapable of delivering the required demands for all commodities, in which case the artificial arc is used to carry flow. As indicated in 5.1.1, the artificial arc represents ad-hoc capacity increase [32]. The existence of the artificial arc makes sure that any design is feasible, including the expected value design. Recall that for the instances in set S, the routing cost of artificial arc is set to the `outsourcing_cost` whose value is 330, while the average routing cost for design arcs is about 50. The relative small difference between the artificial routing cost and "normal" routing costs leads to the acceptable EEV cost. With higher artificial routing costs, the EEV value should have been much higher. The value of the artificial routing cost also has a direct influence on the VSS value, which can be explained intuitively that the more expensive the ad-hoc capacity increase, the higher the cost of ignoring uncertainty.

For the instances in set R, artificial arcs are added with very high routing cost ($1E10$) in order to indicate that no ad-hoc capacity increase is available besides the given design arcs. For most instances in set R, the *expected value designs* are infeasible for one or more scenarios, which makes the EEV value very high.

A conclusion from the VSS calculation is that for the network design problem, this value depends on the artificial routing cost. In the case that no ad-hoc capacity increase is available, the EEV solution is out of interest because of the high risk of having infeasible design for some scenarios.

6.3 PHA1 and PHA2 results

In this section, we report the experimental results of the progressive hedging algorithms proposed in Chapter 4. We denote PHA1 the PHA with Augmented Lagrangian and PHA2 the PHA with heuristic cost adjustment, respectively. The results of PHA1 are given in Table 6.4 and those for PHA2 are given in Table 6.5.

Table 6.4: Problem set S, PHA with Augmented Lagrangian

	Final			Phase I				
Prob	Best	Time	Diff	Best	Time	Diff	Iters	D
16N14C10S	4909.3	20.4	0.00%	5009.3	20.1	2.04%	10	3
16N14C20S	4990.1	39.3	0.00%	5090.1	38.7	2.00%	11	3
30N14C10S	5198.6	161.2	0.00%	5298.6	160.3	1.92%	11	2
30N14C20S	5218.6	357.8	0.00%	5318.6	355.9	1.92%	27	2
16N40C20S	15243.1	244.4	0.38%	15922.4	241.1	4.86%	34	11
16N40C60S	15196.3	686.8	-0.32%	16324.8	672.5	7.09%	27	24
16N40C90S	15194.3	1112.0	-0.07%	17124.6	1057.6	12.63%	24	33
30N40C20S	14498.9	8018.9	1.38%	15121.2	8012.4	5.74%	25	14
30N40C60S	14350.1	11998.7	-2.53%	15623.1	11957.7	6.11%	23	25
30N40C90S	14321.4	13802.0	-2.73%	15623.0	13716.5	6.11%	23	29
16N80C20S	27464.4	342.4	1.09%	29918.2	294.7	10.12%	37	46
16N80C60S	27272.0	12977.7	-4.71%	31121.4	843.9	8.73%	31	74
16N80C90S	27347.4	31349.2	-4.45%	31121.1	1297.9	8.73%	35	75
30N80C20S	31010.6	13023.5	-1.27%	32708.3	13007.0	4.14%	23	30
30N80C60S	30874.2	19048.1	-1.71%	33712.7	18926.2	7.32%	31	54
30N80C90S	30704.5	17236.2	-2.25%	33812.4	16933.2	7.64%	24	57

The parallel progressive hedging algorithms stop after one of the following three criteria is met: the total iteration number is 50, the non improvement iteration number is 10, or the wall-clock running time is over 10 hours. A second phase then follows, which solves the restricted stochastic network design problem sequentially as a mixed integer program using Cplex within a time limit of 30000 seconds.

A calibration phase could be helpful in determining the appropriate values for

Table 6.5: Problem set S, PHA with heuristic adjustment

	Final			Phase I				
Problem	Best	Time	Diff	Best	Time	Diff	Iters	D
16N14C10S	4909.3	20.3	0.00%	5009.3	20.0	2.04%	10	3
16N14C20S	4990.1	39.2	0.00%	5090.1	38.5	2.00%	10	3
30N14C10S	5198.6	154.8	0.00%	5198.6	154.7	0.00%	6	0
30N14C20S	5218.6	352.9	0.00%	5318.6	351.0	1.92%	25	2
16N40C20S	15243.1	227.3	0.38%	16422.4	223.0	8.15%	22	23
16N40C60S	15196.3	743.9	-0.32%	17124.8	715.3	12.33%	38	31
16N40C90S	15194.3	1216.8	-0.07%	17224.6	1162.5	13.28%	40	34
30N40C20S	14321.9	7956.6	0.15%	15121.2	7948.9	5.74%	17	24
30N40C60S	14317.9	12117.8	-2.75%	15923.1	12063.9	8.15%	34	29
30N40C90S	14287.8	13956.3	-2.96%	16223.0	13825.3	10.19%	26	37
16N80C20S	27359.9	795.1	0.71%	30718.2	290.8	13.07%	39	56
16N80C60S	27190.2	30799.5	-5.00%	32321.4	776.6	12.93%	27	88
16N80C90S	27371.2	31478.3	-4.37%	31721.1	1412.3	10.83%	45	76
30N80C20S	30913.5	13037.0	-1.58%	32608.3	13019.8	3.82%	50	30
30N80C60S	30829.7	20780.1	-1.86%	34112.7	20608.8	8.60%	37	61
30N80C90S	30627.4	18761.0	-2.50%	34012.4	18478.6	8.28%	50	59

the key parameters. However, due to time limit, we conducted the experiments with reasonable parameter settings. For PHA1, α is set to 1.1, ρ^0 is set to $1 + lg(1 + D^0)$, where D^0 is the inconsistency level (the number of arcs that are not agreed upon by all scenarios) after the initialization phase. For PHA2, both global and local fixed cost adjusting are enabled, with penalizing ratio $\beta = 1.1$, thresholds $c^{high} = 0.8, c^{low} = 0.2$ for global adjustment, and $c^{far} = 0.7, c^{near} = 0.2$ for local adjustment. As for the modified tabu search strategy, the initial config and initial tabu list are not modified. Most parameters used in Chapter 3 are kept at the same values, except that the running time and intense gap are decreased in order to keep the total running time for both PHA1 and PHA2 less than 10 hours.

In Tables 6.4 and 6.5, the column `Final` displays the results obtained after the second phase, while the column `Phase I` represents the results of the first phase, i.e., the proposed progressive hedging algorithm with different subproblem mod-

ification strategies. The column Best reports the best integer solution found by the algorithms, while the column Time represents the wall-clock time in seconds. The column Diff indicates the relative difference between the results listed in the previous column Best and the best results returned by Cplex (the Opt column in Table 6.1). In order to see the integer convergence of the progressive hedging algorithms, the total iteration numbers are reported in the column Iters. The column D shows the inconsistency level at the end of Phase I.

Our first conclusion from Tables 6.4 and 6.5 is very encouraging: the proposed algorithms perform very well in finding high quality designs for the stochastic network design problem. For the first group of instances that are relatively easy to solve, both algorithms are able to obtain the optimal solutions. As for the second and third groups, both PHA1 and PHA2 improved the best solutions found by Cplex for 9 instances over 12. As for the 3 instances where PHA1 and PHA2 cannot outperform Cplex, good solutions are achieved very quickly, with a relative difference less than 1.5% for PHA1, and less than 1% for PHA2.

The experiments conducted on the 135 instances of set R also support the above conclusion. PHA1 is able to find the optimal solution for 72 instances, and to improve the solution quality for 40 instances. As for PHA2, optimal solutions are achieved for 65 instances, and improvements are observed for 41 instances. For both algorithms, the largest improvement is over 25%, while the worst deterioration in quality is less than 2%. We summarize in Table 6.6 the numerical results presented in Appendix.

In Table 6.6, 15 groups of instances are sorted according to their difficulty. In other words, how long does it take for Cplex to solve the corresponding deterministic network design problem. The total number of nodes, arcs and commodities, as well as the fixed cost and capacity ratios are displayed in Column Probs. The number following the letter **F** indicates a relatively high or low fixed cost relative

Table 6.6: Problem set R: Summary

		PHA1		PHA2	
Group	Probs	TimeRatio	SolDiff	TimeRatio	SolDiff
R04-1	10N 25A 10C F01 C1	5.24	0.00%	5.27	0.00%
R04-3	10N 25A 10C F10 C1	0.12	0.00%	0.12	0.00%
R06-7	10N 50A 50C F01 C8	2.36	0.00%	2.34	0.00%
R04-5	10N 25A 10C F05 C2	0.07	0.05%	0.07	0.05%
R06-1	10N 50A 50C F01 C1	0.41	0.02%	0.38	0.02%
R04-7	10N 25A 10C F01 C8	0.36	0.06%	0.36	0.06%
R04-9	10N 25A 10C F10 C8	0.17	0.18%	0.17	0.18%
R10-1	20N 100A 40C F01 C1	0.07	0.00%	0.07	0.11%
R06-9	10N 50A 50C F10 C8	0.47	0.16%	0.40	0.16%
R06-3	10N 50A 50C F10 C1	0.65	-1.34%	0.65	-1.43%
R10-7	20N 100A 40C F01 C8	0.64	-1.38%	0.69	-1.38%
R06-5	10N 50A 50C F05 C2	0.66	0.11%	0.63	0.07%
R10-3	20N 100A 40C F10 C1	0.43	-15.99%	0.47	-15.54%
R10-9	20N 100A 40C F10 C8	0.79	-2.52%	0.70	-2.65%
R10-5	20N 100A 40C F05 C2	0.84	-5.01%	0.64	-7.37%

to the routing cost. Three different levels (01, 05, 10) are used. The larger this number, the higher the fixed cost. The number following the letter **C** indicates a relatively tight or loose capacity constraints relative to the demands. Three different levels (1, 2, 8) are used. The larger this number, the tighter the capacity constraints. The columns `TimeRatio` represent the ratio between the total time used by Cplex and that used by PHA1 or PHA2 to solve the 9 instances in a group. The columns `SolDiff` report for each group the average relative difference between the best solution found by Cplex and that found by PHA1 or PHA2.

We can conclude from Table 6.6 that PHA1 and PHA2 outperform Cplex in two aspects. First, for easy problems (except some trivial ones), PHA1 and PHA2 can reach the optimal or near-optimal solutions very efficiently. Second, for harder problems, PHA1 and PHA2 can find better solutions using less time. Our experiments are conducted with a given parameter setting. With careful calibration on the parameters, further improvements can be expected.

Another observation from Tables 6.4 and 6.5 is that PHA2 performs better than PHA1 in finding good solutions. For 8 instances over 12 in the second and third group, the solutions found by PHA2 are better than those of PHA1. As for the instance set R, there is no apparent impact observed for the correlation levels, nor systematic superiority between PHA1 and PHA2.

A more in-depth study of the first phase on instance set S supports the conclusion that for stochastic network design problem, non-convergence of the progressive hedging algorithm is not only possible, but very common. Even for the small instances in the first group, only PHA2 converged for one instance, all other inconsistency levels D are positive at the end of the first phase.

By comparing the relative difference and running time of Phase I with that of the overall algorithm, we can see that over 90% of computation time is used by the first phase, except for two instances (16N80C60S and 16N80C90S). The first phase itself cannot improve the best solution, but it is efficient in decreasing the inconsistency level, thus restricting the size of the original problems.

6.4 Conclusion

Summarizing the computational results, we can conclude that the proposed algorithms are able to find high quality designs efficiently for the stochastic network design problems. The integration of the cycle-based tabu search and the progressive hedging framework is successful. Within the progressive hedging framework, both the Augmented Lagrangian and the heuristic fixed cost adjustment strategies are able to drive different scenario solutions toward a global design. Another conclusion is that non-convergence is common for the progressive hedging algorithms, due to the integrality constraints on the design decisions. A second phase is necessary to determine the final decisions on the controversial arcs.

The progressive hedging approach for stochastic network design problem is very promising. Further research can be undertaken on several avenues. One avenue is to balance the effort of pushing scenario problems to optimality and that of pulling them together as a global design. At each iteration, how good the scenario solutions should be? A comparison between using exact methods and heuristic methods to solve subproblems will be interesting. A second direction is to adapt existing metaheuristic methods for the second phase, such as tabu search, path relinking and scatter search. Finally, the parallel implementation can be carried out using MPI (message passing interface) instead of OpenMP, in order to run the program in parallel over a network of heterogeneous computers and profit more from the parallel structure of the progressive hedging algorithm.

Chapter 7

Conclusion

In this thesis, we proposed and implemented a parallel metaheuristic algorithm that integrates the cycle-based tabu search method and the progressive hedging framework, in order to obtain good solutions for the stochastic fixed-charge capacitated multicommodity network design problem with uncertain demands. Experimental results show that this combination is powerful in finding high quality design decisions.

The contribution of this thesis is twofold. First, from an operations research point of view, the proposed scenario-decomposition and relaxation method for stochastic network design problem, together with the second phase, provides an efficient way to solve the stochastic network design problems with uncertain demands. Second, from the software engineering point of view, the re-engineering work for the existing code of the cycle-based tabu search method, together with the implementation of the proposed progressive hedging-inspired algorithm, provides a good basis for future research.

Several interesting research avenues are now before us. One avenue consists in studying the impact of the subproblem solver on the efficiency and quality of the proposed algorithm. In other words, how to make a better tradeoff between the effort in finding good scenario solutions and that in consolidating the scenario solutions into one global design. More experiments are needed to establish effective parameter ranges and to study the performance of the proposed algorithm. Using

other metaheuristic methods, such as path relinking, in place of tabu search will also be an interesting try.

Another avenue of research would be to group multiple scenarios, solving the resulting smaller number of larger subproblems, and blending these solutions. Finding appropriate criteria for partitioning the scenarios could be a difficult but interesting task. Should similar or different scenarios be grouped together?

Developing the second phase algorithm to solve the restricted problem opens up another intriguing but challenging perspective. Metaheuristics such as tabu search, path relinking and scatter search are promising.

Finally, from the software point of view, an MPI implementation of the parallel algorithm is needed in order to run the algorithm in parallel on bigger clusters or on heterogenous networks.

Bibliography

[1] AHUJA, R.K., T.L. MAGNANTI et J.B. ORLIN, *Network Flows Theory, Algorithms and Applications.* Prentice Hall, New Jersey, 1993.

[2] ANDERSON, D.R., D.J. SWEENEY et T.A. WILLIAMS, *An Introduction to Management Science: Quantitative Approaches to Decision Making.* George Werthman, Mason, Ohio, USA, 2005.

[3] ATAMTÜK, A. et M. ZHANG, Two-Stage Robust Network Flow and Design Under Demand Uncertainty , *Operations Research*, vol. 55, no. 4, 2007, pp. 662–673.

[4] BALAKRISHNAN, A., T.L. MAGNANTI et P. MIRCHANDANI. Network Design . In *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli, et S. Martello, Eds. Jonh Wiley & Sons, Chichester, 1997, pp. 311–334.

[5] BALAKRISHNAN, A., T.L. MAGNANTI, A. SHULMAN et R.T. WONG, Models for Planning Capacity Expansion in Local Access Telecommunication Networks , *Annals of Operations Research*, vol. 33, no. 4, 1991, pp. 239–284.

[6] BIRGE, J.R. et F. LOUVEAUX, *Introduction to Stochastic Programming.* Springer, New York, NY, 1997.

[7] BUTENHOF, D.R., *Programming with POSIX Threads.* Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1997.

[8] CARØE, C.C. et R. SCHULTZ, Dual Decomposition in Stochastic Integer Programming , *Operations Research Letters*, vol. 24, 1998, pp. 37–45.

[9] CARØE, C.C. et J. TIND, L-shaped Decomposition of Two-stage Stochastic Programs with Integer Recourse , *Mathematical Programming*, vol. 83, 1998, pp. 451–464.

[10] CHANDRA, R., R. MENON, L. DAGUM, D. KOHR, D. MAYDAN et J. MCDONALD, *Parallel Programming in OpenMP*. Academic Press, Morgan Kaufmann, 2000.

[11] CHOUMAN, M., T.G. CRAINIC et B. GENDRON. A Cutting-Plane Algorithm Based on Cutset Inequalities for Multicommodity Capacitated Fixed Charge Network Design . Publication CRT-316, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 2003.

[12] CRAINIC, T.G., Service Network Design in Freight Transportation , *European Journal of Operational Research*, vol. 122, 2000, pp. 272–288.

[13] CRAINIC, T.G., A. FRANGIONI et B. GENDRON, Bundle-Based Relaxation Methods for Multicommodity Capacitated Network Design , *Discrete Applied Mathematics*, vol. 112, 2001, pp. 73–99.

[14] CRAINIC, T.G., M. GENDREAU et J.M. FARVOLDEN, A Simplex-Based Tabu Search Method for Capacitated Network Design , *INFORMS Journal on Computing*, vol. 12, no. 3, 2000, pp. 223–236.

[15] CRAINIC, T.G., B. GENDRON et G. HERNU, A Slope Scaling/Lagrangian Perturbation Heuristic with Long-Term Memory for Multicommodity Capacitated Fixed-Charge Network Design , *Journal of Heuristics*, vol. 10, no. 3, 2004, pp. 525–545.

[16] FRANGIONI, A. et B. GENDRON. 0-1 Reformulations of the Multicommodity Capacitated Network Design Problem . Publication CIRRELT-2007-29,

Centre Interuniversitaire de Recherche sur les Réseaux d' Entreprise, la Logistique et le Transport (CIRRELT), Montréal, QC, Canada, 2007.

[17] GENDREAU, M. et J.Y. POTVIN. A Guide to Tabu Search . Publication CRT-2003-23, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 2003.

[18] GENDRON, B., T.G. CRAINIC et A. FRANGIONI. Multicommodity Capacitated Network Design . In *Telecommunications Network Planning*, P. Soriano et B. Sansò, Eds. Kluwer Academics Publisher, 1999, pp. 1–19.

[19] GHAMLOUCHE, I. *Métaheuristiques de Recherche avec Tabous pour le Problème de Synthèse de Réseau Multiproduits avec Capacités.* PhD thesis, Université de Montréal, Montréal, QC, Canada, 2004.

[20] GHAMLOUCHE, I., T.G. CRAINIC et M. GENDREAU, Path Relinking, Cycle-Based Neighborhoods and Capacitated Multicommodity Network Design , *Annals of Operations Research*, vol. 131, 2004, pp. 109–133.

[21] GHAMLOUCHE, I., T.G. CRAINIC et M. GNDREAU, Cycle-Based Neighbourhoods for Fixed-Charge Capacitated Multicommodity Network Design , *Operations Research*, vol. 51, no. 4, 2003, pp. 655–667.

[22] GROPP, W., E. LUSK et A. SKJELLUM, *Using MPI, 2nd Edition: Portable Parallel Programming with the Message Passing Interface.* MIT Press in Scientific and Engineering Computation Series, Cambridge, MA, USA, 1999.

[23] HOFF, A., A-G. LIUM, A. LØKKETANGEN et T.G. CRAINIC. A Metaheuristic for Stochastic Service Network Design . Publication CIRRELT-2007-623, Centre Interuniversitaire de Recherche sur les Réseaux d' Entreprise, la Logistique et le Transport (CIRRELT), Montréal, QC, Canada, 2007.

[24] HOLMBERG, K. et D. YUAN, A Lagrangian Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem , *Operations Research*, vol. 48, no. 3, 2000, pp. 461–481.

[25] HØYLAND, K., M. KAUT et S.W. WALLACE, A Heuristic for Moment-Matching Scenario Generation , *Computational Optimization and Applications*, vol. 24, no. 2-3, 2003, pp. 169–185.

[26] ILOG. *ILOG CPLEX 10.0.* ILOG, Mountain View, CA. U.S.A, 2006.

[27] KALL, P. et S.W. WALLACE, *Stochastic Programming.* Jonh Wiley & Sons, Chichester, 1994.

[28] KAUT, M. et S.W. WALLACE. Evaluation of Scenario-generation Methods for Stochastic Programming . Stochastic Programming E-Print Series (SPEPS) 2003-14, 2003.

[29] KLIEWER, G. et L. TIMAJEV. Relax-and-Cut for Capacitated Network Design . In *Proceedings of Algorithms-ESA 2005: 13th Annual European Symposium on Algorithms* (2005), Lecture Notes in Computer Science 3369, pp. 47–58.

[30] KUMAR, V., A. GRAMA, A. GUPTA et G. KARPIS, *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms.* Benjamin-Cummings Publishing Company, Harlow, England, 1993.

[31] LAPORTE, G. et F.V. LOUVEAUX, The Integer L-Shaped Method for Stochastic Integer Programs with Complete Recourse , *Operations Research Letters*, vol. 13, no. 3, 1993, pp. 133–142.

[32] LIUM, A.-G., CRAINIC, T.G. et WALLACE, S., A Study of Demand Stochasticity in Service Network Design , *Transportation Science*, 2007. forthcoming.

[33] LØKKETANGEN, A. et D. WOODRUFF, Progressive Hedging and Tabu Search Applid to Mixd Integer (0,1) Multistage Stochastic Programming , *Journal of Heuristics*, vol. 2, 1996, pp. 111–128.

[34] MAGNANTI, T.L. et R.T. WONG, Network Design and Transportation Planning: Models and Algorithms , *Transportation Science*, vol. 18, no. 1, 1984, pp. 1–55.

[35] MINOUX, M., Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications , *Networks*, vol. 19, no. 3, 1989, pp. 313–360.

[36] RIIS, M. et K.A. ANDERSEN, Capacitated Network Design with Uncertain Demand , *INFORMS Journal on Computing*, vol. 14, no. 3, 2002, pp. 247–260.

[37] ROCKAFELLAR, R.T. et R.J.-B. WETS, Scenarios and Policy Aggregation in Optimization under Uncertainty , *Mathematics of Operations Research*, vol. 16, 1991, pp. 119–147.

[38] SELLMANN, M., G. KLIEWER et A. KOBERSTEIN. Lagrangian Cardinality Cuts and Variable Fixing for Capacitated Network Design . In *Proceedings of Algorithms-ESA 2002: 10th Annual European Symposium on Algorithms* (2002), Lecture Notes in Computer Science 2461, pp. 845–858.

[39] WETS, R.J.-B. The Aggregation Principle in Scenario Analysis and Stochastic Optimization . In *Algorithms and Model Formulations in Mathematical Programming* (Berlin, 1989), S. Wallace, Ed., vol. 51, Springer-Verlag, pp. 91–113.

Appendix

Results for the instances of set R are given in Tables 7.1 to 7.15.

We tested on three groups of deterministic instances (R04, R06, R10) selected from the data set **R** of [21], with 5 different combined levels of fixed cost and capacity ratio (1, 3, 5, 7, 9). The total number of nodes, arcs and commodities, as well as the relative fixed cost ratio and capacity ratio are displayed in the title of the tables. Finally, the optimal solution value for the deterministic network design problem is displayed. Recall that the stochastic demands are generated using the triangle distribution whose mode value is set to the value of its corresponding deterministic demands w^k, min value is zero, and max value is $1.25w^k$. The optimal solution value of the deterministic problem represents the total cost of the decision where the unknown demands are estimated as 80% of their maximum values. By comparing this value with the solution values of the stochastic model, we can see the saving of using the stochastic model, or, in another words, the value of the stochastic solution.

The following information is displayed in the tables:

- `Prob`: The correlation coefficient used to generate the scenarios and the number of scenarios.
- `Opt`: The best integer solution found by Cplex.
- `LB`: The lower bound found by Cplex.
- `Gap`: The optimality gap between the best integer solution and the lower

bound found by Cplex, in the case that an optimal solution is not found within the time limit.

- `PHA1`: The best integer solution found by the progressive hedging algorithm with Augmented Lagrangian.
- `PHA2`: The best integer solution found by the progressive hedging algorithm with heuristic fixed cost adjusting.
- `Diff`: The relative difference between the best solution found by PHA and that obtaind by Cplex.

On the second line, the figures under the solutions are the wall-clock time in seconds.

Table 7.1: R04-1: 10N, 25A, 10C, F01, C1, 31730

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	24443.5	24443.5	0.00%	24443.5	0.00%
	0.76	3.31		3.39	
0r 32S	25219.9	25219.9	0.00%	25219.9	0.00%
	1.40	6.87		6.70	
0r 64S	25291.3	25291.3	0.00%	25291.3	0.00%
	5.03	20.83		19.41	
0.2r 16S	24557.7	24557.7	0.00%	24557.7	0.00%
	0.68	3.30		3.47	
0.2r 32S	25437.4	25437.4	0.00%	25437.4	0.00%
	1.73	9.47		9.95	
0.2r 64S	25091.2	25091.2	0.00%	25091.2	0.00%
	3.72	19.71		18.89	
0.8r 16S	25765.8	25765.8	0.00%	25765.8	0.00%
	0.55	4.49		4.13	
0.8r 32S	25403.5	25403.5	0.00%	25403.5	0.00%
	1.21	9.25		8.32	
0.8r 64S	25122.2	25122.2	0.00%	25122.2	0.00%
	3.47	19.93		23.59	

Table 7.2: R04-3: 10N, 25A, 10C, F10, C1, 63767

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	53399.3	53399.3	0.00%	53399.3	0.00%
	5.20	5.78		5.83	
0r 32S	54527.5	54527.5	0.00%	54527.5	0.00%
	30.23	11.66		11.82	
0r 64S	54503.1	54503.1	0.00%	54503.1	0.00%
	183.38	26.51		25.71	
0.2r 16S	56121.9	56121.9	0.00%	56121.9	0.00%
	17.63	5.72		5.72	
0.2r 32S	57463.3	57463.3	0.00%	57463.3	0.00%
	173.54	13.80		11.95	
0.2r 64S	57062.9	57062.9	0.00%	57062.9	0.00%
	542.90	27.03		27.19	
0.8r 16S	55470.2	55470.2	0.00%	55470.2	0.00%
	4.07	5.74		6.40	
0.8r 32S	54765.0	54765.0	0.00%	54765.0	0.00%
	21.27	12.39		14.05	
0.8r 64S	54340.3	54340.3	0.00%	54340.3	0.00%
	180.23	28.46		28.14	

Table 7.3: R04-5: 10N, 25A, 10C, F05, C2, 53790

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	44681.7	44720.2	0.09%	44720.2	0.09%
	19.12	5.48		5.40	
0r 32S	46033.2	46033.2	0.00%	46033.2	0.00%
	196.80	12.87		13.05	
0r 64S	45938.3	46097.2	0.35%	46097.2	0.35%
	686.51	42.00		42.34	
0.2r 16S	45307.0	45307.0	0.00%	45307.0	0.00%
	21.74	5.98		6.02	
0.2r 32S	46377.5	46377.5	0.00%	46377.5	0.00%
	167.54	14.05		13.93	
0.2r 64S	45840.0	45840.0	0.00%	45840.0	0.00%
	780.34	45.28		44.90	
0.8r 16S	47302.3	47302.3	0.00%	47302.3	0.00%
	44.09	6.00		5.78	
0.8r 32S	46842.5	46842.5	0.00%	46842.5	0.00%
	182.25	13.50		12.87	
0.8r 64S	46392.4	46392.4	0.00%	46392.4	0.00%
	1080.20	66.26		65.60	

Table 7.4: R04-7: 10N, 25A, 10C, F01, C8, 68291.7

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	47003.3	47003.3	0.00%	47003.3	0.00%
	15.67	9.82		9.95	
0r 32S	48707.3	48711.2	0.01%	48711.2	0.01%
	86.57	36.87		36.94	
0r 64S	48975.9	48975.9	0.00%	48975.9	0.00%
	546.18	109.74		110.56	
0.2r 16S	48382.3	48645.6	0.54%	48645.6	0.54%
	42.28	13.88		15.78	
0.2r 32S	50128.5	50128.5	0.00%	50128.5	0.00%
	49.24	27.88		28.60	
0.2r 64S	49649.7	49649.7	0.00%	49649.7	0.00%
	294.58	127.91		165.44	
0.8r 16S	51716.8	51716.8	0.00%	51716.8	0.00%
	23.53	13.13		13.28	
0.8r 32S	51049.1	51049.1	0.00%	51049.1	0.00%
	88.74	36.29		36.34	
0.8r 64S	50270.5	50270.5	0.00%	50270.5	0.00%
	481.98	218.06		173.22	

Table 7.5: R04-9: 10N, 25A, 10C, F10, C8, 163208

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	140513	140513	0.00%	140513	0.00%
	409.15	52.30		38.70	
0r 32S	144062	144062	0.00%	144062	0.00%
	1083.92	116.88		118.14	
0r 64S	147914	147914	0.00%	147914	0.00%
	7465.32	545.62		544.44	
0.2r 16S	141344	141344	0.00%	141344	0.00%
	640.23	62.59		55.78	
0.2r 32S	147961	147961	0.00%	147961	0.00%
	1915.91	221.81		221.61	
0.2r 64S	148351	148351	0.00%	148351	0.00%
	10260.20	1968.49		1983.03	
0.8r 16S	151937	153176	0.82%	153176	0.82%
	119.08	42.55		42.68	
0.8r 32S	151795	153034	0.82%	153034	0.82%
	581.80	383.22		382.58	
0.8r 64S	152150	152155	0.00%	152155	0.00%
	4825.28	1197.54		1197.11	

Table 7.6: R06-1: 10N, 50A, 50C, F01, C1, 245936

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	188606	188606	0.00%	188606	0.00%
	269.45	71.24		70.90	
0r 32S	188093	188093	0.00%	188093	0.00%
	1099.49	326.69		321.48	
0r 64S	189598	189598	0.00%	189598	0.00%
	4283.06	2081.46		2077.88	
0.2r 16S	192667	193050	0.20%	193050	0.20%
	225.56	78.47		60.56	
0.2r 32S	191256	191256	0.00%	191256	0.00%
	705.08	330.24		256.47	
0.2r 64S	190419	190419	0.00%	190419	0.00%
	3088.04	1251.32		1098.92	
0.8r 16S	191947	191947	0.00%	191947	0.00%
	210.95	70.43		68.87	
0.8r 32S	194169	194169	0.00%	194169	0.00%
	895.40	288.38		241.59	
0.8r 64S	191903	191903	0.00%	191903	0.00%
	3205.25	1240.86		1081.42	

Table 7.7: R06-3: 10N, 50A, 50C, F10, C1, 559477

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	464744	464744	0.00%	464744	0.00%	464744	0.00%
	27687.9			3946.04		3927.46	
0r 32S	471673	441978	6.30%	464115	-1.60%	464115	-1.60%
	t			30051.1		30052.5	
0r 64S	536416	424970	20.78%	482959	-9.97%	482959	-9.97%
	t			30100.4		30101.7	
0.2r 16S	472709	472709	0.00%	476182	0.73%	476182	0.73%
	17809.4			4749.63		4719.27	
0.2r 32S	477375	449023	5.94%	473709	-0.77%	473709	-0.77%
	t			30052.7		30052.0	
0.2r 64S	485055	424766	12.43%	485843	0.16%	485843	0.16%
	t			30101.6		30100.0	
0.8r 16S	474333	474333	0.00%	474333	0.00%	474333	0.00%
	9749.17			1660.8		1659.15	
0.8r 32S	480563	461377	3.99%	477338	-0.67%	477338	-0.67%
	t			15978.1		16013.5	
0.8r 64S	477260	438256	8.17%	477417	0.03%	473763	-0.73%
	t			30076.0		30083.6	

Table 7.8: R06-5: 10N, 50A, 50C, F05, C2, 498266

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	397396 t	392374	1.26%	396980 13108.0	-0.10%	396980 3843.9	-0.10%
0r 32S	399850 t	386240	3.40%	401241 30054.7	0.35%	401241 30055.4	0.35%
0r 64S	405562 t	375486	7.42%	408260 30107.4	0.67%	408260 30106.5	0.67%
0.2r 16S	408612 t	405503	0.76%	408612 6144.8	0.00%	408612 5643.1	0.00%
0.2r 32S	413984 t	396902	4.13%	411309 30054.3	-0.65%	411309 30060.3	-0.65%
0.2r 64S	416166 t	382776	8.02%	423123 30104.9	1.67%	423123 30105.8	1.67%
0.8r 16S	419464 t	410285	2.19%	419464 15715.0	0.00%	419464 13187.6	0.00%
0.8r 32S	420822 t	407749	3.11%	423535 30053.2	0.64%	421995 30061.5	0.28%
0.8r 64S	431105 t	393020	8.83%	424362 30105.2	-1.56%	424362 30109.7	-1.56%

Table 7.9: R06-7: 10N, 50A, 50C, F01, C8, 682921

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	412085 9.81	412085 28.23	0.00%	412085 29.88	0.00%
0r 32S	409609 19.95	409609 53.98	0.00%	409609 56.51	0.00%
0r 64S	409033 59.37	409033 114.93	0.00%	409033 109.30	0.00%
0.2r 16S	431646 8.33	431646 26.16	0.00%	431646 25.67	0.00%
0.2r 32S	424527 20.39	424527 57.64	0.00%	424527 57.12	0.00%
0.2r 64S	417223 54.92	417223 111.76	0.00%	417223 110.04	0.00%
0.8r 16S	441098 7.96	441098 25.71	0.00%	441098 25.74	0.00%
0.8r 32S	436548 17.44	436548 53.32	0.00%	436548 53.43	0.00%
0.8r 64S	428714 50.71	428714 114.55	0.00%	428714 113.85	0.00%

Table 7.10: R06-9: 10N, 50A, 50C, F10, C8, 423316

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	353124	353124	0.00%	359863	1.91%	359863	1.91%
	3427.45			1176.18		1173.16	
0r 32S	351558	351558	0.00%	358349	1.93%	358349	1.93%
	7900.68			2886.36		3246.53	
0r 64S	353682	339976	3.88%	350335	-0.95%	350335	-0.95%
	t			16837.1		12681.60	
0.2r 16S	352792	352792	0.00%	352792	0.00%	352792	0.00%
	1768.30			388.60		389.95	
0.2r 32S	351464	351464	0.00%	351464	0.00%	351464	0.00%
	7746.51			1544.03		1519.97	
0.2r 64S	355284	345867	2.65%	355284	0.00%	355284	0.00%
	t			15519.5		15511.1	
0.8r 16S	366153	366153	0.00%	366153	0.00%	366153	0.00%
	2148.84			746.32		741.50	
0.8r 32S	363209	363209	0.00%	363209	0.00%	363209	0.00%
	7202.98			2489.33		2497.49	
0.8r 64S	366715	352438	3.89%	361316	-1.47%	361316	-1.47%
	t			22708.6		17708.3	

Table 7.11: R10-1: 20N, 100A, 40C, F01, C1, 200087

Prob	Opt	PHA1	Diff	PHA2	Diff
0r 16S	157212	157212	0.00%	157391	0.11%
	128.07	40.22		41.29	
0r 32S	157027	157027	0.00%	157268	0.15%
	634.57	78.83		87.24	
0r 64S	158162	158162	0.00%	158428	0.17%
	4421.14	187.62		203.15	
0.2r 16S	161953	161953	0.00%	161976	0.01%
	207.00	37.95		38.46	
0.2r 32S	162697	162697	0.00%	162959	0.16%
	778.63	105.02		113.06	
0.2r 64S	161811	161811	0.00%	161936	0.08%
	5095.02	371.95		226.15	
0.8r 16S	149496	149496	0.00%	149991	0.33%
	169.95	42.05		54.81	
0.8r 32S	157132	157132	0.00%	157132	0.00%
	818.55	86.05		88.67	
0.8r 64S	157982	157982	0.00%	157982	0.00%
	4336.90	243.95		251.86	

Table 7.12: R10-3: 20N, 100A, 40C, F10, C1, 488015

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	422737 t	397312	6.01%	414893 785.35	-1.86%	414893 535.18	-1.86%
0r 32S	545291 t	390615	28.37%	411154 4800.88	-24.60%	418407 13919.2	-23.27%
0r 64S	545526 t	358575	34.27%	417406 30169.7	-23.49%	423467 30228.9	-22.37%
0.2r 16S	445535 t	407364	8.57%	433244 3387.66	-2.76%	433244 3379.81	-2.76%
0.2r 32S	550822 t	401589	27.09%	435142 24188.1	-21.00%	436274 30094.8	-20.80%
0.2r 64S	549234 t	366599	33.25%	443535 30169.4	-19.24%	443535 30169.3	-19.24%
0.8r 16S	413712 t	385596	6.80%	400062 1563.38	-3.30%	403744 621.74	-2.41%
0.8r 32S	560316 t	389341	30.51%	415919 14700.2	-25.77%	415919 12358.6	-25.77%
0.8r 64S	545477 t	352787	35.33%	426133 30166.5	-21.88%	428732 30169.5	-21.40%

Table 7.13: R10-5: 20N, 100A, 40C, F05, C2, 411664

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	351503 t	321461	8.55%	345486 30062.5	-1.71%	345321 29804.3	-1.76%
0r 32S	364302 t	322212	11.55%	341437 30120.7	-6.28%	341437 30120.9	-6.28%
0r 64S	468867 t	305347	34.88%	450573 30247.0	-3.90%	450573 30256.5	-3.90%
0.2r 16S	357027 t	330396	7.46%	353769 30067.8	-0.91%	348255 5240.4	-2.46%
0.2r 32S	465347 t	331946	28.67%	352548 30125.6	-24.24%	349931 13574.3	-24.80%
0.2r 64S	463309 t	307762	33.57%	450668 30259.1	-2.73%	357695 30317.5	-22.80%
0.8r 16S	335980 t	310975	7.44%	330408 30060.5	-1.66%	333614 8773.5	-0.70%
0.8r 32S	359393 t	320636	10.78%	357134 30125.7	-0.63%	357134 30126.8	-0.63%
0.8r 64S	468849 t	298968	36.23%	454733 30251.2	-3.01%	454733 30250.9	-3.01%

Table 7.14: R10-7: 20N, 100A, 40C, F01, C8, 486895

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	326484	326484	0.00%	326484	0.00%	326484	0.00%
	32803.10			13153.7		14385.9	
0r 32S	327372	321219	1.88%	326778	-0.18%	327334	-0.01%
	t			30152.0		30162.7	
0r 64S	344050	318810	7.34%	330915	-3.82%	330915	-3.82%
	t			30303.7		30304.7	
0.2r 16S	345146	343040	0.61%	345130	0.00%	345130	0.00%
	t			9292.4		21534.6	
0.2r 32S	350681	347923	0.79%	350585	-0.03%	350510	-0.05%
	t			30145.7		30144.6	
0.2r 64S	359702	333581	7.26%	344542	-4.21%	344542	-4.21%
	t			30290.2		30292.5	
0.8r 16S	307068	307068	0.00%	307712	0.21%	307712	0.21%
	16010.60			4788.3		4750.4	
0.8r 32S	329339	325059	1.30%	329031	-0.09%	329031	-0.09%
	t			14558.2		14489.2	
0.8r 64S	348317	324212	6.92%	333436	-4.27%	333011	-4.39%
	t			30313.5		30294.8	

Table 7.15: R10-9: 20N, 100A, 40C, F10, C8, 1421740

Prob	Opt	LB	Gap	PHA1	Diff	PHA2	Diff
0r 16S	1143160	1059240	7.34%	1141320	-0.16%	1140800	-0.21%
	t			30096.4		12370.2	
0r 32S	1224320	1032810	15.64%	1145330	-6.45%	1147270	-6.29%
	t			30197.3		30216.9	
0r 64S	1223720	987389	19.31%	1204500	-1.57%	1199340	-1.99%
	t			30347.5		30416.0	
0.2r 16S	1177240	1095320	6.96%	1176120	-0.10%	1176120	-0.10%
	t			15732.4		3467.6	
0.2r 32S	1251260	1065030	14.88%	1180580	-5.65%	1180740	-5.64%
	t			30168.8		30197.7	
0.2r 64S	1259000	1015950	19.31%	1242250	-1.33%	1242250	-1.33%
	t			30364.0		30441.6	
0.8r 16S	1114670	1041250	6.59%	1115020	0.03%	1114540	-0.01%
	t			30103.4		29695.6	
0.8r 32S	1246500	1053660	15.47%	1153380	-7.47%	1152530	-7.54%
	t			30185.0		30211.4	
0.8r 64S	1210520	1006030	16.89%	1210520	0.00%	1201710	-0.73%
	t			30328.1		30416.4	