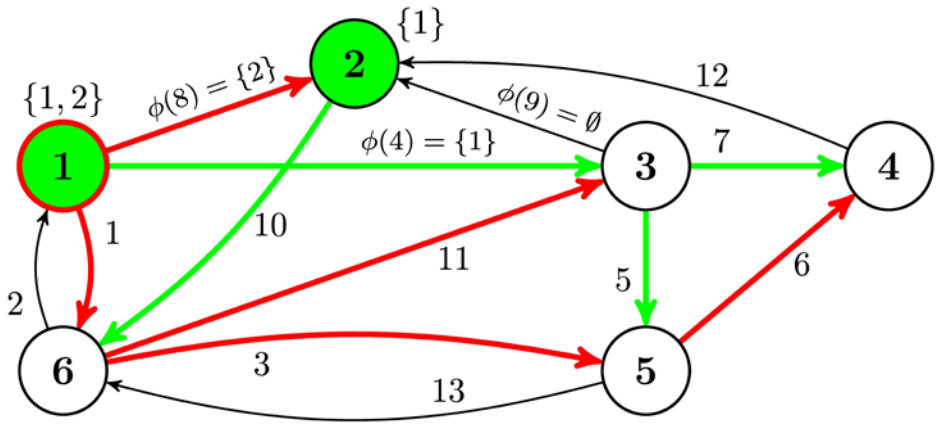
Data distribution optimization in a system of collaborative systems

Ronan Bocquillon

Par **Ronan BOCQUILLON**

Data distribution optimization in a system of collaborative systems

Thèse présentée pour l'obtention du grade de Docteur de l'UTC



Soutenue le 16 novembre 2015
Spécialité : Technologies de l'Information et des Systèmes

D2232

Data Distribution Optimization in a System of Collaborative Systems

Ronan Bocquillon

Sorbonne Universités, Université de Technologie de Compiègne,
CNRS, Heudiasyc UMR 7253

Thèse soutenue le 16 novembre 2015

Jury

Jacques Carlier	Professeur des Universités	Président
Chengbin Chu	Professeur des Universités	Rapporteur
Marie-Christine Costa	Professeur des Universités	Rapporteur
Vincent Jost	Chargé de Recherche CNRS	Examinateur
Antoine Jouglet	Maître de Conférences HDR	Directeur
Jérôme Rogerie	IBM-Ilog	Examinateur

Remerciements

Je souhaite remercier Chengbin Chu et Marie-Christine Costa pour avoir accepté de rapporter ce manuscrit, ainsi que Jacques Carlier et Vincent Jost pour avoir participé au jury. Je suis heureux d'avoir pu leur présenter mes travaux, et j'espère très sincèrement qu'ils ont pris plaisir à évaluer cette thèse. Je remercie tout particulièrement Jacques, notre Grand Guru à tous, qui en plus de nous avoir conseillés dans l'étude de complexité menée en début de thèse, m'a fait l'immense honneur de présider le jury.

Je remercie ensuite Antoine Jouglet pour sa patience et sa disponibilité durant ces trois années passées en sa compagnie. Chargé de TD en recherche opérationnelle, en programmation orientée objet, en programmation logique, en programmation par contraintes, puis responsable d'un projet de recherche sur le job shop, et enfin directeur de thèse passionnant, il aura été à tous les étages de ma formation utcéenne. Dans chacune de ses missions, sa pédagogie excelle, son dévouement est complet, et ses qualités humaines... simplement indiscutables. Mon aventure universitaire se serait sans aucun doute arrêtée trois ans plus tôt sans Antoine. Je tiens d'ailleurs à m'excuser auprès de son épouse Delphine et de leurs enfants pour les nombreuses heures dominicales qu'il a sacrifiées pour moi.

Je remercie également tous les membres du labo avec qui j'ai pu discuter musique, rando, jeux, recherche et autres, en pause-café, au Minibar, en conf et ailleurs... Je remercie particulièrement Ada, Benjamin, Benoît, Bérengère, Brigitte, Dominique, Dritan, Gérald, Gildas, Laurie, Louise, Lyes, Michael, Mylène, Nathalie, Paul, Rym, Sabine, Sébastien (docteur ès *tikz*), Séverine, Sohaib, Sylvain, Taha, Yves et Véronique. Merci à Ali de toujours œuvrer à la cohésion du labo !

Je souhaiterais aussi remercier tous ceux avec qui j'ai eu la chance de partager le pain, le sel, et surtout... les dés ! Je remercie évidemment David, Esther et Lucien pour les "soirées du jeudi", les *Witness*, les frites au gras de bœuf, les hamburgers, les courses de "patézés", les *Love Letters*, le *Compact Curling*, et tant d'autres... Encore merci à Antoine de m'avoir fait découvrir *Hypérion*, *Gazpacho*, et l'incontournable *Race For The Galaxy*. Je m'excuse platement pour mon manque de sensibilité face aux bonnes bières, ou face à un bon rouge. Dans cette tâche, je reconnais mon échec (long et difficile est le chemin de la Force) ! Je remercie en outre Olivier pour *Roll For The Galaxy*, et pardonne Charline pour ses très (trop !) nombreuses tergiversations dans *Dungeon Petz*. Je remercie les habitués de *PlayUTC*, en particulier Thibaut, les GI'12 (Nicolas et Brice), et Vincent ("il fallait bien une cible !") pour les interminables parties de *Cyclades*.

Je remercie chaleureusement mes amis les plus chers : Aurélien et Marie-Ève, Guillaume, Laura, et notre toute nouvelle star Agathe, Mathilde, Victor et Élise. Je pense aussi à ceux que j'aurais aimé voir plus souvent : Clément, Florian et Louise, François et Lucie, Guiz, Martin, Maxime, ... Mes retours dans le Nord ont toujours été – et seront encore – mon meilleur remède dans les moments difficiles. Merci à vous pour les nombreuses soirées *Tennessee*, les soirées bouffes, les soirées jeux, sans oublier les moments de débauche sur *Command and Conquer*, *Call of Duty*, et *cie*. Je remercie également tous les utcéens : Andry, Christophe, Damien, Enguerrand, Ivan, Marianne, Gaëtan, et la troupe des geeks : Corentin, François, les deux Guigui, Karim, Maxime, Thibaud, Vivien, et toute la clique.

Je ne remercierai jamais assez ma famille. Mes parents pour avoir cru en moi jusqu'au bout, pour leur patience et leur soutien, pour les innombrables corrections orthaugrafiks, et pour tout l'amour qu'ils m'ont témoigné pendant 26 ans. Mes trois frères et sœur, mes éternels modèles, et inépuisables sources d'admiration. Merci également Christian, Eliane, Gaëtan et Audrey pour les week-ends sur ~~Paris~~ Cormeilles !

Pour terminer, j'embrasse Tifenn pour tout ce que les mots ne sauraient exprimer...

Abstract

Keywords – Operational research · combinatorial optimization · dominance rules · constraint programming · delay-tolerant networks · systems of systems

Systems of systems are supersystems comprising elements which are themselves complex, independent operational systems, all interacting to achieve a common goal [30]. When the subsystems are mobile and deployed in extreme environments, these may suffer from a lack of continuous end-to-end connectivity. To address the technical issues in such networks, the common approach is termed *delay-tolerant networking* [20]. Routing relies on a *store-forward* mechanism – *i.e.* data are sent from one system to another, depending on the communication opportunities that arise when two systems are close to each other, and stored throughout the network in hope that all messages will reach their destination.

In this work, we assume that the trajectory of each system (of each node) is deterministic and perfectly known. Thus, we focus on applications where it is possible to make realistic predictions about node mobility. This includes satellite networks (where the trajectory of nodes depends on straightforward physics) and public transportation systems. The problem is making the best use of knowledge about possibilities for communication – termed *contacts* in the literature – when data need to be routed from a set of nodes to another within a given time horizon. The fundamental question is "which elements of the information should be transferred from which node to which node when contacts occur". A solution to this problem is termed a *transfer plan*.

The literature on that topic is limited. Intermittently-connected networks have been widely addressed with opportunistic [5] and stochastic [3, 14, 34] approaches, but few papers [2, 27, 29, 35] have considered the deterministic case (although the applications mentioned above may require such studies). In particular, in our opinion, the literature already proposes very interesting models, but still lacks a clear and unified framework for such a situation.

That is why we started the thesis by formalizing the problem. We notably studied a simplified version – the *dissemination problem* – where we consider a single *datum*, split into several *datum units*, to be transmitted from a set of *source* nodes to a set of *recipient* nodes. To this end, we consider a sequence of *contacts*, *i.e.* an ordered set of pairs of nodes. During each contact, at most one datum unit can be transferred from one node called *sender*, to another called *receiver*. We proved that this (combinatorial) problem is strongly NP-Hard, when there are at least two recipients, or at least two datum units.

Subsequently we worked towards solving this problem. More specifically, we proposed several dominance rules to reduce its search space. These lead to deduction algorithms aiming at identifying useless contacts, and necessary transfers. Typically, a contact that is never leveraged in a dominant solution can be removed from the instance before the main solving process starts, *i.e.* before the enumeration procedure starts. These deductions algorithms, used as preprocessing procedures, achieved very promising results. Unfortunately our numerical experiments showed that they were inefficient when combined with a branch-and-cut algorithm (when they were dynamically used while an integer linear program modelling the problem was being solved).

Afterwards, to more appropriately use the dominance rules, we proposed a constraint-programming-based enumeration algorithm (a branch-and-bound procedure). This significantly outperformed the integer-linear-programming-based approach mentioned above. Constraint programming was shown to be particularly adapted to incorporation of *ad hoc* computations/methods, *e.g.* lower bounds, symmetry breaking techniques, nogood recording. It achieved the best experimental results on a benchmark of self-generated instances.

Finally, we addressed a robust version of the problem. This ongoing work aims to find robust transfer plans. More precisely, we seek solutions enabling the datum units to be correctly delivered, even if some transfers fail. This is based on an adaptation of the constraint programming algorithm developed beforehand for the case where all failures are disregarded. We actually hope that our approach will help prediction errors to be more effectively managed in practice.

Note that this work was presented at three international congress (namely EURO|INFORMS 2013, MISTA 2013 [9] and BWCCA 2013 [8]). It won the first MS2T Award during the first international workshop organised by Labex MS2T [37] in october 2013. A study on the complexity of the dissemination problem has been published in the international, peered-reviewed, *European Journal of Operational Research* [12]. Two other papers have been submitted to similar journals. These papers [10, 11] deal with more technical aspects – *e.g.* dominances rules, constraint programming, *etc.*

Table of contents

1 The dissemination problem **1**

1.1 Challenged internets 2

1.2 Delay-tolerant networks 3

1.3 Systems of systems 6

1.4 The dissemination problem 8

2 Complexity results **13**

2.1 The data transfer problem 15

2.1.1 The case $u \geq 2$ 16

2.1.2 The case $|\mathcal{R}| \geq 2$ 20

2.2 Polynomial-time cases 27

2.2.1 The one-datum-unit problem ($u = 1$) 27

2.2.2 The delivery problem ($|\mathcal{R}| = 1$) 29

2.2.3 Upper bounded parameters 34

2.3 Additional results 36

2.3.1 Arc-disjoint branchings in an evolving graph 36

2.3.2 Arc-disjoint Steiner trees in a digraph 42

2.4 Conclusion 43

3 Dominance rules, preprocessings, and integer linear programming 45

3.1 Dominance rules 46

3.2 Transfer graph 51

3.2.1 About the transfer graph 52

3.2.2 Transfer graph and subsets of transfer plans 54

3.2.3 Additional graph properties and complex subsets of transfer plans 57

3.2.4 Using the transfer graph 59

3.3 Deductive elements 59

3.3.1 Finding non-minimal transfer plans 60

3.3.2 Elementary reasonings 62

3.3.3 Evaluating *min-card* and *max-card* 74

3.4 Solving the dissemination problem 82

3.4.1 Integer linear programming 82

3.4.2 Additional constraints 84

3.5 Computational results 84

3.5.1 About the benchmarks 85

3.5.2 About the models 87

3.5.3 About the preprocessing procedures 88

3.6 Conclusion 95

4 Constraint programming 97

4.1 Modelling the dissemination problem 98

4.1.1 Constraint programming model 98

4.1.2 Branching algorithm 100

4.2 Additional features 102

4.2.1 Lower bounds 102

4.2.2 Symmetry-breaking techniques 104

4.3 Computational results 112
4.3.1 About the model 112
4.3.2 The additional features 114
4.4 Conclusion 117

5 Robust optimization **119**
5.1 The robust dissemination problem 121
5.1.1 Formal description 121
5.1.2 Robustness and evolving graphs 122
5.2 Robust optimization 123
5.2.1 Necessary and sufficient condition for a transfer plan to be Γ-robust 124
5.2.2 Enumeration procedure 127
5.3 A constraint programming approach 130
5.3.1 Model 130
5.3.2 Additional features 133
5.4 Preliminary results 138
5.4.1 About the benchmark 138
5.4.2 Numerical results 140
5.5 Conclusion 142

6 Conclusion and perspectives **145**

Bibliography **151**

1 The dissemination problem

The present chapter describes the background and the issues that led to this thesis. We will first remind the limits of TCP/IP protocols for networks subject to *frequent partitioning* (termed intermittently-connected networks), and subsequently what leads to the emergence of *delay-tolerant networks* (*cf.* Section 1.1). Then we will focus on the challenges related to routing data in such a network. The common approach is to use a *store-forward* mechanism – *i.e.* data are sent from one node to another, depending on the communication opportunities that arise, and stored throughout the network in hope that messages will reach their destination (*cf.* Section 1.2). These works being part of a study on *systems of systems*, we will also draw the parallel between systems of systems and delay-tolerant networks (*cf.* Section 1.3). Finally, with all these elements in hand, we will formalize the problem that will be tackled in the following chapters. In short, the problem is making use of knowledge about possibilities for communication when data need to be routed from one subset of nodes to another within a given time horizon (*cf.* Section 1.4). Thus this study focuses on applications where node mobility can be reliably predicted.

Contents

1.1 Challenged internets **2**

1.2 Delay-tolerant networks **3**

1.3 Systems of systems **6**

1.4 The dissemination problem **8**

1.1 Challenged internets

In 2015 the estimated number of Internet users is about three billions, that is more than 40% on a world population of seven billions. This uncontested success relies on the highly popular TCP/IP model [41] (that is the Internet protocol suite), whose most important elements are the Transmission Control Protocol and the Internet Protocol. These protocols make key assumptions regarding the performances of underlying links [20] – *e.g.* an end-to-end path exists between a data source and its peer(s), the maximum round-trip time between any pair of nodes is not excessive, and the end-to-end packet drop probability is small. These assumptions may not be realistic in networks that are characterized by many *disconnections* (due to node mobility), a *limited longevity* (especially where end nodes are placed in a hostile environment), possible *lack of large memory* (embedded systems), *low duty cycle operation* (due to energy saving policies for instance), or using *non-reusable protocols* (these do not usually provide a sufficient abstraction for supporting layered protocol families such as Internet). For instance, disconnections result in the absence of a long-standing reliable end-to-end path, and thus lead to higher latency, poor data rate and long queuing time.

Thus, TCP/IP protocols may operate poorly on exotic networks such as terrestrial mobile networks, wireless sensor networks, vehicle *ad hoc* networks, military *ad hoc* networks, low-Earth-orbit space networks, or interplanetary networks.

Many solutions were then proposed to adjust classical protocols to such networks. These include *proxy-agent-based* approaches, where middle boxes entities (dedicated nodes) translate classical protocols to specific ones, and *link-repair* approaches [13], which attempt to fool the classical protocols into believing they are operating over a well-performing physical infrastructure.

In general, unfortunately, the first approach leads to less reusable solutions and does not achieve interoperability satisfactorily, while the second increases the complexity of the architecture too significantly [20].

In fact, the well-known electronic courier service (e-mail) is the closest to address routing challenges in exotic networks. Flexible naming, asynchronous message-based operation, error reporting and interoperability are particularly relevant. E-mail falls short due to its lack of dynamic routing, weakly-defined delivery semantics, and lack of consistent application interface [20]. Therefore Fall called for a paradigm shift, and proposed a new *delay-tolerant network architecture* that uses messages as the primary unit of data interchange.

This architecture was designed to ensure interoperability, performances, and security in heterogeneous networks where end-to-end routing paths may not exist. It operates as an overlay network on top of the transport layer of disparate regional (permanently connected) networks [20] (the routing is then hierarchical). A store-forward mechanism is used to address issues related to disconnections between the regional networks (*cf.* Section 1.2).

For concrete examples, we refer to the following projects:

- *Zebranet* [32]: researchers drive through a forest collecting data about the dispersed zebra population.
- *DakNet* [38]: a public bus carries a mobile access point between villages and a large city that has a high-speed Internet connectivity. This way, the bus provides a disconnected Internet access to isolated villages.
- *Bluespots* [33]: a small computer on a bus serves as a bluetooth content distribution station in a university public transit scenario.
- *Disaster Monitoring Constellation* [44]: a multi-satellite Earth-imaging low-Earth-orbit sensor network where captured image swaths are stored onboard each satellite and later downloaded from the satellite payloads to a ground station.

1.2 Delay-tolerant networks

When end-to-end connections are difficult (impossible) to establish, routing in a delay-tolerant network relies on a *store-forward* approach – *i.e.* messages are transferred from one node to another, depending on the communication opportunities (termed *contact*) that arise, and stored throughout the network in hope that each message will reach its destination(s).

To minimize latency, and/or to maximize the chances of a message being successfully transmitted to its destination(s), the common solution, termed *epidemic routing* [42], is to replicate messages and to spread out many copies over the network. Of course, this approach is not suitable where nodes have limited memory capacity, or where links bandwidth is weak.

In fact, when nodes produce contents that are larger than links capacity, nodes must slice messages in order to transmit fragments (also termed *datum units*) separately. This raises the problem of deciding the datum units to be sent during each contact (whenever nodes meet). Belblidia *et al.* [5] proposed, for example, a popularity-based decentralized heuristic (named Prevalence-Aware Content Spreading) which tends to homogenize the dissemination of each piece of data in the network.

This problem has exercised an increasing number of researchers over the last decade. Many [3, 14, 34] proposed some stochastic methods that estimate different probabilistic metrics to, *in fine*, decide the data to be transmitted. Others [35, 27] considered deterministic time evolving networks and proposed solutions that optimize a given criteria (average delay, robustness, ...). These approaches focus on applications where it is quite possible to make realistic predictions about node mobility. Such applications include satellite networks (where the trajectory of nodes depends on straightforward physics), fleets of drones, and public transportation systems. For example, Jain *et al.* [29] built a scenario with twenty buses, equipped with wireless communication devices, making *scheduled* trips inside San Francisco.

These so-called *deterministic delay-tolerant networks* are the heart of this thesis. We will address the problem of making use of knowledge about node mobility when information needs to be routed from sources to destinations within a given time horizon. The fundamental question is which elements of the information should be transferred from which node to which node when contacts occur. A solution to such a problem is termed a *transfer plan*.

The literature is quite limited at this time. Alonso and Fall [2] proposed a linear formulation for computing a minimum delay transfer plan (routing) with respect to a set of nodes, a set of contacts and a set of messages. Links need to be assigned to data such that every message can travel through the network from one sender to one receiver. The formulation incorporates some constraints that are to be found in real applications, *e.g.* transmission delay (the length of time required by the sending node to process all the bytes that are sent), propagation delay (the amount of time it takes for the head of the signal to travel from the sender to the receiver over the medium concerned), and buffers capacity (embedded memory). As in most of the works presented

below, data transmissions are modelled by unidentified numbers of bytes to be transferred through a dynamic transportation graph. So the problem can be seen as a *dynamic* multi-commodity flow problem [19] in which messages are commodities, and edge capacities are time-varying. The main drawback here is that flow conservation constraints forbid duplication of data, making such approaches unsuitable for multicast and multisource situations.

Alonso's and Fall's works were subsequently extended by Jain *et al.* [29], who in particular proposed *oracles* to compare the performances of routing algorithms in terms of the amount of knowledge of network topology that is required. For example, the *contacts oracle* can answer any question regarding the contacts. Computational tests showed, as expected, that the greater the available knowledge, the better the performances. Zhao *et al.* [47] extended the oracles of Jain *et al.* to take multicasting protocols into account, *e.g.* the *membership oracle* answer questions related to group dynamics. In this thesis we consider that all the oracles are available.

Interferences are often neglected (since delay-tolerant networks are often sparse). In order to address higher-dimensional problems, other assumptions were proposed. For example, Handorean *et al.* [26] defined *atomic contacts*, where contact durations (as opposed to inter-contact durations) are assumed to be instantaneous (both propagation and transmission delays are therefore disregarded). Later, Hay and Giaccone [27] made the same assumptions, and proposed a remarkable model that they called the *event-driven graph*. As the graph is *time-independent* and polynomial in size with respect to the number of contacts in the instance, very basic tools from graph theory can be used to solve numerous problems straightforwardly. So, for example, the authors solve shortest-path or maximum-flow subproblems to minimize the delay or to maximize the network throughput.

Other models were proposed. Merugu *et al.* [35] proposed the *space-time graph*, *i.e.* a graph comprised of several *snapshots* (instantaneous connectivity graphs) placed side by side, and interconnected by *temporal edges*. Ferreira [22] proposed the *evolving graph*, an effective combinatorial model capturing the most significant characteristics of time-varying networks. This model will be described in Section 1.4.

Finally – for those who want to go further – we refer to the *delay-tolerant networking research group* [16], Voyiatzis' survey [43], and to Zhang's survey [46] for their extensive review of the literature. The large number of papers that they reference, reflects a high level of interest in problems of routing in delay-tolerant networks.

However, to our knowledge, no paper has so far addressed the multisource case, despite its relevance if resulting algorithms are to be executed on-line, such as when routing tables need to be refreshed dynamically, following new predictions on node mobility or connectivity.

1.3 Systems of systems

We actually address problems of routing in delay-tolerant networks to better identify issues related to communications in *systems of systems* (these works being part of a multi-disciplinary scientific program that is focused on *Control of Technological Systems of Systems* [37]).

Systems of systems have been defined in many ways. However, a practical definition may be that systems of systems are *supersystems* comprising other elements that are themselves complex, independent operational systems, all interacting to achieve a common goal [30]. To this end, the systems (satellites, drones, sensors, ...) exchange information and collaborate.

From a data transportation point of view, when systems are mobile and intermittently connected, a system of systems can be considered as a delay-tolerant network comprised of several heterogeneous systems. Every system is characterized by:

- the *capacity* of its *buffer*, which quantifies the amount of bytes that the system is able to store in its non-volatile memory (when this overflows, a specific algorithm has to select some datum units to be dropped);
- the *life expectancy* of its battery, which limits the activity of the whole system for a given period;
- a set of data that it wishes to store during a given period (or before a given deadline), and a set of data it already stores;
- and some *transmitters* (*e.g.* Bluetooth and/or IEEE 802.11 devices).

Transmitters are communication devices (interfaces) that enable the systems to collaborate and share data, by making opportunistic use of the possibilities for communication (*contacts*) that arise when two entities are close enough to each other. Every contacts is characterized by:

- the *time interval* during which the link is active;
- its *transmission delay* – the length of time required by the sending node to process all the bytes that are sent (the delay caused by the bit-rate, also known as the *bandwidth* of the link);

- and its *propagation delay* – the length of time it takes for the signal to travel from the sender to the receiver (this may be significant in some applications, particularly in extraterrestrial networks where signals may take several hours to reach their destination, *e.g.* the end-to-end round trip time from Jupiter and Pluto to Earth vary between 81.6 and 133.3 minutes, and between 593.3 and 1044.4 minutes respectively, according to the orbital location of the planets [1]).

Contacts enable elements of a system of systems to collaborate and to route information from a subset of *source* nodes to a subset of *recipient* nodes. Such a collaboration becomes necessary, for example, where contact durations are relatively short with respect to the quantity of information to be transferred, and where, consequently, the complete data cannot be transferred in a single contact. To tackle this problem, modern protocols subdivide data into several *datum units* which are sent in any order to recipient systems [5].

It is worth nothing here that the datum units may be sent to non-recipient messenger systems whose role is to store and pass on the datum units.

The challenge is to find a valid transfer plan (valid store-forward routing paths) to transfer data from their sources to their destinations. Some criteria may have to be optimized, such as:

- the dissemination lengths (the average or the maximum delay);
- the number of contacts that are actually tapped (energy saving policy);
- or the robustness (the impact of network failures on the solution).

In this thesis, we only focus on the first criterion, *i.e.* the maximum delay is to be minimized. Note, however, that we will also consider some robustness constraints in Chapter 5 (at first networks failures will be disregarded).

In Table 1.1 we provide an overview of the constraints to be found in the literature. In particular, we report the papers which studied these constraints or, conversely, which neglected them. In the following, we make some of these assumptions too.

- The network is assumed to be very sparse. Therefore interferences are disregarded [2, 26, 27, 29, 35, 36, 47].
- Contact durations (as opposed to inter-contact durations) are assumed to be instantaneous. So both propagation and transmission delays are neglected – *i.e.* contacts are *atomic* [26, 27, 36].
- Buffers [26, 35, 36] and batteries [2, 26, 29, 35, 36, 47] are infinite.

	[2]	[26]	[27]	[29]	[35]	[36]	[47]	**our model**
uni/multi-cast	U	U	U	U	U	M	M	**multicast**
flow	yes	yes	yes		yes			**no**
interferences								**no**
atomic contacts		yes	yes			yes		**yes**
buffers	yes		yes	yes			yes	**no** (infinite)
batteries			yes					**no** (infinite)

Table 1.1 – State-of-the-Art (a short summary).

- We consider one datum, sliced into several datum units. These are then sent separately [5]. During every contact, at most one datum unit can be transferred, from one node to another. In this way, we do not follow the traditional flow-based approach [29, 36, 47], and every datum unit can have multiple sources and multiple destination. In fact, we consider a many-to-many – as opposed to one-to-many [36, 47] – approach.

1.4 The dissemination problem

Let us now formally define our problem.

First we consider a set $\mathcal{N} = \{1, 2, \ldots, n\}$ of n interacting mobile systems, termed the *nodes*, and one *datum* $\mathcal{D} = \{1, 2, \ldots, u\}$ of u datum units. Each *datum unit* (also termed "unit") represents a unitary, indivisible fragment of data. Each node $i \in \mathcal{N}$ possesses a subset $\mathcal{O}_i \subseteq \mathcal{D}$ of units from the outset. Subset $\mathcal{R} \subseteq \mathcal{N}$ defines the nodes wishing to obtain the datum $\mathcal{D}$ (*i.e.* all the datum units) inside the given time horizon. For the sake of clarity, the term *source nodes* (or "sources") will refer to the nodes $i \in \mathcal{N} \mid \mathcal{O}_i \neq \emptyset$. The nodes in $\mathcal{R}$ are termed the *recipient nodes* (or "recipients").

Remark 1.1

To simplify the formulation of this problem, we assume that the recipient nodes need to obtain all the datum units. Note, however, that the results described in this manuscript can all be generalised (with minor updates) to the case where the recipients only need a subset of the units.

To ensure the dissemination of the datum $\mathcal{D}$, nodes may exchange datum units whenever they are close enough to communicate (such a communication opportunity is termed a *contact*). We assume that the contacts are perfectly known, or easily predictable at any time, since the trajectory of each node is deterministic. Thus, we consider a *sequence of contacts* $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ of m ordered pairs of $\mathcal{N}^2$. During contact $(s, r) \in \sigma$, the sending node s can send to the receiving node r at most one datum unit that it already possesses (either from the outset or as a result of previous contacts). Once the contact has occurred, node r also possesses unit k. Below nodes s_c and r_c denote the sender and the receiver in contact $\sigma_c \in \sigma$, *i.e.* $\sigma_c = (s_c, r_c)$.

Remark 1.2

To represent an undirected contact $[i, j]$, we can consider a first directed contact (i, j), followed by a reverse contact (j, i). Similarly, to represent a longer contact (i, j), during which several datum units might be sent, we can duplicate the contact (one contact per possible transfer).

A *transfer plan* $\phi : \{1, 2, 3, \ldots, m\} \mapsto \{\emptyset, \{1\}, \{2\}, \ldots, \{u\}\}$ is a function where $\phi(c)$ designates the datum unit received by node r_c during contact σ_c. If $\phi(c) = \emptyset$, then nothing is transmitted during contact σ_c. From now on T_ϕ denotes the target set $\{\emptyset, \{1\}, \{2\}, \ldots, \{u\}\}$ of ϕ. Each transfer plan ϕ has a corresponding set of states $O_i^t \subseteq \mathcal{D}$, defined for each node $i \in \mathcal{N}$, and each time index $t \in \{0, 1, \ldots, m\}$, such that:

$$\begin{array}{l} (1)\ \forall i \in \mathcal{N},\ O_i^0 = \mathcal{O}_i, \\ (2)\ \forall c \in \{1, 2, ..., m\},\ O_{r_c}^c = O_{r_c}^{c-1} \cup \phi(c), \\ (3)\ \forall c \in \{1, ..., m\},\ \forall i \in \mathcal{N} \backslash \{r_c\},\ O_i^c = O_i^{c-1} \end{array} \tag{1.1}$$

Thus each state O_i^t contains the datum units received by node i during the first t contacts of sequence σ (in addition to the datum units that node i has possessed from the outset). The transfer plan is *valid* where nodes transmit only datum units that they possess, *i.e.*

$$\forall \sigma_c \in \sigma,\ \phi(c) \in \{\emptyset\} \cup \{\{k\} \,|\, k \in O_{s_c}^{c-1}\} \tag{1.2}$$

A valid transfer plan ϕ has a *delivery length* $\lambda_i(\phi)$ for each node $i \in \mathcal{N}$, corresponding to the smallest time index t at which node i possesses all the units $k \in \mathcal{D}$, *i.e.* $\lambda_i(\phi) = \min\{t \in \{0, 1, \ldots, m\} \,|\, O_i^t = \mathcal{D}\}$. If this index does not exist, then it is assumed that $\lambda_i(\phi) = \infty$. The *dissemination length* $\lambda(\phi)$ of the transfer plan corresponds to the smallest time index t at which all the recipient nodes are delivered, *i.e.* $\lambda(\phi) = \max_{i \in \mathcal{R}} \{\lambda_i(\phi)\}$.

The *dissemination problem* is to find a valid transfer plan ϕ minimizing the dissemination length $\lambda(\phi)$. The problem is NP-Hard in the strong sense, but it can be polynomially solved if $u = 1$ or $|\mathcal{R}| = 1$ (*cf.* Chapter 2). It can be seen as the *offline* version of the problem tackled by Belblidia *et al.* [5].

Evolving graphs

An instance of this problem can be described by an evolving graph [22], that is a multigraph whose vertices represent nodes, and whose arcs represent connections between these nodes. An arc is labelled with time intervals that indicate when the link is really active. To appropriately take account of time constraints, the notion of *path* is replaced by the notion of *journey*, that is an ordered set of arcs having *increasing* labels. For our requirements, every contact is thus represented by one arc whose label is given by its position in the sequence σ.

In Figure 1.1, $[\sigma_1 = (1, 6), \sigma_3 = (6, 5)]$ is a journey (because $3 \geq 1$). This represents the fact that node 6 can forward the unit it receives from node 1 at time 1 to node 5 at time 3. Nonetheless, $[\sigma_{13} = (5, 6), \sigma_1 = (1, 6)]$ is not a journey (since $1 < 13$). More generally, given a datum unit $k \in \mathcal{D}$, a journey $[(i, u), \ldots, (v, j)]$ between a source node $i \in \mathcal{N} \,|\, k \in \mathcal{O}_i$ and a recipient $j \in \mathcal{R}$ represents a store-forward routing to transfer unit k from i to j.

Therefore, as will be shown in Chapter 2, a set of arc-disjoint branchings (in the evolving graph) that are rooted on the source nodes of a given datum unit $k \in \mathcal{D}$, and such that the whole covers all the recipient nodes, defines a store-forward routing to disseminate unit k. For example, in Figure 1.1, the bold arcs define a set of branchings to disseminate datum unit 1 from nodes 1 and 2 to all the other nodes. Thus, solving the dissemination problem can be seen as finding such a set of arc-disjoint branchings *for each datum unit* in an evolving graph (see the branchings with bold and doubled arcs).

(a) *an instance of the dissemination problem*

$$\begin{cases} \mathcal{N} = \mathcal{R} = \{1,2,3,4,5,6\}; \mathcal{D} = \{1,2\}; \\ \mathcal{O}_1 = \{1,2\}; \mathcal{O}_2 = \{1\}; \mathcal{O}_3 = \mathcal{O}_4 = \mathcal{O}_5 = \mathcal{O}_6 = \emptyset\,; \\ \sigma = [(1,6),(6,1),(6,5),(1,3),(3,5),\ldots,(5,6)] \end{cases}$$

(b) *a valid transfer plan*

$$\begin{cases} \phi(4) = \phi(5) = \phi(7) = \phi(10) = \{1\}; \\ \phi(1) = \phi(3) = \phi(6) = \phi(8) = \phi(11) = \{2\}; \\ \phi(2) = \phi(9) = \phi(12) = \phi(13) = \emptyset \end{cases}$$

(c) *the corresponding evolving graph [22]*

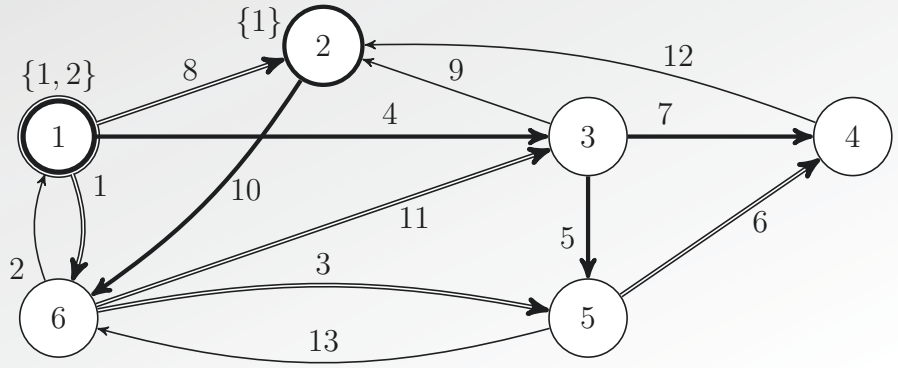


Figure 1.1 – The dissemination problem (an example).

2 Complexity results

As mentioned in Section 1.4, we are going to prove the "dissemination problem" is NP-Hard in the strong sense (*cf.* Section 2.1). Thereafter, we will show that this problem can be solved in *polynomial time* when there is only one datum unit, or only one recipient, or when the number u of datum units and the number $|\mathcal{R}|$ of recipients are lower than a given constant (*cf.* Section 2.2). Besides this, knowing whether there exist k mutually arc-disjoint branchings in an evolving graph – or whether there exist k mutually arc-disjoint Steiner trees in a digraph *without circuit* – will be shown to be NP-Complete in the strong sense (*cf.* Section 2.3).

Contents

2.1 The data transfer problem 15
2.1.1 The case $u \geq 2$ 16
2.1.2 The case $|\mathcal{R}| \geq 2$ 20
2.2 Polynomial-time cases 27
2.2.1 The one-datum-unit problem ($u = 1$) 27
2.2.2 The delivery problem ($|\mathcal{R}| = 1$) 29
2.2.3 Upper bounded parameters 34
2.3 Additional results 36
2.3.1 Arc-disjoint branchings in an evolving graph . . . 36
2.3.2 Arc-disjoint Steiner trees in a digraph 42
2.4 Conclusion 43

To our knowledge, apart from a few papers on evolving graphs [7, 23, 45], there is no previous work attempting to determine the theoretical complexity of routing problems in deterministic delay-tolerant networks. In this chapter we therefore study the complexity of the dissemination problem. We aim to determine the frontier between easy and hard cases.

In Section 2.1, we will show that the general case is NP-Hard in the strong sense. Thereafter, in Section 2.2, we will show that it is polynomially solvable where $u = 1$ or $|\mathcal{R}| = 1$. Finally, in Section 2.3, we will prove the complexity of two problems also related to delay-tolerant networking.

2.1 The data transfer problem

In this section, we show that the problem, called the *data transfer problem*, of finding a valid transfer plan such that $\forall i \in \mathcal{R}$, $O_i^m = \mathcal{D}$ (all the recipients receive all the units) is strongly NP-Hard. It also leads to the NP-Hardness of the dissemination problem (*i.e.* the optimization version, where the delivery length is minimized).

Let us consider the decision version of the data transfer problem, denoted henceforward as DT:

Problem 2.1 – the data transfer problem

Given a set $\mathcal{N} = \{1, 2, \ldots, n\}$ of n nodes · a subset $\mathcal{R} \subseteq \mathcal{N}$ of recipient nodes · a sequence $\sigma = (\sigma_1, \ldots, \sigma_m)$ of m pairs $(i, j) \in \mathcal{N}^2$ with $i \neq j$ · a set $\mathcal{D} = \{1, 2, \ldots, u\}$ of u datum units · for each node $i \in \mathcal{N}$, a subset $\mathcal{O}_i \subseteq \mathcal{D}$ of datum units initially possessed by node i – **is there a valid transfer plan** $\phi : \{1, \ldots, m\} \mapsto \{\emptyset, \{1\}, \ldots, \{u\}\} \,|\, \forall i \in \mathcal{R}$, $O_i^m = \mathcal{D}$?

The data transfer problem (DT) is obviously in NP, since it can be decided in polynomial time whether a transfer plan ϕ is valid and such that $\forall i \in \mathcal{R}$, $O_i^m = \mathcal{D}$. Below we show the problem is strongly NP-Complete when $u = 2$ and $|\mathcal{R}|$ is not upper bounded, or conversely when $|\mathcal{R}| = 2$ and u is not upper bounded. To achieve this, we reduce a problem known as being strongly NP-Complete to the studied cases.

In [25], Garey and Johnson give a comprehensive guide to the theory of NP-Completeness, and provide an extensive list of NP-Complete/NP-Hard problems. In this chapter, we consider the 3-Satisfiability problem, referred to below as 3-SAT, *cf.* [25], page 259. 3-SAT is strongly NP-Complete and is stated as follows:

Problem 2.2 – 3-Satisfiability

Given a set $X = \{x_1, x_2, \ldots, x_p\}$ of p variables · a set $C = \{c_1, \ldots, c_s\}$ of s clauses over X such that, for each $c \in C$, $|c| = 3$ – **is there a truth assignment that satisfies** C ?

2.1.1 The case $u \geq 2$

We now show that 3-SAT is reducible in polynomial time to the special case of DT where $u = 2$ (DT2U).

Theorem 2.1

The data transfer problem is strongly NP-Hard for $u \geq 2$.

Polynomial-time reduction

Consider an instance of 3-SAT (*cf.* Problem 2.2). From this instance we build an instance of DT2U as follows.

Datum – We have $u = 2$ and then $\mathcal{D} = \{1, 2\}$.

Nodes – We consider the following nodes.

- With each clause $c_j \in C$, $j \in \{1, 2, \ldots, s\}$, is associated a node $\omega_j \in \mathcal{N}$ in DT2U with $\mathcal{O}_{\omega_j} = \{2\}$. The datum units possessed by node ω_j will correspond to the logical value of clause c_j, in such a way that clause c_j will be considered to be *true* if node ω_j possesses datum unit 1, and to be *false* otherwise. In this way, all clauses are *false* at the beginning (since the associated nodes possess only unit 2).
- With each variable $x_i \in X$, $i \in \{1, 2, \ldots, p\}$, are associated 2 nodes l_i and $\bar{l}_i \in \mathcal{N}$ in DT2U with $\mathcal{O}_{l_i} = \mathcal{O}_{\bar{l}_i} = \emptyset$. The datum units received by these nodes will respectively correspond to the logical values of literals x_i and $\bar{x}_i$. The literal among $\{l_i, \bar{l}_i\}$ which will store datum unit 1 will be considered to be *true*, while the other one (which will store datum unit 2) will be considered to be *false*. Initially all literals are considered to be undetermined (the associated nodes possess no datum units).
- In DT2U is also created a node $\alpha \in \mathcal{N}$ with $\mathcal{O}_\alpha = \{1, 2\}$. This node is the only one that initially possesses the two datum units. It will enable the logical value of each literal to be given by providing unit 1 only to the nodes associated with literals which have to be *true*, and then unit 2 only to the nodes associated with literals which have to be *false*.
- The recipients are $\mathcal{R} = \{\omega_j \mid j \in \{1, \ldots, s\}\} \cup \{l_i \mid i \in \{1, \ldots, p\}\}$.

Therefore, exactly $n = s + 2p + 1$ nodes have been created.

Sequence of contacts – The sequence of contacts $\sigma = \sigma^1 \circ \sigma^2 \circ \sigma^3$ is built from the concatenation of 3 subsequences:

- subsequence σ^1 is built such that for each $i \in \{1, 2, 3, \ldots, p\}$, we have $\sigma^1_i = (\alpha, l_i)$ and $\sigma^1_{p+i} = (\alpha, \bar{l}_i)$ (hence $|\sigma^1| = 2p$);
- subsequence σ^2 (of size $3s$) is built such that for each $j \in \{1, 2, \ldots, s\}$, considering that $c_j = \{l_{j1}, l_{j2}, l_{j3}\}$, we have:

$$\text{for } k \in \{1,2,3\},\ \sigma^2_{3(j-1)+k} = \begin{cases} (l_i, \omega_j) \text{ if } l_{jk} = x_i \\ (\bar{l}_i, \omega_j) \text{ if } l_{jk} = \bar{x}_i \end{cases};$$

- and subsequence σ^3 is built such that for each $i \in \{1, 2, \ldots, p\}$, we have $\sigma^3_i = (\bar{l}_i, l_i)$ (hence $|\sigma^3| = p$).

Thus, a sequence of exactly $m = 3p + 3s$ contacts has been created.

For example, let us consider the following instance of 3-SAT:

- $p = 3$ and $X = \{x_1, x_2, x_3\}$;
- $s = 2$ and $C = \{c_1, c_2\}$;
- $c_1 = \{x_1, \bar{x}_2, \bar{x}_3\}$ and $c_2 = \{\bar{x}_1, \bar{x}_2, x_3\}$.

The associated instance for DT2U is (*cf.* Figure 2.1):

- $u = 2$ and $\mathcal{D} = \{1, 2\}$;
- $n = 9$, $\mathcal{N} = \{\omega_1, \omega_2, l_1, l_2, l_3, \bar{l}_1, \bar{l}_2, \bar{l}_3, \alpha\}$, and $\mathcal{R} = \{\omega_1, \omega_2, l_1, l_2, l_3\}$;
- $\mathcal{O}_{\omega_1} = \mathcal{O}_{\omega_2} = \{2\}$, $\mathcal{O}_{l_i} = \mathcal{O}_{\bar{l}_i} = \emptyset$ for all $i \in \{1, 2, 3\}$, and $O_\alpha = \{1, 2\}$;
- $m = 15$, $\sigma = \sigma^1 \circ \sigma^2 \circ \sigma^3$ with:
  - $\sigma^1 = [(\alpha, l_1), (\alpha, l_2), (\alpha, l_3), (\alpha, \bar{l}_1), (\alpha, \bar{l}_2), (\alpha, \bar{l}_3)]$;
  - $\sigma^2 = [(l_1, \omega_1), (\bar{l}_2, \omega_1), (\bar{l}_3, \omega_1), (\bar{l}_1, \omega_2), (\bar{l}_2, \omega_2), (l_3, \omega_2)]$;
  - $\sigma^3 = [(\bar{l}_1, l_1), (\bar{l}_2, l_2), (\bar{l}_3, l_3)]$.

From a valid transfer plan to a truth assignment satisfying C

Suppose that there exists a valid transfer plan ϕ such that $\forall i \in \mathcal{R}$, $O^m_i = \mathcal{D}$ for the DT2U instance defined above. From this transfer plan, we can build a truth assignment that satisfies C as follows.

The logical values of all literals are set during subsequence σ^1. Consider the pair of nodes $\{l_i, \bar{l}_i\}$ ($i \in \{1, 2, \ldots, p\}$) associated with the pair of literals $\{x_i, \bar{x}_i\}$. First, during subsequence σ^1, node l_i has an initial contact with α during which it receives at most one unit of $\mathcal{D}$. The situation is identical for

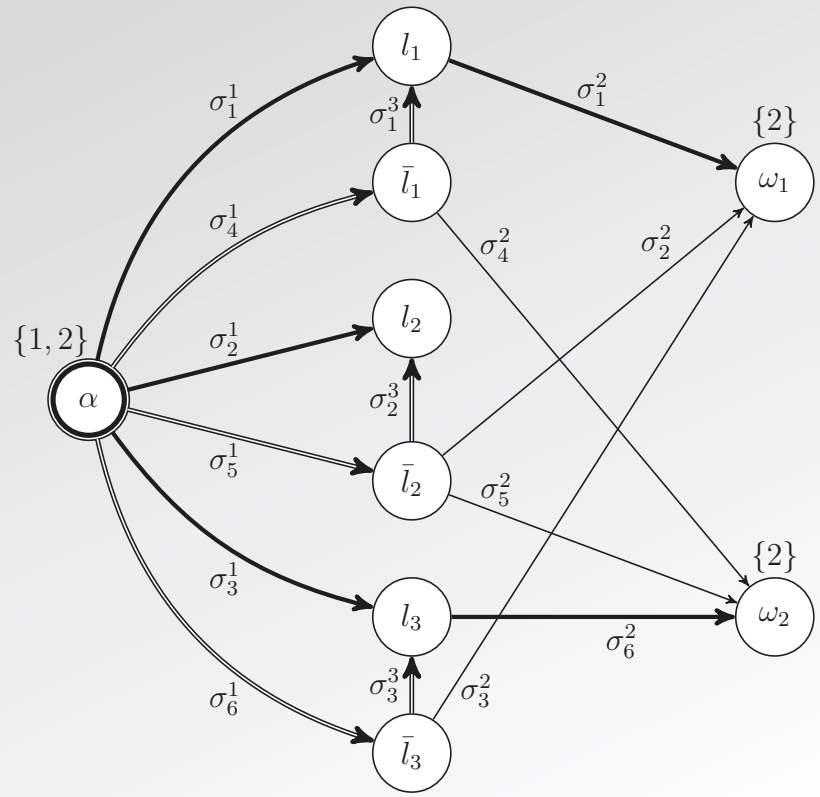


Figure 2.1 – The DT2U instance associated with the 3-SAT instance given on page 17.

node $\bar{l}_i$. Then, during subsequence σ^3, a contact occurs between nodes $\bar{l}_i$ and l_i. Since $l_i \in \mathcal{R}$, the transfer plan is such that $O_{l_i}^m = \mathcal{D}$. Therefore, the only way for such a transfer plan to be obtained is where, during the contacts of σ^1, l_i obtains one datum unit of $\mathcal{D}$ and $\bar{l}_i$ obtains the other one. Next, during the contacts of σ^3, each node $\bar{l}_i$ gives its unit to node l_i. This process ensures that following subsequence σ^1 (at time $t = 2p$):

$$\left[O_{l_i}^{2p} = \{1\} \text{ and } O_{\bar{l}_i}^{2p} = \{2\}\right] \textbf{ or } \left[O_{l_i}^{2p} = \{2\} \text{ and } O_{\bar{l}_i}^{2p} = \{1\}\right]$$

In the first case, x_i is set to *true* (and $\bar{x}_i$ is set to *false*). In the second case, x_i is set to *false*. At the end of the sequence σ^1, exactly one of the literals in $\{x_i, \bar{x}_i\}$ is then considered to be *true*, and the other *false*.

Let us now show that this assignment satisfies C. During the contacts of sequence σ^2, nodes $\{l_i, \bar{l}_i \,|\, i \in \{1, 2, \ldots, p\}\}$ can only transfer the datum unit they possess to the nodes $\{\omega_j \,|\, j \in \{1, 2, \ldots, s\}\}$ associated with the clauses they are related to. During sequence σ^2, each node ω_j, $j \in \{1, \ldots, s\}$, related to clause c_j, is in contact with 3 nodes. These represent the 3 literals which compose the clause. There are therefore 3 possible ways for each node ω_j to retrieve datum unit 1 (meaning that c_j is *true*). After sequence σ^2, all nodes ω_j, $j \in \{1, \ldots, s\}$ are such that $1 \in O_{\omega_j}^{2p+3s} = O_{\omega_j}^m = \mathcal{D}$, since $\omega_j \in \mathcal{R}$. Thus there is a truth assignment for C.

In Figure 2.1 the bold arcs correspond to the contacts where datum unit 1 is transmitted (conversely, the doubled arcs correspond to the contacts where datum unit 2 is transmitted). It gives rise to the truth assignment $x_1 = \mathit{true}$, $x_2 = \mathit{true}$, and $x_3 = \mathit{true}$, which satisfies $C = \{\{x_1, \bar{x}_2, \bar{x}_3\}, \{\bar{x}_1, \bar{x}_2, x_3\}\}$.

From a truth assignment satisfying C to a valid transfer plan

Let us assume there exists a truth assignment that satisfies C. We can build a corresponding valid transfer plan as follows.

- **Subsequence σ^1:** For each $i \in \{1, 2, \ldots, p\}$,

$$\begin{cases} \phi(i) = \{1\} \text{ and } \phi(p+i) = \{2\} \textbf{ if } x_i \text{ is } \mathit{true} \ (\bar{x}_i \text{ is } \mathit{false}); \\ \textbf{or } \phi(i) = \{2\} \text{ and } \phi(p+i) = \{1\} \text{ otherwise.} \end{cases}$$

- **Subsequence σ^2:** For $j \in \{1, 2, 3, \ldots, s\}$, considering that $\{l_{j1}, l_{j2}, l_{j3}\}$ are the 3 nodes associated with the literals of clause c_j,

$$\forall k \in \{1, 2, 3\}, \begin{cases} \phi(|\sigma^1| + 3(j-1) + k) = \{1\} \textbf{ if } l_{jk} \text{ stores unit 1;} \\ \phi(|\sigma^1| + 3(j-1) + k) = \emptyset \text{ otherwise.} \end{cases}$$

  If clause c_j is satisfied, at least one node l_{jk} can transmit datum unit 1 to ω_j. Therefore, after subsequence σ^2, every node ω_j, $j \in \{1, 2, \ldots, s\}$, possesses all the datum units, *i.e.* $O_{\omega_j}^m = \mathcal{D}$.

- **Subsequence σ^3:** For each $i \in \{1, \ldots, p\}$, we set:

$$\phi(|\sigma^1 \circ \sigma^2| + i) = O_{\bar{l}_i}^{2p}$$

  Recall that $\bar{l}_i$ possesses the datum unit that l_i needs. Therefore every node l_i, $i \in \{1, 2, \ldots, p\}$, possesses all the datum units after σ^3.

Every node $i \in \mathcal{R}$ possesses all the datum units. Thus the resulting transfer plan is such that $\forall i \in \mathcal{R}$, $O_i^m = \mathcal{D}$. □

2.1.2 The case $|\mathcal{R}| \geq 2$

We now show that 3-SAT is reducible in polynomial time to the special case of DT where $|\mathcal{R}| = 2$ (DT2R). For this purpose we adapt the proof of Even, Itai and Shamir [19] showing that the two-commodity integral flow problem is strongly NP-Complete. The significant difference is that instead of dealing with units of flow, we are now dealing with *identified* datum units which can be *duplicated* (*i.e.* conservation constraints no longer hold).

Theorem 2.2

The data transfer problem is strongly NP-Hard for $|\mathcal{R}| \geq 2$.

Polynomial-time reduction

Consider an instance of 3-SAT (*cf.* Problem 2.2, page 15). From this instance we build an instance of DT2R as follows.

Datum – We set $u = s+1$ and $\mathcal{D} = \{0, 1, \ldots, s\}$ (exceptionally, for the sake of simplicity, datum units are numbered starting with 0).

Nodes – We consider the following nodes.

- For $i \in \{1, 2, \ldots, p\}$, let a_i and b_i be respectively the number of occurrences of x_i and of $\bar{x}_i$ in the clauses of C. With every variable $x_i \in X$, $i \in \{1, 2, \ldots, p\}$, is associated set

$$L_i = \{\beta_i, l_i^1, l_i^2, l_i^3, \ldots, l_i^{2a_i}, \bar{l}_i^1, \ldots, \bar{l}_i^{2b_i}, \epsilon_i\} \subseteq V$$

  of $2(1 + a_i + b_i)$ nodes in DT2R (*cf.* Figure 2.2). Initially, these nodes possess no units – *i.e.* $\forall x \in L_i$, $\mathcal{O}_x = \emptyset$.
- With each clause $c_j \in C$, $j \in \{1, 2, \ldots, s\}$, is associated a node $C_j \in \mathcal{N}$ in DT2R with $\mathcal{O}_{C_j} = \emptyset$ (*cf.* Figure 2.3).
- Moreover four nodes α_1, α_2, ω_1 and $\omega_2 \in \mathcal{N}$ are created in DT2R with $\mathcal{O}_{\alpha_1} = \mathcal{O}_{\omega_2} = \{0\}$, $\mathcal{O}_{\alpha_2} = \mathcal{O}_{\omega_1} = \{1, \ldots, s\}$ (*cf.* Figures 2.2 and 2.3).
- Finally the recipient nodes are $\mathcal{R} = \{\omega_1, \omega_2\}$.

Therefore, exactly $n = 4 + s + 2\sum_{i=1}^{p}(1 + a_i + b_i) = 4 + 2p + 7s$ nodes have been created (since there are three literals per clause).

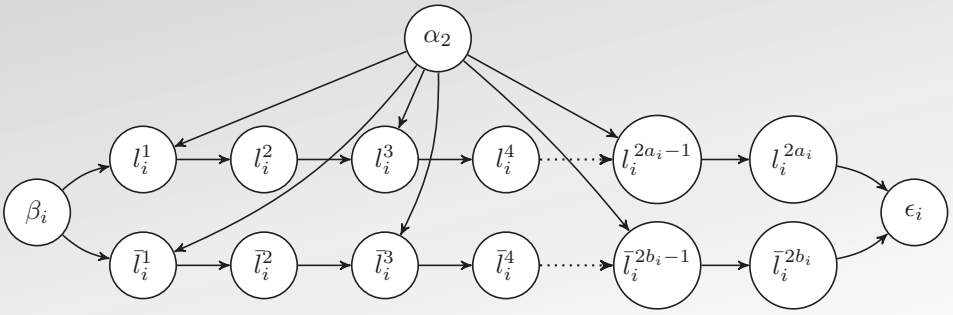


Figure 2.2 – The set of nodes L_i (associated with variable x_i), node α_2, and the corresponding contacts.

Sequence of contacts – The sequence $\sigma = \sigma^1 \circ \sigma^2 \circ \sigma^3 \circ \sigma^4$ of contacts is the concatenation of several subsequences built as follows.

- Subsequence $\sigma^1 = \sigma^{1,1,1} \circ \sigma^{1,2,1} \circ \sigma^{1,1,2} \circ \sigma^{1,2,2} \circ \cdots \circ \sigma^{1,1,i} \circ \sigma^{1,2,i} \circ \cdots \circ \sigma^{1,1,p} \circ \sigma^{1,2,p}$ is built such that for $i \in \{1, 2, \ldots, p\}$,
  - we have $\sigma_j^{1,1,i} = (\alpha_2, l_i^{2j-1})$ with $j \in \{1, 2, \ldots, a_i\}$;
  - and $\sigma_j^{1,2,i} = (\alpha_2, \bar{l}_i^{2j-1})$ with $j \in \{1, 2, \ldots, b_i\}$ (*cf.* Figure 2.2).

  Therefore σ^1 contains $\sum_{i=1}^{p} (a_i + b_i) = 3s$ contacts.
- Next σ^2 is itself the concatenation of several subsequences.

  Let

$$\sigma^{2,i} = [(\beta_i, l_i^1), (l_i^1, l_i^2), (l_i^2, l_i^3), \ldots, (l_i^{2a_i-1}, l_i^{2a_i}), (l_i^{2a_i}, \epsilon_i), \\ (\beta_1, \bar{l}_i^1), (\bar{l}_i^1, \bar{l}_i^2), (\bar{l}_i^2, \bar{l}_i^3), \ldots, (\bar{l}_i^{2b_i-1}, \bar{l}_i^{2b_i}), (\bar{l}_i^{2b_i}, \epsilon_i)]$$

  be a subsequence of $2(a_i + b_i + 1)$ contacts associated with variable x_i, and built as follows (*cf.* Figure 2.2):
  - If $a_i = 0$ (literal x_i never occurs in C) then contact (β_i, ϵ_i) occurs, else contact (β_i, l_i^1) occurs, followed in succession by the contacts (l_i^j, l_i^{j+1}), $j \in \{1, 2, \ldots, 2a_i - 1\}$, and finally by contact $(l_i^{2a_i}, \epsilon_i)$.
  - Subsequently, if $b_i = 0$, contact (β_i, ϵ_i) occurs, else contact $(\beta_1, \bar{l}_i^1)$ occurs, followed in succession by the contacts $(\bar{l}_i^j, \bar{l}_i^{j+1})$, $j \in \{1, \ldots, 2b_i - 1\}$, and finally by contact $(\bar{l}_i^{2b_i}, \epsilon_i)$.

The overall sequence σ^2 is such that (*cf.* Figure 2.3):

$$\sigma^2 = [(\alpha_1, \beta_1)] \circ \sigma^{2,1} \circ [(\epsilon_1, \beta_2)] \circ \sigma^{2,2} \circ [(\epsilon_2, \beta_3)] \circ \ldots \\ \ldots \circ \sigma^{2,p-1} \circ [(\epsilon_{p-1}, \beta_p)] \circ \sigma^{2,p} \circ [(\epsilon_p, \omega_1)]$$

First a contact occurs from α_1 to β_1. Thereafter each subsequence $\sigma^{2,j}$, $j \in \{1, \ldots, p\}$, is successively applied with a contact $(\epsilon_j, \beta_{j+1})$ between each pair of sequences $(\sigma^{2,j}, \sigma^{2,j+1})$, $j \in \{1 \ldots, p-1\}$. Finally a contact occurs from ϵ_p to ω_1.

Therefore σ^2 contains $2 + p - 1 + \sum_{i=1}^{p} 2(a_i + b_i + 1) = 2 + 3p + 6s - 1$ contacts (recalling that there are three literals per clause).

- Subsequence σ^3 is built as follows. For the x^{th} occurrence of literal x_i (literal $\bar{x}_i$), there is one contact from l_i^{2x} ($\bar{l}_i^{2x}$) to the node C_j associated with the clause c_j in which x_i ($\bar{x}_i$) occurs.

  In Figure 2.3, contacts (l_1^2, C_2), $(\bar{l}_2^2, C_2)$ and $(\bar{l}_p^{2b_p}, C_2)$ represent the fact that clause c_2 is such that $c_2 = \{x_1, \bar{x}_2, \bar{x}_p\}$, and such that they are the 1^{st} occurrences of x_1 and $\bar{x}_2$, and the $(b_p)^{th}$ occurrence of $\bar{x}_p$ in C.

  Therefore, subsequence σ^3 contains $3s$ contacts.

- Subsequence $\sigma^4 = [(c_1, \omega_2), (c_2, \omega_2), \ldots, (c_s, \omega_2)]$. Thus subsequence σ^4 contains exactly s contacts (*cf.* Figure 2.3).

In total, a sequence of $m = 13s + 3p + 1$ contacts has been created.

For example, let us consider the 3-SAT instance defined in Section 2.1.1 – see page 17. The associated instance for DT2R is (*cf.* Figure 2.4):

- $u = 3$ and $\mathcal{D} = \{0, 1, 2\}$;
- $n = 4 + 2p + 7s = 24$ and $\mathcal{N} = \{\, \alpha_1, \alpha_2, \omega_1, \omega_2, C_1, C_2, \beta_1, \beta_2, \beta_3, \epsilon_1, \epsilon_2, \epsilon_3, l_1^1, l_1^2, \bar{l}_1^1, \bar{l}_1^2, \bar{l}_2^1, \bar{l}_2^2, \bar{l}_2^3, \bar{l}_2^4, l_3^1, l_3^2, \bar{l}_3^1, \bar{l}_3^2 \,\}$;
- $\mathcal{O}_{\alpha_1} = \mathcal{O}_{\omega_2} = \{0\}$, $\mathcal{O}_{\alpha_2} = \mathcal{O}_{\omega_1} = \{1, 2\}$, $\mathcal{O}_i = \emptyset$ for all other nodes;
- $m = 15$ and $\sigma = \sigma^1 \circ \sigma^2 \circ \sigma^3 \circ \sigma^4$ with:
  - $\sigma^1 = [(\alpha_2, l_1^1), (\alpha_2, \bar{l}_1^1), (\alpha_2, \bar{l}_2^1), (\alpha_2, \bar{l}_2^3), (\alpha_2, l_3^1), (\alpha_2, \bar{l}_3^1)]$;
  - $\sigma^2 = [\, (\alpha_1, \beta_1), (\beta_1, l_1^1), (l_1^1, l_1^2), (l_1^2, \epsilon_1), (\beta_1, \bar{l}_1^1), (\bar{l}_1^1, \bar{l}_1^2), (\bar{l}_1^2, \epsilon_1),$
    $(\epsilon_1, \beta_2), (\beta_2, \epsilon_2), (\beta_2, \bar{l}_2^1), (\bar{l}_2^1, \bar{l}_2^2), (\bar{l}_2^2, \bar{l}_2^3), (\bar{l}_2^3, \bar{l}_2^4), (\bar{l}_2^4, \epsilon_2),$
    $(\epsilon_2, \beta_3), (\beta_3, l_3^1), (l_3^1, l_3^2), (l_3^2, \epsilon_3), (\beta_3, \bar{l}_3^1), (\bar{l}_3^1, \bar{l}_3^2), (\bar{l}_3^2, \epsilon_3), (\epsilon_3, \omega_1)]$;
  - $\sigma^3 = [(l_1^2, C_1), (\bar{l}_2^2, C_1), (\bar{l}_3^2, C_1), (\bar{l}_1^2, C_2), (\bar{l}_2^4, C_2), (l_3^2, C_2)]$;
  - $\sigma^4 = [(C_1, \omega_2), (C_2, \omega_2)]$.

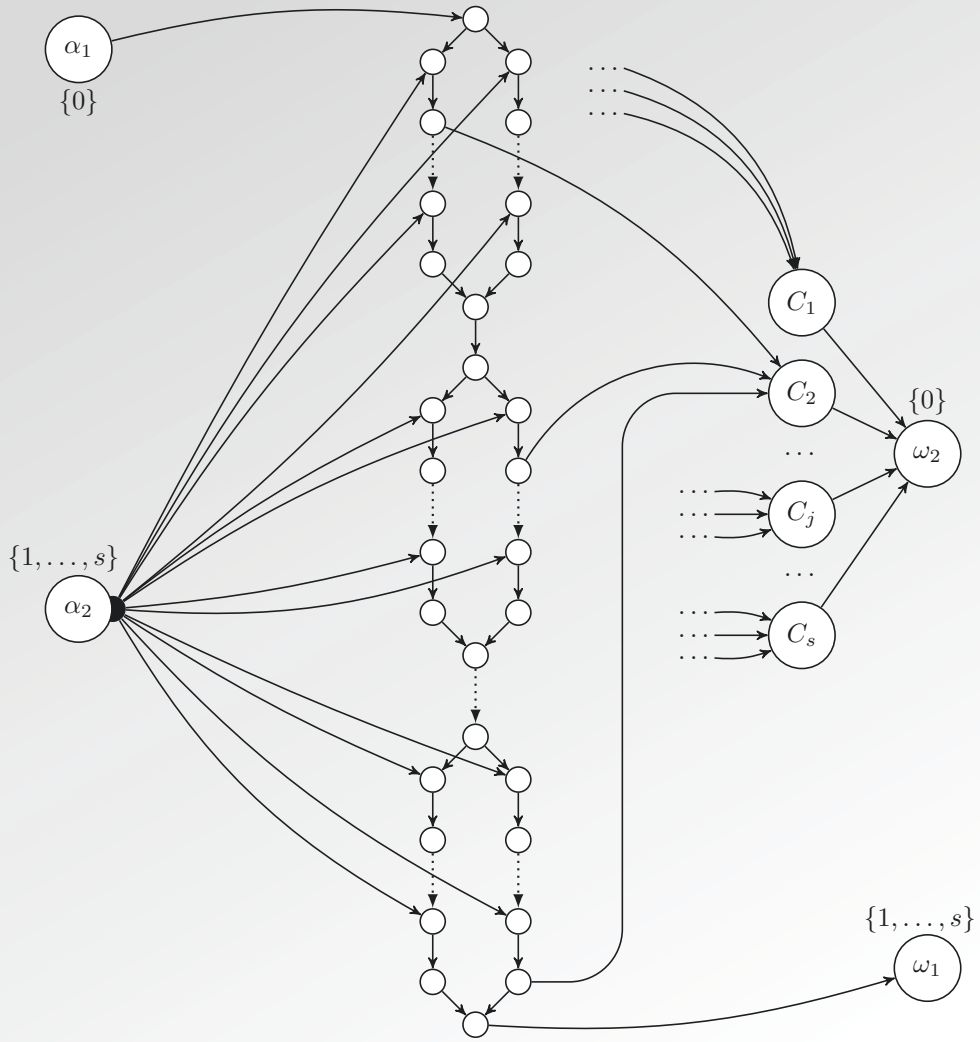


Figure 2.3 – The DT2R instance built from a 3-SAT instance.

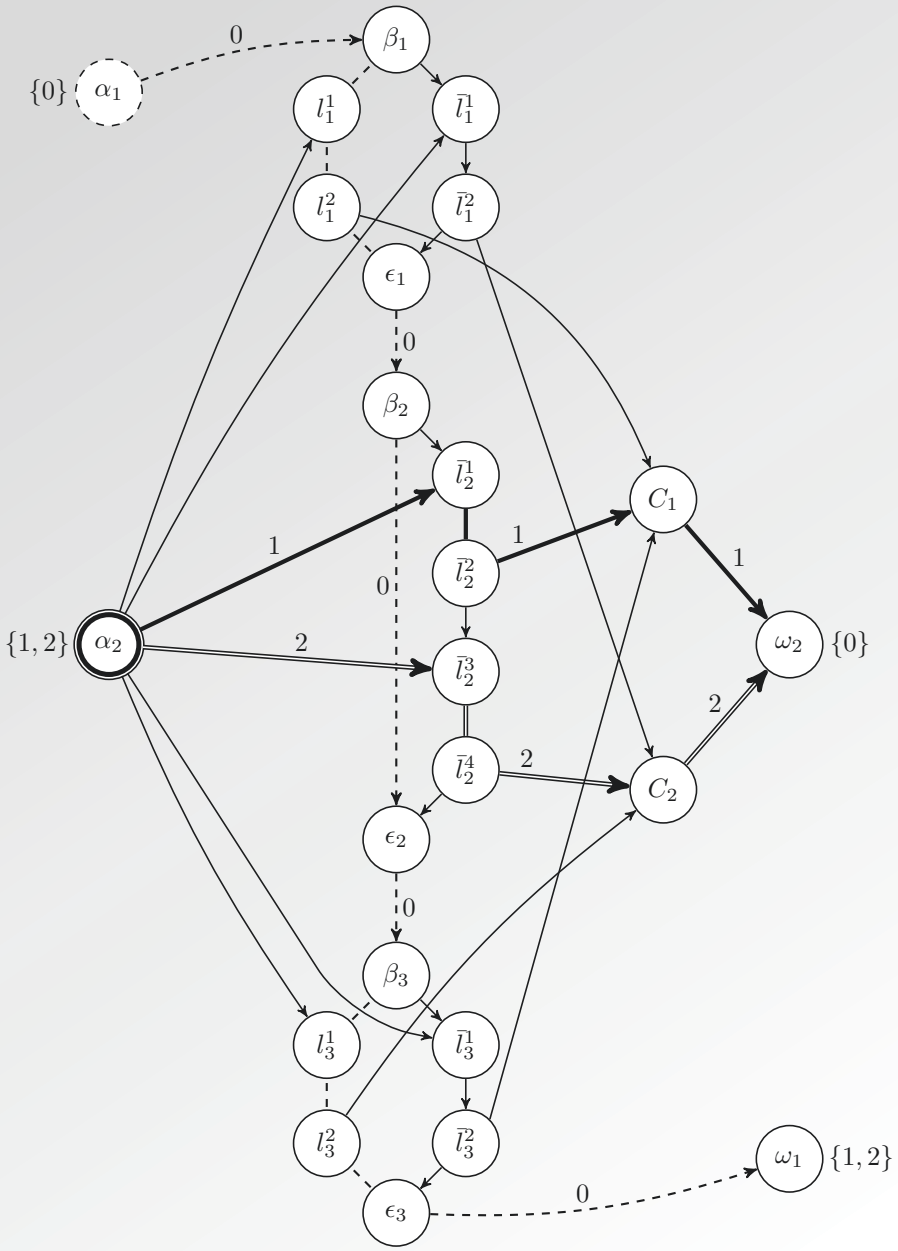


Figure 2.4 – The DT2R instance associated with the 3-SAT instance given on page 17.

From a valid transfer plan to a truth assignment satisfying C

Suppose that there exists a valid transfer plan ϕ such that $\forall i \in \mathcal{R}$, $O_i^m = \mathcal{D}$ for the DT2R instance defined above. From this transfer plan, we can build a truth assignment that satisfies C as follows.

Node ω_1 needs to obtain datum unit 0, because initially it possesses only units $\{1, 2, \ldots, s\}$. This can only occur where unit 0 is transmitted from α_1 to β_1 during contact (α_1, β_1), and then from β_1 to ω_1 using some contacts of subsequence σ^2. Thus, for $i \in \{1, \ldots, p\}$, unit 0 is transmitted through:

$$\begin{cases} [(\beta_i, l_i^1), (l_i^1, l_i^2), (l_i^2, l_i^3), \ldots, (l_i^{2a_i-1}, l_i^{2a_i}), (l_i^{2a_i}, \epsilon_i)] \\ \textbf{or through } [(\beta_1, \bar{l}_i^1), \bar{l}_i^1, \bar{l}_i^2), (\bar{l}_i^2, \bar{l}_i^3), \ldots, (\bar{l}_i^{2b_i-1}, \bar{l}_i^{2b_i}), (\bar{l}_i^{2b_i}, \epsilon_i)] \end{cases}$$

This means that at least one of these two subsequences is devoted exclusively to transmitting unit 0 (no other unit can then transit through these nodes). Unit 0 next uses contact $(\epsilon_i, \beta_{i+1})$ if $i < p$, or contact (ϵ_i, ω_1) if $i = p$.

Node ω_2 needs to obtain the subset $\{1, 2, \ldots, s\}$ of datum units, because initially it possesses only datum unit 0. Each unit must therefore be obtained from a different node C_j, since there are exactly s contacts from these nodes to ω_2. The datum unit k which is transmitted during contact (C_j, ω_2) comes initially from node α_2. It has transited either through

$$\begin{cases} \text{some nodes from } l_i^{2x-1} \text{ to } l_i^{2y} \ (x, y \in \{1, 2, \ldots, a_i\}, y > x), \\ \textbf{or through some nodes from } \bar{l}_i^{2x-1} \text{ to } \bar{l}_i^{2y} \ (x, y \in \{1, \ldots, b_i\}, y > x), \end{cases}$$

so as finally to be transmitted to C_j. Note that if datum unit k is transmitted to C_j through a node l_i^{2y} or a node $\bar{l}_i^{2y} \in L_i$ with $i > 1$, then it cannot come from a node l_z^{2x-1} or $\bar{l}_z^{2x-1} \in L_z$ with $1 \leq z < i$ (since datum unit 0 is always transmitted during contact $(\epsilon_{i-1}, \beta_i)$).

Actually, the path of contacts used to transmit datum unit k to node C_j makes it possible to identify the literal that makes clause c_j *true*.

- If k is transmitted through nodes from l_i^{2x-1} to l_i^{2y} $(x, y \in \{1, \ldots, a_i\}$, with $y > x)$, then variable x_i is set to *true*.
- Conversely, if k has been transmitted through from nodes $\bar{l}_i^{2x-1}$ to $\bar{l}_i^{2y}$ $(x, y \in \{1, \ldots, b_i\}$, with $y > x)$, then variable x_i is set to *false*.

We should remember that unit 0 has transited using some of the contacts in subsequence σ^{2i}, and that x_i obviously cannot be *true* and *false* at the same time. It should also be noted that the same literal can cause several clauses to be *true*. In this case, several units – one per *true* clause – transit through

nodes of L_i, using one disjoint path of contacts per unit. In other cases, none of the datum units in $\{1, \ldots, s\}$ transits through nodes of L_i. It means that variable x_i can be arbitrarily set to *true* or *false*.

Given that for each clause c_j we have found exactly one literal that makes c_j *true*, we can therefore conclude that there is a truth assignment for C.

For the previous example, *cf.* Figure 2.4, datum unit 1 can be transmitted through contacts $[(\alpha_2, \bar{l}_2^1), (\bar{l}_2^1, \bar{l}_2^2), (\bar{l}_2^2, C_1), (C_1, \omega_2)]$, while datum unit 2 can be transmitted through $[(\alpha_2, \bar{l}_2^3), (\bar{l}_2^3, \bar{l}_2^4), (\bar{l}_2^4, C_2), (C_2, \omega_2)]$ – making $x_2 = \mathit{false}$ – and then making both c_1 and c_2 *true*. In addition unit 0 can be transmitted throughout sequence $[(\alpha_1, \beta_1), (\beta_1, l_1^1), (l_1^1, l_1^2), (l_1^2, \epsilon_1), (\epsilon_1, \beta_2), (\epsilon_2, \beta_3), (\epsilon_2, \beta_3), (\beta_3, l_3^1), (l_3^1, l_3^2), (l_3^2, \epsilon_3), (\epsilon_3, \omega_2)]$. Finally x_1 and x_3 can be set arbitrarily.

From a truth assignment satisfying C to a valid transfer plan

Let us assume there exists a truth assignment that satisfies C. We can build a corresponding valid transfer plan as follows.

First, unit 0 is transmitted during contact (α_1, β_1), during every contact $(\epsilon_i, \beta_{i+1})$ with $i \in \{1, 2, \ldots, p-1\}$ and then during contact (ϵ_p, ω_1). For each $i \in \{1, \ldots, p\}$, datum unit 0 is also transmitted through contacts:

$$\begin{cases} [(\beta_i, l_i^1), (l_i^1, l_i^2), \ldots, (l_i^{2a_i-1}, l_i^{2a_i}), (l_i^{2a_i}, \epsilon_i)] \textbf{ if } x_i = \mathit{true}\,; \\ \textbf{or through } [(\beta_i, \bar{l}_i^1), (\bar{l}_i^1, \bar{l}_i^2), \ldots, (\bar{l}_i^{2b_i-1}, \bar{l}_i^{2b_i}), (\bar{l}_i^{2b_i}, \epsilon_i)] \text{ otherwise.} \end{cases}$$

Note that unit 0 is transmitted through contact (β_i, ϵ_i) if x_i is *true* and $a_i = 0$ or if x_i is *false* and $b_i = 0$. Since datum unit 0 has been transmitted from α_1 to ω_1, ω_1 possesses all datum units at the end of sequence σ.

In each clause c_j, $j \in \{1, 2, \ldots, s\}$, we choose the first *true* literal.

- Let us suppose that this literal is x_i, $i \in \{1, 2, \ldots, p\}$, and that it is the y^{th} occurrence of literal x_i in set C. Datum unit j is transferred during contacts (α_2, l_i^{2y-1}), (l_i^{2y-1}, l_i^{2y}), (l_i^{2y}, C_j), and (C_j, ω_2).
- Let us now suppose that this literal is $\bar{x}_i$, $i \in \{1, \ldots, p\}$, and that it is the y^{th} occurrence of literal $\bar{x}_i$ in C. Then datum unit j is transmitted during contacts $(\alpha_2, \bar{l}_i^{2y-1})$, $(\bar{l}_i^{2y-1}, \bar{l}_i^{2y})$, $(\bar{l}_i^{2y}, C_j)$, and finally (C_j, ω_2).

In both cases, these contacts cannot have been used to transmit unit 0 since the contrary would mean that both literals x_i and $\bar{x}_i$ are *true*.

All other transfers can be set to $\emptyset$. Each unit $j \in \mathcal{D}$ has been transmitted from α_2 to ω_2. Thus the transfer plan is such that $\forall i \in \mathcal{R}$, $O_i^m = \mathcal{D}$. □

2.2 Polynomial-time cases

In this section, we first show that the data transfer problem can be solved in polynomial time if $u = 1$. This specific case is also called the *one-datum-unit problem*. Then we show that the data transfer problem can be polynomially solved if set $\mathcal{R} = \{\omega\}$ is a singleton. This specific case is termed the *delivery problem*, and is to find a valid transfer plan ϕ minimizing $\lambda_\omega(\phi)$. Finally we show the case where u and $|\mathcal{R}|$ are both constant is also polynomial.

2.2.1 The one-datum-unit problem ($u = 1$)

In this problem, $\mathcal{D} = \{1\}$ (there is only one datum unit). If there is at least one transfer plan leading to the dissemination of the whole datum, then there is at least one optimal transfer plan where, for each contact $\sigma_c \in \sigma$, we have $\phi(c) = \{1\}$ if $O_{r_c}^{c-1} = \emptyset$ and $O_{s_c}^{c-1} = \{1\}$ (if node r_c does not possess the unit, while node s_c does). Indeed, there is no advantage to be gained in delaying the dissemination of the only datum unit. Thus it is sufficient to go through the sequence of contacts, and to enforce that every node obtains the datum as soon as possible. This process is summarized in Algorithm 2.1.

In short $o[i]$ represents the current state of node $i \in \mathcal{N}$ – *i.e.* $o[i]$ is equal to $\{1\}$ if node i possesses the datum, or to $\emptyset$ otherwise – while N_d indicates the current number of recipient nodes which possess the datum. In the first loop, every element $o[i]$ ($\forall i \in \mathcal{N}$) is therefore set to $\mathcal{O}_i$, while N_d is initialized with the number of recipients storing the datum at the outset. In the second loop, contacts are considered in the order of the sequence. During a contact $\sigma_c \in \sigma$ (from $c = 1$ to m), the datum is transmitted to node r_c if needed and if possible, *i.e.* if $o[r_c] = \emptyset$ and $o[s_c] = \{1\}$. Of course $o[r_c]$, $\phi(c)$, and N_d are updated accordingly. This loop terminates if all recipient nodes possess the datum (if $N_d = |\mathcal{R}|$), or if the end of sequence σ has been reached. If it stops while $N_d < |\mathcal{R}|$, then it means the instance is not feasible. Finally the third loop sets all remaining transfers (if any) to $\emptyset$.

It is worth nothing that Algorithm 2.1 runs in $O(\max(n, m))$ time.

Theorem 2.3

The dissemination problem can be solved in $O(\max(n, m))$ time if $u = 1$ (if there is only one datum unit).

Algorithm 2.1 – Solving the *one-datum-unit problem*

Require: An instance of the one-datum-unit problem ;

```
 1:
 2: # only the sources possess the datum unit from the outset.
 3: N_d ← 0 ; λ(φ) ← ∞ ;
 4: for i : 1 → n do
 5:     o[c] ← O_c ;
 6:     if O_c = {1} and i ∈ R then N_d ← N_d + 1 ;
 7:
 8: c ← 1 ;
 9: while c ≤ m and N_d < |R| do
10:
11:     # the recipient node obtains the datum unit if possible.
12:     if o[r_c] = ∅ and o[s_c] = {1} then
13:         φ(c) ← {1} ; o[r_c] ← {1} ;
14:         if r_c ∈ R then N_d ← N_d + 1 ;
15:
16:     # the transfer is set to ∅ otherwise.
17:     else φ(c) = ∅ ;
18:
19:     # the procedure stops if every recipient possesses the datum.
20:     if N_d = |R| then λ(φ) ← c ;
21:     c ← c + 1 ;
22:
23: end while
24:
25: if N_d < |R| then
26:     return "This instance is not feasible." ;
27:
28: # the remaining transfers are set to ∅.
29: while c ≤ m do
30:     φ(c) = ∅ ; c ← c + 1 ;
31:
32: return φ and λ(φ) ;
33:
```

2.2.2 The delivery problem ($|\mathcal{R}| = 1$)

Recall that this problem is similar to the dissemination problem, except that only one specified node $\omega \in \mathcal{N}$ needs to obtain all the datum units.

Obviously, the delivery problem can be solved in $O(\max(n, m))$ if $u = 1$ (with Algorithm 2.1). We will now show that the general case is polynomial as well. To this end, we show that solving the delivery problem is equivalent to solving $\max(u, m)$ separate maximum flow problems.

Theorem 2.4

The dissemination problem can be solved in $O((nu+m)\max(u, m))$ time when $|\mathcal{R}| = 1$ (when there is only one recipient).

Polynomial-time reduction

Let us consider an instance of the delivery problem. Let $\mu_i = |\{\sigma_c \in \sigma \text{ such that } r_c = i\}|$ denote the number of contacts where node $i \in \mathcal{N}$ is the receiver (note that even a source node may be the receiver of a contact).

We consider a transportation network $G = (V, A, cap)$, built as follows:

- first we add a source vertex $src \in V$;
- with each datum unit $k \in \mathcal{D}$, we associate a vertex $d_k \in V$;
- for each node $i \in \mathcal{N}$, we add a set of vertices $\{i^0, \ldots, i^{\mu_i}\} \subset V$;
- $\forall k \in \mathcal{D}$, we add an arc $(src, d_k) \in A$ with capacity $cap(src, d_k) = 1$;
- $\forall i \in \mathcal{N}$ and $\forall k \in \mathcal{D}$ – if datum unit $k \in \mathcal{O}_i$ – we add an arc $(d_k, i^0) \in A$ with capacity $cap(d_k, i^0) = 1$;
- $\forall i \in \mathcal{N}$, $\forall x \in \{1, \ldots, \mu_i\}$, we add (i^{x-1}, i^x) with $cap(i^{x-1}, i^x) = u$;
- for each contact $\sigma_c = (i, j) \in \sigma$ – assuming that node i is the receiver of x contacts before σ_c – and that σ_c is the y^{th} contact where node j is the receiver – we add an arc $(i^x, j^y) \in A$ of capacity $cap(c^x, j^y) = 1$;
- finally we add an arc $(\omega^{\mu_\omega}, src) \in A$ with $cap(\omega^{\mu_\omega}, src) = u$.

For example, let us consider the following instance of the delivery problem:

- $n = 4$ and $\mathcal{N} = \{1, 2, 3, 4\}$; $u = 2$ and $\mathcal{D} = \{1, 2\}$;
- $\mathcal{O}_1 = \{1, 2\}$, $\mathcal{O}_2 = \{2\}$ and $\mathcal{O}_3 = O_4 = \emptyset$; $\omega = 4$, *i.e.* $\mathcal{R} = \{4\}$;
- $m = 5$ and $\sigma = [(1, 3), (2, 4), (2, 3), (2, 4), (3, 4)]$.

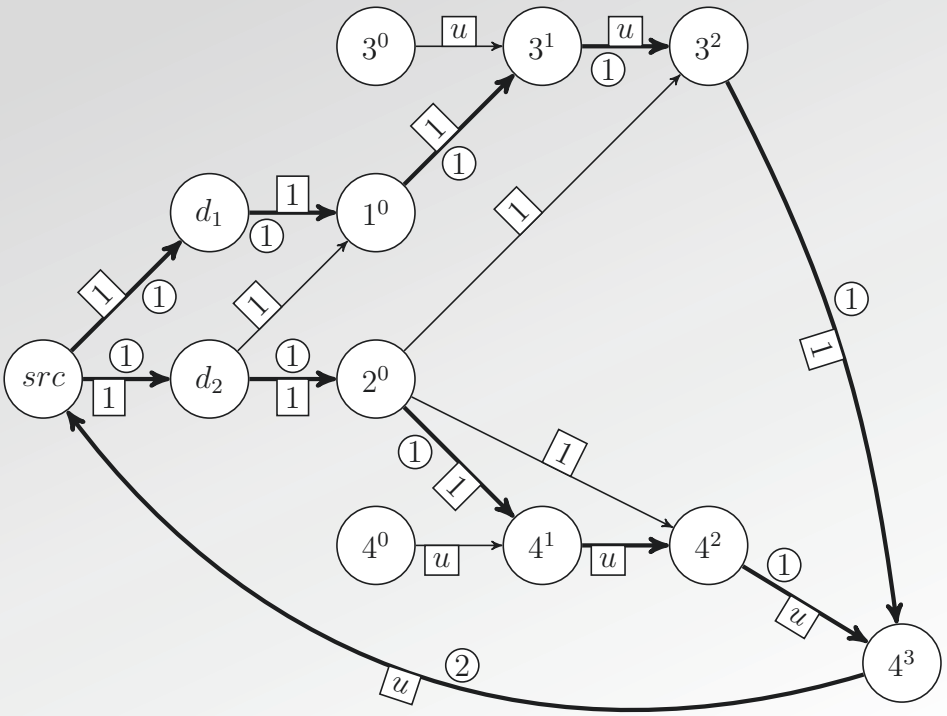


Figure 2.5 – The transportation network associated with the given instance of the delivery problem.

The associated transportation network is depicted in Figure 2.5. The arcs of capacity u model the capacity for each node to store data, whereas the arcs of unitary capacity represent possible transfers (first to initialize the sources, and then to model the contacts).

It is worth nothing that the maximum flow on arc $(\omega^{\mu_\omega}, src)$ cannot exceed its capacity u. The resulting transportation network contains $1 + u + n + m$ vertices and $u + \sum_{i \in \mathcal{N}} |\mathcal{O}_i| + 2m + 1 \leq (n+1)u + 2m + 1$ arcs.

We show that solving the delivery problem (decision version) is equivalent to searching for a flow of value u in the associated flow network.

Proposition 2.1

Given any instance of the delivery problem, there exists a valid transfer plan such that the recipient node obtains the whole datum if and only if there exists a flow of value u in the associated flow network.

From a flow of value u to a valid transfer plan

Let us consider a flow $f : A \mapsto \mathcal{D}$ (in the associated flow network) satisfying capacity and conservation constraints, and such that the flow entering vertex src is equal to u. A valid transfer plan ϕ can be obtained as follows.

First we consider the amount of flow from each vertex d_k ($k \in \{1, \ldots, u\}$) to each vertex i^0 ($i \in \{1, 2, \ldots, n\}$). If $f(d_k, i^0) = 1$ then i^0 is the vertex that is considered to be the owner of datum unit k. Each datum unit is considered to be owned by at most one vertex, since the flow entering vertex d_k cannot exceed 1. From now on, every unit of flow from one vertex to another in the transportation network is considered as a change of owner for a unit.

We consider the contacts in the order of the sequence. For each contact $\sigma_c = (i, j) \in \sigma$, let $(i^s, j^t) \in A$ be the corresponding arc in the transportation network – *i.e.* $i^s \in V$ and $j^t \in V$ are the nodes such that i has already been the receiver of s contacts before contact σ_c, and such that σ_c is the t^{th} ($t > 0$) contact in which j is the receiver.

- If $f(j^{t-1}, j^t) = p$, then j^t becomes the owner of p datum units that j^{t-1} previously owned. Note that flow conservation constraints ensure that j^{t-1} was the owner of at least p datum units. If j^{t-1} was the owner of more than p units, the p units whose owner is changing can be chosen indifferently, *e.g.* the units with the smallest indices.
- If $f(i^s, j^t) = 1$, then j^t becomes the owner of one datum unit k that i^s previously owned (arbitrarily chosen), and we set $\phi(c) = \{k\}$.
- Finally, if $f(i^s, j^t) = 0$, then we set $\phi(c) = \emptyset$.

At the end of this step, note that every datum unit is still considered to be owned by only one vertex. The process is iteratively applied to each contact in sequence σ. Thereafter we look at the amount of flow through $(\omega^{\mu_\omega}, src)$. If $f(\omega^{\mu_\omega}, src) = u$, then we can conclude that vertex ω^{μ_ω} is the owner of all the datum units (and thus that the transfer plan is such that node ω obtains all these datum units during the sequence of contacts).

For example, in Figure 2.5 (*cf.* instance page 29), the flow corresponds to the transfer plan ϕ such that $\phi(1) = \phi(5) = \{1\}$ and $\phi(2) = \phi(4) = \{2\}$.

From a valid transfer plan to a flow of value u

Conversely, let us now consider a valid transfer plan ϕ, solution of the delivery problem. From the sequence of contacts, and from this transfer plan, we can compute the states O_i^c of each node $i \in \mathcal{N}$ after the different contacts $\sigma_c \in \sigma$. Then we can build the flow function $f : A \mapsto \mathcal{D}$ as follows.

Initially the flow is null – *i.e.* $\forall a \in A$, $f(a) = 0$. Subsequently we consider vertex ω^{μ_ω} in the transportation network, and we assume that this vertex is the owner of all datum units. The procedure consists of searching iteratively for the different owners of each datum unit $k \in \mathcal{D}$, until a vertex i^0 ($i \in \mathcal{N}$) becomes its owner.

To this end we consider a datum unit k that is initially owned by vertex $i^s = \omega^{\mu_\omega}$. We will assume $s > 0$ (if this were not the case, the process would terminate immediately, since ω would never be the receiver in a contact and would therefore have to own the whole datum at the outset). Let σ_x be the s^{th} contact whose i is the receiver, and thus let O_i^x be the set of datum units possessed by i after contact σ_x.

- If $k \in O_i^{x-1}$, then vertex i^{s-1} becomes the new owner of datum unit k, and $f(i^{s-1}, i^s)$ is incremented by one unit.
- Conversely, if $k \notin O_i^{x-1}$, then node i has received datum unit k during contact σ_x. Let j be the sender in this contact, and let $t \in \{1, \ldots, \mu_j\}$ be the number of contacts occurring before σ_c in which node j was the receiver. j^t becomes the new owner of k, and we set $f(j^t, i^s) = 1$.

Note that the flow values obtained always respect the capacity constraints of the flow network. Besides – by setting $i^s = i^{s-1}$ or $i^s = j^{t-1}$ according to the case – the process can be repeated until unit k is owned by a vertex i^0.

At the end of each iteration there is only one owner of datum unit k, and the change of owner results in an increase in the flow on the arc linking the new owner to the previous owner (ensuring, this way, flow conservation).

This operation is repeated for all of the units. At the end of this process we therefore know which vertex i^0 is the owner of each datum unit k, and we set $f(d_k, i^0) = 1$ accordingly. For each $d_k \in \mathcal{D}$, we set $f(src, d_k) = 1$. Finally we set $f(\omega^{\mu_\omega}, src) = u$. □

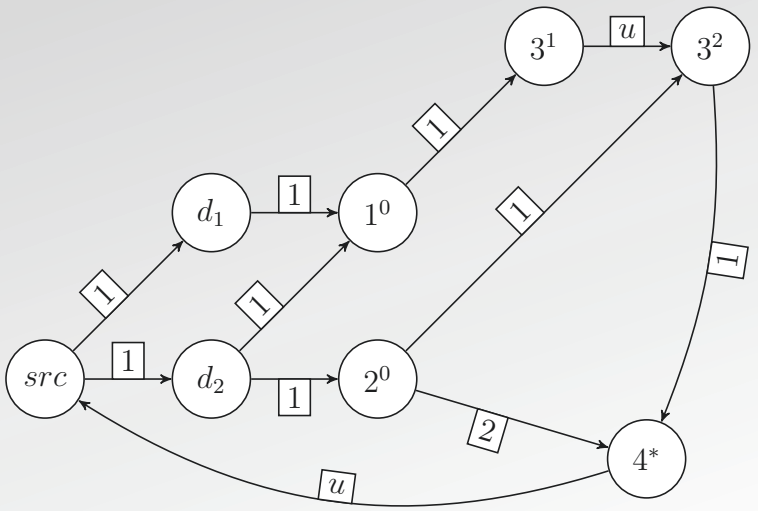


Figure 2.6 –The simplified transportation network associated with the instance of the delivery problem given on page 29.

In practice the flow network can be simplified by iteratively removing the vertices that have no successor or no predecessor, and through which the flow can only be null. Moreover the vertices representing the states of node ω can be merged into a single node (the resulting vertex ω^* is the destination of all arcs entering a vertex ω^x, and any arc going out of such a vertex is removed, apart from the one to src) – *cf.* Figure 2.6.

Optimizing the delivery length $\lambda_\omega(\phi)$

Node ω needs at least $u - |\mathcal{O}_\omega|$ contacts to receive the whole datum. Thus, one way of finding a valid transfer plan ϕ that minimizes the delivery length $\lambda_\omega(\phi)$ is to build the flow network associated with the shortest subsequence of σ involving $u-|\mathcal{O}_\omega|$ contacts whose node ω is the receiver. If $f(\omega^{\mu_\omega}, src) = u$, then the transfer plan ϕ associated with f is optimal. Otherwise, we complete the current subsequence of σ in such a way that it involves $u - f(\omega^{\mu_\omega}, src)$ more contacts with node ω. So a new maximum flow can be computed (note that in practice the previous flow can be easily transformed to obtain a new starting flow). The procedure is repeated until $f(\omega^{\mu_\omega}, src) = u$, or until the

current sequence equals σ. If the procedure stops while $f(\omega^{\mu_\omega}, src) < u$, then it means that the instance is infeasible. Using Ford-Fulkerson algorithm [24] or Edmonds and Karp algorithm [18] to compute the maximum flow of each transportation network that is considered, the overall procedure can run in $O((nu+m)\max(u,m))$ time, given that each augmenting path can be found in $O(|A|)$ time, and that we need to find at most $\max(u,m)$ such paths.

2.2.3 Upper bounded parameters

In this section we look at an instance of the dissemination problem in which both u and $|\mathcal{R}|$ are assumed to be upper bounded by given positive constant numbers K_1 and K_2, with $K_1 \geq u \geq 2$ and $K_2 \geq |\mathcal{R}| \geq 2$.

We might be tempted to use the same kind of technique as in the previous section, but as $|\mathcal{R}| \geq 2$ this is no longer possible. In the case where $|\mathcal{R}| = 1$, this worked because exactly u units were flowing through the transportation network, and each unit of flow was interpreted as the ownership of a datum unit. Note that the datum units did not need to be identified. Now, however, the datum units might need to be duplicated (in order to be transmitted to several nodes), which is prohibited by conservation constraints. In addition, taking one unit of flow per datum unit per recipient is not possible, because the units of flow are not identified (this could lead to transfer plans in which a recipient node gets the same datum unit twice).

Nonetheless, by taking one unit of flow per datum unit and per recipient, we can build Algorithm 2.2.

Theorem 2.5

The dissemination problem can be solved in polynomial time when the number of units and the number of recipients are upper bounded.

Throughout the procedure, a set of states *States* is managed, and exactly $|\mathcal{R}|$ copies of each datum unit are considered (because there is never a need to duplicate a unit more than $|\mathcal{R}|$ times). Every state $S \in States$ is defined by a bidimensional array $S.o$, where $S.o[k][z]$ designates the node owning the z^{th} ($z \in \{1, \ldots, |\mathcal{R}|\}$) copy of unit $k \in \mathcal{D}$, and by the transfer plan $S.\phi$ from which this state has arisen.

The procedure is started with only one state S (*cf.* lines 2–7) in which an imaginary node 0 owns all copies of all datum units, and where $S.\phi(c) = \emptyset$, $\forall c \in \{1, 2, \ldots, m\}$. The ownership of each copy z of each datum unit is then

Algorithm 2.2 – Solving the *dissemination problem*

Require: An instance of the dissemination problem ;

```
 1:
 2: # computation of the initial state.
 3: S ← newState() ;
 4: for k ← 1 to u do
 5:     for z = 1 to |R| do S.o[k][z] ← 0 ;
 6: for c ← 1 to m do S.φ(c) ← ∅ ;
 7: States ← {S} ;
 8:
 9: # datum unit copies are distributed to initial owners.
10: for k ← 1 to u do
11:     for z = 1 to |R| do
12:         # the z^th copy of datum unit k is distributed.
13:         States' ← ∅ ;
14:         for all S ∈ States do
15:             for all i ∈ N | k ∈ O_i do
16:                 S' ← copy(S) ; S'.o[k][z] ← i ;
17:                 add(States', S') ;
18:             end for
19:         add(States, States') ;
20:
21: # the ownership of unit copies is transferred through contacts.
22: for c ← 1 to m do
23:     States' ← ∅ ;
24:     for all S ∈ States do
25:         for k ← 1 to u do
26:             for z = 1 to |R| do
27:                 # the ownership of the z^th copy of datum unit k
28:                 # is transferred to r_c if possible – i.e. if the current
29:                 # owner of this copy is s_c.
30:                 if S.o[k][z] = s_c then
31:                     S' ← copy(S) ; S'.o[k][z] ← r_c ; S'.φ(c) ← {k} ;
32:                     add(States', S') ;
33:                     if isSolution(S') then return S'.φ ;
34:                 end if
35:     end for
36:     add(States, States') ;
37:
```

distributed (*cf.* lines 9–19) among source nodes i with $k \in \mathcal{O}_i$. At the end of this phase, a node i can own several copies of the same datum unit (so that this datum unit can be transferred to several other nodes).

Next, the ownership of datum unit copies is transferred through contacts (*cf.* lines 21–36). At the end of each iteration $c \in \{1, \ldots, m\}$, $States$ contains one state for each possible situation that can arise from contacts $[\sigma_1, \ldots, \sigma_c]$. For this purpose a new set of states $States'$ is computed from the set $States$ obtained at the end of the previous iteration. For each $S \in States$ and each copy z of each datum unit k, if $S.o[k][z] = s_c$ (if a transfer is possible), a new state $S' \in States'$ is created by copying S, and by transferring the ownership of copy z of unit k to node r_c (see $S'.o[k][z] \leftarrow r_c$). It corresponds to transfer $\phi(c) = \{k\}$, *i.e.* $S'.\phi(c) \leftarrow \{k\}$. We here assume that S' is not considered if there exists another state with exactly the same ownership matrix (although not necessarily the same transfer plan). Note that instruction $isSolution(S')$ checks whether all recipient have received the whole datum (this can be done in polynomial time). If this is the case, the algorithm immediately stops and returns the computed transfer plan.

Time-complexity – Since the ownership of the z^{th} copy, $z \in \{1, \ldots, |\mathcal{R}|\}$, of datum unit $k \in \mathcal{D} = \{1, 2, \ldots, u\}$ can take $n+1$ values, there are at most $(n+1)^{|\mathcal{R}|u}$ possible states. Therefore the overall algorithm runs in $O(mn^{|\mathcal{R}|u})$ time – which is polynomial in m and n, since $|\mathcal{R}|$ and u are upper bounded by two positive constant numbers K_1 and K_2 (of course, this algorithm can only be used for small values of $|\mathcal{R}|$ and u).

2.3 Additional results

In this section, we discuss complementary results arising from what has been said so far. In particular, it will be shown that knowing whether there exist k mutually arc-disjoint branchings in an evolving graph – or whether there exist k mutually arc-disjoint Steiner trees in a directed graph without circuit – are strongly NP-Complete

2.3.1 Arc-disjoint branchings in an evolving graph

To address this problem, we first remind its statement and classical results for the more conventional case of directed graphs. Thereafter we show that these results no longer hold for evolving graphs, and establish that the problem is actually NP-Complete in this case (we reduce it to Dt).

Usual graphs

Let $G = (V, A)$ be a directed multigraph.

- Given $t \in V$, a *branching* B rooted at t is a subgraph of G such that, for every vertex $v \in V$, there is exactly one path in B from t to v.
- Given $X \subset V$, $\delta_G(X)$ denotes the set of arcs $(i, j) \in A$ such that $i \in X$ and $j \in V \backslash X$ – *i.e.* the arcs "going out" of X. Such a set is commonly termed a *cut* in graph theory.

Theorem 2.6 – Edmonds 1972 [17]

There exist k mutually arc-disjoint branchings rooted at t if and only if for any $X \subset V$ such that $t \in X$ and $X \neq V$, we have $|\delta_G(X)| \geq k$.

Let then

$$c_G(t) = \min_{\{X | t \in X, X \neq V\}} |\delta_G(X)|$$

be the maximum number of arc-disjoint branchings rooted at t in G. In fact, $c_G(t)$ can be regarded as the "outwards connectivity" of t – *i.e.* the minimum number of arcs that would need to be removed to make at least one vertex unreachable from t [40]. $c_G(t)$ is actually the maximum number of mutually arc-disjoint paths from t to any other vertex in V. This well-known property is used by a number of polynomial algorithms that have been developed for finding k mutually arc-disjoint branchings in a multigraph, such as Shiloach's [40] in $O(k^2|V| \times (|V| + |A|))$.

Evolving graphs

As mentioned in Chapter 1, an *evolving graph* is a theoretical graph model, first introduced by Ferreira [21], and designed to capture main characteristics of intermittently connected networks. It is a directed multigraph $G(V, A, \tau)$ whose vertices represent nodes, and whose arcs represent links between these nodes. $\tau : A \mapsto I$ (I being the set of intervals which can be built over a given time horizon) indicates the interval on which links can be used. Thus an arc can represent a contact between two nodes. In its simplest version, τ is such that $\tau : A \mapsto \mathbb{N}$ – *i.e.* the interval of time during which every link is active is reduced to a singleton. In this case, a path $[a_0, a_1, a_2, ..., a_z]$ ($a_j \in A$) is such that $\forall j \in \{0, \ldots, z-1\}$, we have $\tau(a_j) \leq \tau(a_{j+1})$. From now on such a path is termed a *journey*.

Surprisingly, unlike usual graphs, knowing whether there exists a strong connected component of a given size in an evolving graph (when considering journeys instead of usual paths) is NP-Complete [7]. However, the minimum spanning trees (rooted at each vertex) in a strongly connected evolving graph can be polynomially computed. Some other algorithms in the literature can be polynomially generalized in evolving graphs. These include the search for foremost journeys (*i.e.* the journeys with the earliest arrival dates), shortest journeys (using the smallest number of arcs) and fastest journeys (with the smallest difference between departure and arrival times) from a source vertex to all other vertices [45].

We now turn to the problem of knowing whether there exist k mutually arc-disjoint branchings in an evolving graph. To our knowledge, this problem has never been investigated.

Note first that Theorem 2.6 no longer holds for evolving graphs. This can be shown with the following counter-example (*cf.* Figure 2.7):

- $G = (V, A, \tau)$; $V = \{t, a, b, c, d, e, f\}$; $A = \{a_1, a_2, \ldots, a_{12}\}$;
- $a_1 = (t, a), a_2 = (a, c), a_3 = (a, e), a_4 = (t, b), a_5 = (b, c), a_6 = (c, d)$, $a_7 = (d, e), a_8 = (d, f), a_9 = (b, f), a_{10} = (t, a), a_{11} = (t, b), a_{12} = (c, d)$;
- $\tau : V \mapsto \mathbb{N}$; $\forall i \in \{1, 2, \ldots, 12\}, \tau(a_i) = i$;

Proof. We consider root $t \in V$. There are at least two arc-disjoint journeys from t to any other vertex $v \in V \backslash \{t\}$, *i.e.* $c_G(t) \geq 2$. Now let us show there does not exist two mutually arc-disjoint branchings rooted at t.

First, let us assume there are two arc-disjoint branchings B_1 and B_2 rooted at t. Each branching has exactly 6 arcs. Thus, $\{B_1, B_2\}$ has to be a partition of the set A of arcs. a_{10} and a_{11} cannot be in the same branching, otherwise vertices c, d, e and f could not be reached. Let us assume that $a_{10} \in B_1$ and $a_{11} \in B_2$. It means that arc $a_4 \in B_1$, and $a_1 \in B_2$. The only possible course of action is to add arc a_5 to B_1, and to add arc a_2 to B_2. Then, we have to put a_6 in one branching, and a_{12} in the other one. If $a_6 \in B_1$ and $a_{12} \in B_2$, then a_7 and a_3 have respectively to be in B_1 and B_2 (because a_3 cannot be used after a_{10}). Thus, both arcs a_9 and a_8 can be added in B_1, but none can be added to B_2 because a_9 cannot be used after a_{11}, and a_8 cannot be used after a_{12}. The same contradiction arises with $a_{10} \in B_2$ and $a_{11} \in B_1$, or with $a_6 \in B_2$ and $a_{12} \in B_1$. □

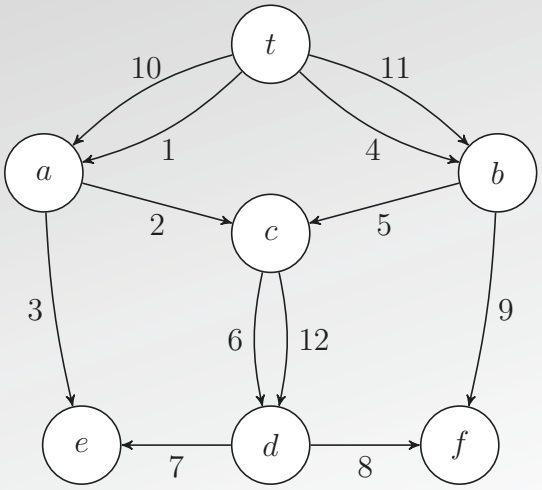


Figure 2.7 – This counter-example shows Theorem 2.6 no longer holds for evolving graphs.

In fact knowing whether there exist k mutually arc-disjoint branchings in an evolving graph is NP-Complete. To prove this, let us consider the decision version of this problem, referred to below as Adbeg.

Problem 2.3 – arc-disjoint branchings in evolving graphs

Given an evolving graph $G = (V, A, \tau)$ · a root $t \in V$ · an integer k with $k \leq |A|/(|V| - 1)$ – **are there** k **mutually arc-disjoint branchings rooted at** t **in** G ?

Theorem 2.7

Adbeg (Problem 2.3) is NP-Complete.

Polynomial-time reduction

Let us show that DT is reducible to ADBEG in polynomial time. To this end, we consider an instance of DT (*cf.* Problem 2.1, page 15), and we build the following instance of ADBEG.

Vertices – With each node $i \in \{1, \ldots, n\}$ in DT, we associate a vertex $i \in V$ in ADBEG. Then, with each datum unit $k \in \mathcal{D}$ in DT, we associate a vertex $d_k \in V$ in ADBEG. Finally, we add a vertex $t \in V$.

Arcs – For each datum unit $k \in \mathcal{D}$ of DT, we add one arc $(t, d_k) \in A$ with $\tau(t, d_k) = 0$ in ADBEG. Then, for each node $i \in \{1, 2, \ldots, n\}$ and each datum unit $k \in \mathcal{D}$ in DT – *iif.* $k \in \mathcal{O}_i$ – we add an arc $(d_k, i) \in A$ with $\tau(d_k, i) = 0$ in ADBEG. With each contact $\sigma_c \in \sigma$ in DT, we associate an arc $(s_c, r_c) \in A$ with $\tau(s_c, r_c) = c$ in ADBEG. Finally, for each node $i \in \overline{\mathcal{R}} = \mathcal{N} \backslash \mathcal{R}$, we add u arcs (t, i) with $\tau(t, i) = m + 1$.

An example is given in Figure 2.8. The DT instance corresponds to the one of Figure 1.1 (page 11), with the addition of a non-recipient node.

Proof of equivalence

Suppose that there exist a set $\{B_1, B_2, \ldots, B_u\}$ of u mutually arc-disjoint branchings in $G = (V, A)$. The only way to reach a vertex $i \in \mathcal{R}$ from t is to first use an arc (t, d_k) with $\tau(t, d_k) = 0$, and then to follow some arcs a_j with $1 \leq \tau(a_j) \leq m$. Each branching is then associated with exactly one arc (t, d_k) (there are only u such arcs). Let B_k denote the branching associated with arc (t, d_k).

We can build a valid transfer plan ϕ for DT as follows.

- We initially set $\phi(c) = \emptyset$ for $c \in \{1, 2, \ldots, m\}$.
- Next, for each branching B_k, $k \in \{1, 2, \ldots, u\}$, and for each arc $a \in B_k$ with $1 \leq \tau(a) \leq m$, we set $\phi(\tau(a)) = \{k\}$. Note that since the different branchings are mutually arc-disjoint, and since there is exactly one arc $a \in A$ associated with a contact in σ, we never try to set several values to the same transfer $\phi(c)$.

In each branching B_k, and for each $i \in \mathcal{R}$, there is a journey $[a_0, a_2, \ldots, a_x]$ such that $a_0 = (t, d_k)$, $a_1 = (d_k, z)$ with $k \in \mathcal{O}_z$, and for all $j \in \{3, 4, \ldots, x\}$, $1 \leq \tau(a_j) \leq m$ and $\tau(a_{j-1}) < \tau(a_j)$. Consequently it corresponds to a list of contacts which enable datum unit k to be transmitted from source node z to node i. Thus, at the end of σ, all $i \in \mathcal{R}$ are such that $O_i^m = \mathcal{D}$.

(a) *an instance of the data transfer problem*

$$\begin{cases} \mathcal{N} = \{1, 2, \ldots, 7\}; \mathcal{R} = \mathcal{N} \backslash \{7\}; \mathcal{D} = \{1, 2\}; \\ \mathcal{O}_1 = \{1, 2\}; \mathcal{O}_2 = \{1\}; \mathcal{O}_3 = \mathcal{O}_4 = \mathcal{O}_5 = \mathcal{O}_6 = \mathcal{O}_7 = \emptyset\,; \\ \sigma = [(1,6), (6,1), (6,5), (1,3), (3,5), \ldots, (5,6), (4,7)] \end{cases}$$

(b) *the associated* ADBEG *instance*

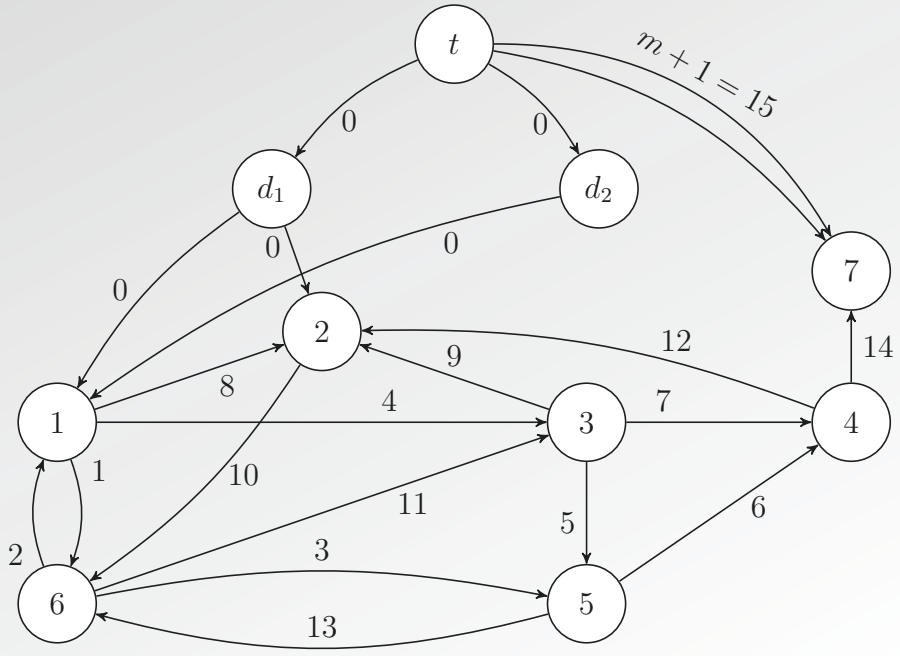


Figure 2.8 – An instance of the data transfer problem (DT) and the corresponding ADBEG instance.

Now, suppose there is a valid transfer plan ϕ. For each datum unit $k \in \mathcal{D}$, we first set $B_k = (V, A_{B_k} = \{(t, d_k)\})$. Next, for each node $i \in \mathcal{N}$ such that $k \in \mathcal{O}_i$, we add arc (d_k, i) in A_{B_k}. Then, for each $c \in \{x \mid \phi(x) = k\}$, we add arc (s_c, r_c) in A_{B_k}. Note that for any list of transfers enabling datum unit k to be transmitted from a node $z \in \mathcal{N}$ with $k \in \mathcal{O}_z$ to node $i \in \mathcal{N}$, there is a corresponding journey from t to i in B_k. In particular, it holds for any node $i \in \mathcal{R}$. Moreover, the nodes $i \in V \backslash \mathcal{R}$ for which there is no path from t to i cannot be connected to another node in B_k. For such nodes, we add an arc $(t, i) \in A_{B_k}$ with $\tau(t, i) = m + 1$. This is always possible since there exist u arcs of this kind. Thus subgraph B_k is a branching. It must finally be noted that $B_1, B_2, \ldots, B_k$ are mutually arc-disjoint by construction. □

2.3.2 Arc-disjoint Steiner trees in a digraph

The problem of packing arc capacitated directed Steiner trees is a well known problem [15]. It is stated as follows.

Problem 2.4 – Steiner tree packing problem

Let us consider a directed graph $G = (V, A, cap)$ with positive capacities $(cap : A \mapsto \mathbb{N}^+)$, a set $T \subseteq V$ of terminals, and a root $t \in T$. A *directed Steiner tree* rooted at t is a directed subgraph of G (a tree) that contains a path from t to every terminal $v \in T$. The *Steiner tree packing problem* is to find the **maximum number of Steiner trees rooted at** t, such that for every arc $a \in A$, the total number of Steiner trees containing a is at most $cap(a)$.

Cheriyan and Salavatipour [15] established that the problem of knowing whether there are two arc-disjoint Steiner trees in a graph with unit capacities and only three terminals is NP-Complete. The proof is based on the following reduction — given two pairs of vertices (x_1, y_1) and $(x_2, y_2) \in V^2$, are there two arc-disjoint paths, one from x_1 to y_1, and the other from x_2 to y_2 ?

In fact, it can be shown that this problem is NP-Complete even if there are three terminals and *the graph is without circuit*. Indeed, as discussed in Section 2.1.2, the data transfer problem is NP-Complete in the strong sense, even if $|\mathcal{R}| = 2$ (*i.e.* DT2R).

In addition, with an instance of DT2R, we can associate a directed graph $G = (V, A, cap)$ similar to the one we built for the delivery problem (Section 2.2.2 and Figure 2.5).

- first we add a source vertex $src \in V$;
- with each datum unit $k \in \mathcal{D}$, we associate a vertex $d_k \in V$;
- for each node $i \in \mathcal{N}$, we add a set of vertices $\{i^0, \ldots, i^{\mu_i}\} \subset V$;
- $\forall k \in \mathcal{D}$, we add an arc $(src, d_k) \in A$ with capacity $cap(src, d_k) = 1$;
- $\forall i \in \mathcal{N}$ and $\forall k \in \mathcal{D}$ – if datum unit $k \in \mathcal{O}_i$ – we add an arc $(d_k, i^0) \in A$ with capacity $cap(d_k, i^0) = 1$;
- $\forall i \in \mathcal{N}$, $\forall x \in \{1, \ldots, \mu_i\}$, we add (i^{x-1}, i^x) with $cap(i^{x-1}, i^x) = u$;
- for each contact $\sigma_c = (i, j) \in \sigma$ – assuming that node i is the receiver of x contacts before σ_c – and that σ_c is the y^{th} contact where node j is the receiver – we add an arc $(i^x, j^y) \in A$ of capacity $cap(c^x, j^y) = 1$;
- note that we no longer consider the "return" arc $(\omega^{\mu_\omega}, src)$.

Thus, knowing whether there is a valid transfer plan $O_i^m = \mathcal{D}$ $(\forall i \in \mathcal{R})$ with $|\mathcal{R}| = 2$ becomes equivalent to knowing whether there exist two arc-disjoint Steiner trees rooted at src with terminals $T = \{src\} \cup \mathcal{R}$ in the corresponding transportation network. Here it should be noted that this network is always without circuit (by construction).

2.4 Conclusion

In this chapter, we highlighted several polynomial cases for the dissemination problem, *i.e.* the latter is solvable in polynomial time when there is only one datum unit, or only one recipient, or when both the number of datum units and the number of recipients are upper bounded by a given constant.

However, we also proved the general case is strongly NP-Hard. Therefore, non-polynomial algorithms can be investigated in order to solve the problem in the general case, like branch-and-cut/branch-and-bound procedures.

3 Dominance rules, preprocessings, and integer linear programming

Now that the NP-Completeness of the dissemination problem has been established, it seems interesting to propose non-polynomial algorithms (such as branch-and-bound procedures) to solve it. With this aim in view, in this chapter we are going to propose a number of *dominance rules* for this problem. These yield conditions on which a subset of the search space considered to solve the problem can be *ignored* (*cf.* Section 3.1). Thereafter, we will propose algorithms which make use of these rules to deduce additional constraints, and this way, eliminate *dominated solutions* – *i.e.* solutions which can be ignored according to the dominance rules. The algorithms rely on a graph model, called the *transfer graph* (*cf.* Section 3.2), aiming at capturing knowledge about admissible transfer plans. Finally, all of this will be tested and incorporated into *preprocessing procedures* (*cf.* Section 3.3). These will aim to strengthen an *integer linear program* modelling the problem (*cf.* Section 3.4).

Contents

3.1 Dominance rules 46

3.2 Transfer graph 51

3.2.1 About the transfer graph 52

3.2.2 Transfer graph and subsets of transfer plans 54

3.2.3 Additional graph properties and complex subsets of transfer plans 57

3.2.4 Using the transfer graph 59

3.3 Deductive elements 59

3.3.1 Finding non-minimal transfer plans 60

3.3.2 Elementary reasonings 62

3.3.3 Evaluating *min-card* and *max-card* 74

3.4 Solving the dissemination problem 82

3.4.1 Integer linear programming 82

3.4.2 Additional constraints 84

3.5 Computational results 84

3.5.1 About the benchmarks 85

3.5.2 About the models 87

3.5.3 About the preprocessing procedures 88

3.6 Conclusion 95

3.1 Dominance rules

The solving techniques discussed in the present thesis are based on a number of dominance rules (see [31] for a comprehensive paper on that topic) which dramatically improve performances of any enumeration algorithm. All these dominance rules will be defined in this first section. These results will form the basis for additional constraints and deduction algorithms to be presented in the following sections.

Firstly, it will be remarked that more than one transmission of the *same* unit to the *same* receiving node might occur during the *same* valid transfer plan. From an operational point of view, resources are wasted, while from a computational point of view, taking such solutions into account significantly *enlarges the search space* we consider – which makes it desirable to disallow redundant transfers.

Hence the definitions below and the ensuing dominance rule.

Definition 3.1

The transfer occurring at time $c \in \{0, 1, \ldots, m\}$ in transfer plan ϕ is said to be *null* if and only if no datum unit is transmitted during contact σ_c, *i.e.* $\phi(c) = \emptyset$.

Definition 3.2

The transfer occurring at time $c \in \{0, 1, \ldots, m\}$ in transfer plan ϕ is said to be *improving* if and only if the receiving node r_c obtains a new datum unit during contact σ_c, *i.e.* $|O_{r_c}^{c-1}| < |O_{r_c}^{c}|$.

Definition 3.3 – minimal transfer plan

A transfer plan ϕ is said to be *minimal* if and only if all its transfers are either null or improving, *i.e.* no node receives the same datum unit more than once.

Proposition 3.1

The set of *minimal* transfer plans is *dominant*.

Proof. Let ϕ be a valid non-minimal transfer plan. Therefore there exists at least one transfer which is neither null nor improving, *i.e.* $\exists c \in \{1, 2, \ldots, m\}$ such that $\phi(c) = \{k\} \subseteq O_{r_c}^{c-1}$. The transfer plan ϕ' – obtained by copying ϕ and by setting $\phi'(c) = \emptyset$ – has the same dissemination length than ϕ, that is $\lambda(\phi') = \lambda(\phi)$. This process is to be repeated as long as the new transfer plan is not minimal. □

The idea behind minimal transfer plans may be further reinforced by only considering the minimal transfer plans during which any non-recipient node forwards any datum unit it receives at least once. From an operational point of view, this means avoiding transferring data to a non-recipient node which is not able to contribute to a better dissemination of the datum.

Formally, it gives the following dominance rule.

Definition 3.4 – strictly-minimal transfer plan

A transfer plan ϕ is said to be *strictly-minimal* if and only if it is minimal and every non-recipient node forwards at least once all the datum units it receives.

Proposition 3.2

The set of *strictly-minimal* transfer plans is *dominant*.

Proof. Let ϕ be a minimal, non-strictly-minimal, transfer plan. There exists at least one transfer $\phi(c)$, $c \in \{1, 2, \ldots, m\} \,|\, r_c \in \mathcal{N}\backslash\mathcal{R}$, such that $\phi(c) = \{k\}$ ($k \in \mathcal{D}$) and $\forall c' \in \{c+1, \ldots, m\} \,|\, s_{c'} = r_c$, $\phi(c') \neq \{k\}$ – *i.e.* a non-recipient node $r_c \in \mathcal{N}\backslash\mathcal{R}$ obtains datum unit $k \in \mathcal{D}$ during contact $\sigma_c \in \sigma$, but never forwards it. The transfer plan ϕ' obtained by copying ϕ, and then by setting $\phi'(c) = \emptyset$ has the same dissemination length, *i.e.* $\lambda(\phi') = \lambda(\phi)$. The process is to be repeated as long as the transfer plan is not strictly-minimal. □

Unfortunately, in practice, this dominance rule is found to be less efficient than the minimality rule introduced above. Our numerical tests even showed that it is beneficial to schedule improving transfers as possible, rather than trying to reduce their number at a high computational cost.

Therefore we propose the following dominance rule.

Definition 3.5

A transfer plan ϕ is said to be *active* if and only if there exists no contact $\sigma_c \in \sigma$ in which a datum unit $k \in \mathcal{D}$ is transmitted from s_c to r_c, where the same transmission could have been done earlier making better use of a *non-improving* transfer $\phi(c')$. Formally,

$$\forall c \in \{2, 3, \ldots, m\} \text{ such that } |O^{c-1}_{r_c}| < |O^{c}_{r_c}|,$$
$$\nexists c' \in \{1, \ldots, c-1\} \text{ with } r_{c'} = r_c,\ \phi(c) \subseteq O^{c'-1}_{s_{c'}} \text{ and } O^{c'-1}_{r_{c'}} = O^{c'}_{r_{c'}}$$

Proposition 3.3

The set of *active* transfer plans is dominant.

Proof. Let ϕ be a non-active transfer plan. There exist $c, c' \in \{1, 2, \ldots, m\}$ and $k \in \mathcal{D}$ such that $c' < c$, $r_c = r_{c'}$, $k \in O_{s_{c'}}^{c'-1}$, $k \notin O_{r_{c'}}^{c'-1}$, $O_{r_{c'}}^{c'-1} = O_{r_{c'}}^{c'}$, and $\phi(c) = \{k\}$. Let us consider ϕ', the transfer plan obtained first by copying ϕ and then by setting $\phi'(c') = \{k\}$. The dissemination length of transfer plan ϕ' is better than or equal to the dissemination length of ϕ, *i.e.* $\lambda(\phi') \leq \lambda(\phi)$. The process is repeated until ϕ' is active. $\phi'(c)$ can also be set to $\emptyset$ (at each iteration) if minimality properties have to be maintained. □

In an active transfer plan, an improving transfer might still be ignored in favour of another non-improving or null transfer. In particular, there might be $c' \in \{1, \ldots, m-1\}$ and $k \in \mathcal{D}$ with $k \in O_{s_{c'}}^{c'-1}$, $k \notin O_{r_{c'}}^{c'}$, and $O_{r_{c'}}^{c'-1} = O_{r_{c'}}^{c'}$, **if** $r_{c'} \notin \mathcal{R}$ **and** $\forall c \in \{c', \ldots, m\} \,|\, r_c = r_{c'}$, $\phi(c) \neq \{k\}$.

The dominance rule below strengthens the idea of an active transfer plan by ensuring that a fruitless transfer can never be preferred to an improving transfer.

Definition 3.6

A transfer plan ϕ is said to be *strictly-active* if and only if all transfers are improving when possible, *i.e.* $\forall c \in \{1, 2, \ldots, m\}$, if $\exists k \in \mathcal{D}$ such that $k \in O_{s_c}^{c-1}$ and $k \notin O_{r_c}^{c-1}$, then $|O_{r_c}^{c-1}| < |O_{r_c}^{c}|$.

Proposition 3.4

The set of *strictly-active* transfer plans is dominant.

Proof. Let ϕ be a non-strictly-active transfer plan, *i.e.* $\exists \sigma_c \in \sigma$ and $\exists k \in \mathcal{D}$ such that $k \in O_{s_c}^{c-1}$, $k \notin O_{r_c}^{c-1}$, and $O_{r_c}^{c-1} = O_{r_c}^{c}$. Then, let ϕ' be the transfer plan obtained by copying ϕ, and by setting $\phi'(c) = \{k\}$. The dissemination length of ϕ' is necessarily better than or equal to that of ϕ, *i.e.* $\lambda(\phi') \leq \lambda(\phi)$. The process is repeated until ϕ' is strictly-active. □

Remark 3.1

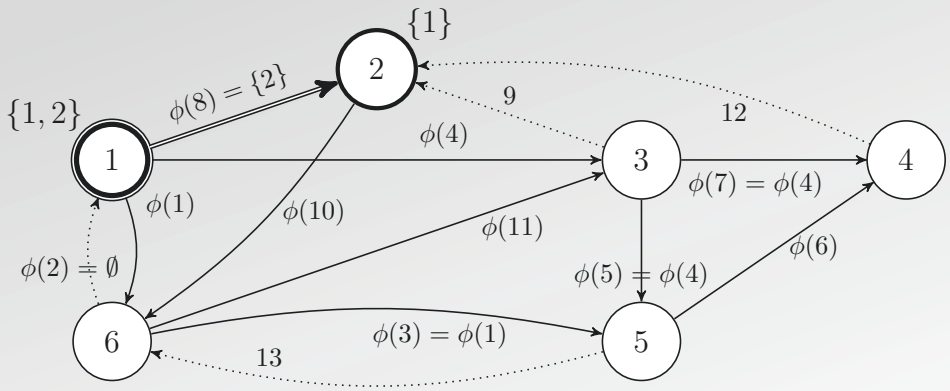
A strictly-active transfer plan is active (by definition). Moreover we can show that the sets of minimal-active and minimal-strictly-active transfer plans are both dominant (by combining the above proofs).

The remainder of the chapter will focus on algorithms designed to identify the non-minimal or the non-strictly-active transfer plans. Beforehand we will illustrate how these dominance rules can help us to solve the problem.

To this end, we consider the instance depicted in Figure 1.1 (on page 11):

- **dominance-rule-based deductions** – First node 1 initially possesses all units, whereas node 6 starts with an empty buffer, *i.e.* $\mathcal{O}_1 = \{1, 2\}$ and $\mathcal{O}_6 = \emptyset$. Thus, $\phi(1) \neq \emptyset$ holds in every strictly-active transfer plan, *i.e.* $\phi(1) = \{1\} \vee \phi(1) = \{2\}$. It also means that node 6 possesses either datum unit 1 or datum unit 2 afterwards. Since node 1 possesses these two units from the start, then $\phi(2)$ is null in all minimal strictly-active transfer plans. Therefore, in practice, contact σ_2 can be removed from the instance before any computation is done. With the same approach, it can be shown that $\phi(3) = \phi(1)$ and $\phi(8) = \{2\}$ hold in any minimal strictly-active transfer plan. Figure 3.1 illustrates the results that can be collected by applying these methods over each contact (in the order of the sequence). Five contacts are seen to have been set.
- **delivery-requirement-based deductions** – A feasible transfer plan has still to be found. It needs to be decided how to continue the process, *e.g.* by deciding $\phi(1) = \{1\}$ or, conversely, $\phi(1) = \{2\}$. Let us assume that $\phi(1) = \{2\}$. There exists only one possibility for delivering datum unit 1 to nodes 6 and 5, with $\phi(4) = \phi(5) = \phi(7) = \phi(10) = \{1\}$. Next, to transfer datum unit 2 to node 4, the only remaining possibility is to set $\phi(6) = \{2\}$, *cf.* Figure 1.1. It should be remarked that a symmetric solution can be found with $\phi(1) = \{1\}$. These transfer plans are both minimal, strictly-active and optimal.

In fact, the methodology described above is applied within any branching algorithm, *e.g.* a branch-and-bound or a branch-and-cut algorithm. Decisions and backtracks constitute the *branching stage* (*i.e.* generation and selection of nodes), with local deductions performed at every node of the search tree to filter dominated solutions, *e.g.* through constraint propagation or cuts.



$$\phi(1) \in \{\{1\},\{2\}\} \bullet \phi(2) = \emptyset \bullet \phi(3) = \phi(1) \bullet \phi(4) \in \{\{1\},\{2\}\}$$
$$\phi(5) \in \{\phi(4), \emptyset\} \bullet \phi(6) \in \{\phi(1), \phi(4)\} \bullet \phi(7) \in \{\phi(4), \emptyset\} \bullet \phi(8) = \{2\}$$
$$\phi(9) = \emptyset \bullet \phi(10) = \{\{1\},\{2\}\} \bullet \phi(11) \in \{\phi(1), \phi(10)\} \bullet \phi(12) = \emptyset \bullet \phi(13) = \emptyset$$

Figure 3.1 – The knowledge collected using dominance-based deduction tools.

The following two sections describe how these deduction techniques have been implemented. Thereafter, Section 3.4 focuses on a particular branching procedure (based on integer-linear programming) that we proposed to solve the dissemination problem.

3.2 Transfer graph

In this section, we propose a graph model – the *transfer graph* – which enables a set of valid transfer plans to be represented, *i.e.* a subset of the search space associated with the dissemination problem. It is the data structure that forms the basis for some deduction algorithms that will be subsequently proposed (Section 3.3). These procedures update the transfer graph so that the search space is dynamically reduced.

These techniques will be applied to pre-process the instances (to reduce their size), and within an integer-linear-programming framework, *i.e.* within a branch-and-cut procedure (*cf.* Section 3.4).

3.2.1 About the transfer graph

First, let us recall that each state $O_i^t \subseteq \mathcal{D}$ with $i \in \mathcal{N}$ and $t \in \{0, 1, \ldots, m\}$ contains the subset of units possessed by node i after the first t contacts of the sequence. Let us also recall that a transfer plan is valid if nodes transmit only datum units that they have possessed from the outset, or that they have obtained as a result of previous transfers. Hence the following assertion:

$$\forall c \in \{1, 2, \ldots, m\},\ O_{s_c}^{c-1} = \mathcal{O}_{s_c} \cup \underbrace{\left(\bigcup_{\substack{t \in \{1,\ldots,c-1\} \\ r_t = s_c}} \phi(t) \right)}_{\substack{\text{the units obtained as a} \\ \text{result of former transfers}}}$$

From now on we will be looking at a graph – the *transfer graph* – designed to take account of dependencies between transfers. It is defined as follows.

Definition 3.7 – transfer graph

Let us consider an instance of the dissemination problem. The associated *transfer graph* is a directed acyclic graph $G_\phi = (V, A)$ with $V = V_D \cup V_T$, $A = A_1 \cup \cdots \cup A_m \subseteq V_T \times V$, where V and A are built as follows:

1. $V_D = \{{}^v\{k\} \,|\, k \in \mathcal{D}\}$ – where vertex ${}^v\{k\}$ is associated with datum unit k;
2. $V_T = \{{}^v\phi(c) \,|\, c \in \{1, 2, \ldots, m\}\}$ – where vertex ${}^v\phi(c)$ is associated with transfer $\phi(c)$;
3. $\forall c \in \{1, 2, \ldots, m\}$,

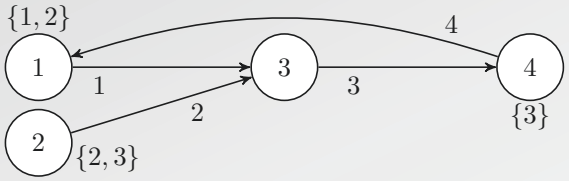
$$A_c = \underbrace{\{({}^v\phi(c), {}^v\{k\}) \,|\, k \in \mathcal{O}_{s_c}\}}_{\substack{\text{corresponds to the units} \\ \text{initially possessed by } s_c}} \cup \underbrace{\{({}^v\phi(c), {}^v\phi(t)) \,|\, t \in \{1, \ldots, c-1\} \text{ and } r_t = s_c\}}_{\substack{\text{corresponds to the units obtained by } s_c \\ \text{during the contacts which precede contact } \sigma_c}}$$

In such a graph, an arc between two vertices ${}^v\phi(c) \in V_T$ and ${}^v\{k\} \in V_D$ symbolises the fact that node $s_c \in \mathcal{N}$ can transmit datum unit $k \in \mathcal{D}$ during contact $\sigma_c \in \sigma$ because it has possessed that datum unit from the outset, *i.e.*

(a) *an instance of the problem*

$$\begin{cases} \mathcal{N} = \{1,2,3,4\}; \mathcal{R} = \{1\}; \mathcal{D} = \{1,2,3\}; \\ \mathcal{O}_1 = \{1,2\}; \mathcal{O}_2 = \{2,3\}; \mathcal{O}_3 = \emptyset; \mathcal{O}_4 = \{3\}; \\ \sigma = [(1,3),(2,3),(3,4),(4,1)] \end{cases}$$

(b) *its evolving graph*



(c) *the corresponding transfer graph*

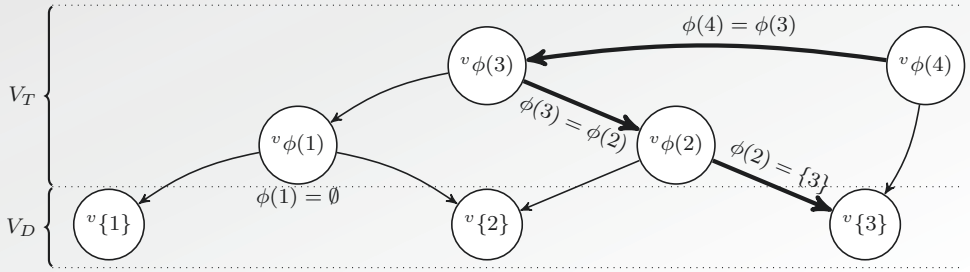


Figure 3.2 – An instance of the problem, its evolving graph, and finally the corresponding transfer graph.

$k \in \mathcal{O}_{s_c}$. Similarly, an arc between two vertices $^v\phi(c)$ and $^v\phi(t) \in V_T$ models the possibility that node s_c will forward during σ_c a unit that it has received during σ_t, as $t < c$ and $r_t = s_c$.

Remark 3.2

The transfer graph contains *no circuits*, and is *polynomial* in size of the instance with which it is associated (there are exactly $u+m$ vertices and $O(mu + m^2)$ arcs).

Remark 3.3

The vertices in V_D are termed *unit* vertices, and those in V_T are termed *transfer* vertices. As we shall see below, the distinction between the unit vertices and the transfer vertices is rarely required. We frequently need to refer to the unit $\phi(c)$ that is transmitted during a contact σ_c, without knowing precisely which unit $k \in \mathcal{D}$ it corresponds to, or even whether anything has really been transmitted. To clarify our notations, such an element is termed a *transfer value*. While a vertex ${}^v\{k\} \in V_D$ represents a datum unit and a vertex ${}^v\phi(c) \in V_T$ represents a transfer, we will use the notation ${}^v z \in V$ to refer to any kind of vertex.

These definitions are illustrated in Figure 3.2. The set of *unit* vertices is $V_D = \{{}^v\{1\}, {}^v\{2\}, {}^v\{3\}\}$, and the set of *transfer* vertices is $V_T = \{{}^v\phi(1), {}^v\phi(2), {}^v\phi(3), {}^v\phi(4)\}$. The set of *transfer values* is then

$$\{\{1\}, \{2\}, \{3\}, \phi(1), \phi(2), \phi(3), \phi(4)\}$$

The arcs going out of vertex ${}^v\phi(4)$ are $({}^v\phi(4), {}^v\{3\})$ (because s_4 has possessed unit 3 from the outset) and $({}^v\phi(4), {}^v\phi(3))$ (because contact σ_3 occurs before contact σ_4 and the receiver r_3 of that contact is s_4).

3.2.2 Transfer graph and subsets of transfer plans

By construction, the set A_c of arcs going out of each vertex ${}^v\phi(c) \in V_T$ defines the set of all values that transfer $\phi(c)$ can take in a valid transfer plan. Thus, a path $[{}^v\phi(c_p), \ldots, {}^v\phi(c_2), {}^v\phi(c_1), {}^v\{k\}]$ (of $p+1$ arcs) from a transfer vertex ${}^v\phi(c_p) \in V_T$ to a unit vertex ${}^v\{k\} \in V_D$ defines a possible way to route datum unit k from the source s_{c_1} to r_{c_p}. A solution where $\phi(c_1) = \cdots = \phi(c_p) = \{k\}$ is valid and enables nodes $r_{c_1}, \ldots, r_{c_p}$ to obtain datum unit k. More generally, an anti-branching rooted on a unit vertex ${}^v\{k\} \in V_D$ describes a routing to disseminate datum unit $k \in \mathcal{D}$. It corresponds to a set of branchings in the evolving graph.

Thus a transfer plan can equally be defined as a subgraph of the transfer graph made up of u vertex-disjoint anti-branchings that are each rooted on a different unit vertex. It corresponds to a spanning subgraph of the transfer graph containing at most one arc going out of each vertex. This constraint originates from the fact that at most one unit can be transmitted during each contact. Therefore selecting an arc $({}^v\phi(c), {}^v\{k\})$ is interpreted as $\phi(c) = \{k\}$, while selecting an arc $({}^v\phi(c_2), {}^v\phi(c_1))$ ensures that the unit transferred during

contact σ_{c_1} is transmitted during contact σ_{c_2}, that is $\phi(c_1) = \phi(c_2)$. In short, in this subgraph, transfer $\phi(c)$ is either considered to be null if no arc out of vertex ${}^v\phi(c)$ has been selected, or equal to $\{k\}$ if vertices ${}^v\phi(c)$ and ${}^v\{k\}$ are linked by a path. For example, in Figure 3.2 the bolded arcs define a way to transmit datum unit 3 to node 1. Moreover, the anti-branchings associated with vertices ${}^v\{1\}$ and ${}^v\{2\}$ contain no arcs, because these datum units are never transmitted.

Remark 3.4

Such a subgraph does *not* really need to be the union of anti-branchings. It prevents situations where a subgraph includes an arc $({}^v\phi(c_2), {}^v\phi(c_1))$, but does not include an arc going out of vertex ${}^v\phi(c_1)$ (the path starting from ${}^v\phi(c_2)$ does not end on a unit vertex). The meaning of this path – that is $\phi(c_2) = \phi(c_1) = \emptyset$ – can already be expressed by not selecting arc $({}^v\phi(c_2), {}^v\phi(c_1))$ at all. Taking such sets into account would therefore not enhance the expressiveness of the transfer graph, but would dramatically increase the number of subsets to be explored.

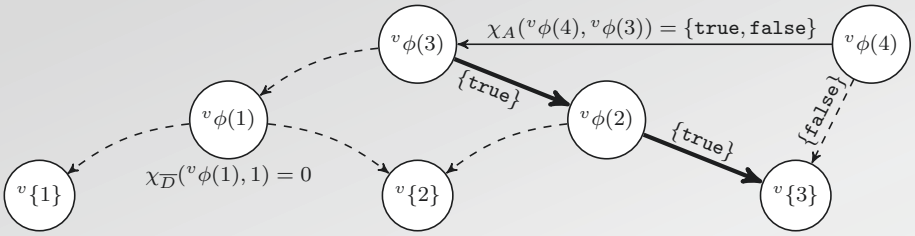
Remark 3.5

In Figure 3.2, it is worth nothing that the same transfer plan is obtained by selecting arc $({}^v\phi(4), {}^v\{3\})$ rather than arc $({}^v\phi(4), {}^v\phi(3)) \in A$. This is because the transfer plan represented by these subgraphs is not minimal (*cf.* Definition 3.3). Node 4 receives unit 3 during contact σ_3 even though it has possessed this unit from the outset. Therefore, during contact σ_4, node 4 can send either the unit it has possessed from the outset, or the new copy it has just received. Fortunately, a *minimal* transfer plan has a unique corresponding subgraph, and consequently we never address this case in practice.

To represent a particular subset of transfer plans, we need to be able to arbitrarily disallow certain arcs or, alternatively, stipulate that certain arcs must be used. To this end we introduce function $\chi_A : A \mapsto \mathcal{P}(\{\texttt{true}, \texttt{false}\})$, whose role is to indicate which arcs are allowed to be selected when a set of u anti-branchings (a transfer plan) is constructed. So, $\forall({}^v\phi(c), {}^vz) \in A$,

- $\chi_A({}^v\phi(c), {}^vz) = \{\texttt{true}\}$ means that arc $({}^v\phi(c), {}^vz)$ must be selected and that node s_c must transmit transfer value z during contact σ_c;

(a) *the transfer graph and some of its properties*



(b) *additional vertex properties*

- $\chi_\phi({}^v\{3\}) = \{\{3\}\}$
- $\chi_\phi({}^v\phi(1)) = \{\emptyset\}$
- $\chi_\phi({}^v\phi(2)) = \{\{2\}\}$
- $\chi_\phi({}^v\phi(4)) = \{\emptyset, \{3\}\}$
- $\chi_{\underline{D}}({}^v\{1\}, 3) = \infty$
- $\chi_{\overline{D}}({}^v\{2\}, 1) = 0$
- $\chi_{\underline{D}}({}^v\{3\}, 3) = 2$
- $\chi_{\overline{D}}({}^v\{3\}, 3) = \infty$

(c) *the two transfer plans represented by this transfer graph*

$$\phi_1 : \begin{cases} \phi_1(1) = \emptyset \\ \phi_1(3) = \phi_1(2) = \{3\} \\ \phi_1(4) = \emptyset \end{cases} \quad \phi_2 : \begin{cases} \phi_2(1) = \emptyset \\ \phi_2(3) = \phi_2(2) = \{3\} \\ \phi_2(4) = \phi_2(3) = \{3\} \end{cases}$$

Figure 3.3 – The transfer graph and its properties (the instance is given in Figure 3.2).

- $\chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$ means that arc $({}^v\phi(c), {}^vz)$ cannot be selected and that s_c must not transmit transfer value z during contact σ_c;
- $\chi_A({}^v\phi(c), {}^vz) = \{\texttt{true}, \texttt{false}\}$ means that no decision has been taken yet, *i.e.* there is no additional constraint upon this transfer.

As shown in Figure 3.3, this function can represent various constraints.

- For example, $\chi_A({}^v\phi(3), {}^v\phi(2)) = \{\texttt{true}\}$ expresses $\phi(3) = \phi(2)$.
- whereas $\chi_A({}^v\phi(1), {}^v\{1\}) = \{\texttt{false}\} \wedge \chi_A({}^v\phi(1), {}^v\{2\}) = \{\texttt{false}\}$ force transfer $\phi(1)$ to be null, *i.e.* $\phi(1) = \emptyset$.

Below, in the light of Remark 3.4, a transfer $\phi(c)$ ($c \in \{1, 2, \ldots, m\}$) will be considered as null if and only if all arcs $a \in A$ leaving or entering ${}^v\phi(c) \in V_T$ are such that $\chi_A(a) = \{\texttt{false}\}$.

3.2.3 Additional graph properties and complex subsets of transfer plans

The transfer graph, or more specifically function χ_A, represents *a set of valid transfer plans*. If all arcs $a \in A$ are such that $\chi_A(a) = \{\texttt{true}, \texttt{false}\}$, that is the whole transfer graph is retained, then all valid solutions are represented. On the other hand, if χ_A includes constraints, *e.g.* some arcs cannot or must be selected, then some transfer plans can no longer be built, and in this case only a subset of transfer plans is represented.

Nevertheless, it remains impossible to represent more complex subsets of transfer plans, *e.g.* the transfer plans such that $\phi(3) = \{2\}$ (since there is no arc between vertices ${}^v\phi(3)$ and ${}^v\{2\}$), or the set of transfer plans such that a given node receives a given datum unit between two given dates (since time windows are not represented in the graph).

However, these sets of transfer plans are all defined by sets of constraints which can easily be represented with the additional properties we introduce below:

- $\chi_\phi : V \mapsto \mathcal{P}(T_\phi)$ (recalling that $T_\phi = \{\emptyset, \{1\}, \ldots, \{u\}\}$) : this property specifies the domain of every transfer value – that is the set of values that transfer values are allowed to have, *e.g.* $\chi_\phi({}^v\phi(c)) = \{\emptyset, \{1\}, \{2\}\}$ indicates that transfer $\phi(c)$ must either be null, equal to $\{1\}$, or equal to $\{2\}$. In short,
  - $\forall {}^v\phi(c) \in V_T$, $\chi_\phi({}^v\phi(c))$ is the domain of $\phi(c)$ and $\phi(c) \in \chi_\phi({}^v\phi(c))$ must hold. Thus $\chi_\phi({}^v\phi(c)) = \{\emptyset\}$ means that $\phi(c)$ has to be null, while $\emptyset \notin \chi_\phi({}^v\phi(c))$ means that $\phi(c)$ cannot be null.
  - $\forall {}^v\{k\} \in V_D$, $\chi_\phi({}^v\{k\}) = \{\{k\}\}$ by convention.

  This property is used to represent various kinds of constraints. Looking back at Figure 3.3, $\chi_\phi({}^v\phi(1)) = \{\emptyset\}$ ensures that transfer $\phi(1)$ is null, while $\chi_\phi({}^v\phi(4)) = \{\emptyset, \{3\}\}$ enforces that transfer $\phi(4)$ is either null or equal to $\{3\}$. Transfer $\phi(2)$ has been set to $\{2\}$. Note that this property is used when a transfer is shown to be improving in all minimal strictly-active solutions represented by the transfer graph.
- $\chi_{\underline{D}}$ and $\chi_{\overline{D}} : V \times \mathcal{N} \mapsto \{0, \ldots, m, \infty\}$: these two properties specify an earliest and a latest date at which each node is allowed to receive each transfer value ($\infty = m + 1$ in practice):
  - $\forall {}^v z \in V$, $\forall i \in \mathcal{N}$, $\chi_{\underline{D}}({}^v z, i)$ forbids node i to receive transfer value z too soon, *i.e.* $\forall t \in \{0, \ldots, m\}$, if $t < \chi_{\underline{D}}({}^v z, i)$, then $z \not\sqsubseteq O_i^t$;

– $\forall {}^v z \in V, \forall i \in \mathcal{N}$, $\chi_{\overline{D}}({}^v z, i)$ imposes a deadline on node i to obtain transfer value z, *i.e.* $\forall t \in \{0, \ldots, m\}$, if $t \geq \chi_{\overline{D}}({}^v z, i)$, then $z \subseteq O_i^t$ must hold.

Node i is allowed to receive transfer value z during contacts occurring between dates $\chi_{\underline{D}}({}^v z, i)$ and $\chi_{\overline{D}}({}^v z, i)$. If we look back at the example in Figure 3.3, node 1 is constrained to possess $\phi(1)$ from the outset as $\chi_{\overline{D}}({}^v \phi(1), 1) = 0$ – *i.e.* $\phi(1) \subseteq \mathcal{O}_1$ must hold, regardless of the value of $\phi(1)$ (the fact that $\phi(1)$ is null as a result of χ_ϕ does not matter here). In addition, node 3 is not allowed to receive unit 3 before contact σ_2, since $\chi_{\underline{D}}({}^v\{3\}, 3) = 2$, and is even not required to possess this unit at the end of a transfer plan as $\chi_{\overline{D}}({}^v\{3\}, 3) = \infty$.

Remark 3.6

The properties focus on different aspects of the problem. Although linked by constraints, the different properties have their particular features and are not interchangeable. Redundancy is not really a problem in practice, but deduction algorithms must take account of it, for example in ensuring that $\forall c \in \{1, \ldots, m\}$, all arcs $a \in A$ leaving or entering a transfer vertex ${}^v\phi(c) \in V_T$ are such that $\chi_A(a) = \{\texttt{false}\}$ when $\chi_\phi({}^v\phi(c)) = \{\emptyset\}$ – and vice versa...

Altogether, the transfer graph shown in Figure 3.3, (*cf.* the instance given in Figure 3.2) represents the set $\{\phi_1, \phi_2\}$ of transfer plans where the transfer arising during contact σ_1 is null, and where datum unit 3 is transmitted from node 2 to node 4 during contacts σ_2 and σ_3. Then, the fourth transfer must either be null, *i.e.* $\phi_1(4) = \emptyset$, or improving, *i.e.* $\phi_2(4) = \phi_2(3) = \phi_2(2) = \{3\}$. Transfer plan ϕ_2 is seen not to be a solution, since it does not fulfil delivery requirements (datum unit 3 is not transmitted to recipient node 1).

What represents what – a short summary of Sections 3.2.1 to 3.2.3:

1. the transfer graph *represents* the whole set of valid transfer plans;
2. a spanning subgraph of the transfer graph (described by function χ_A) thus *represents* a subset of valid transfer plans;
3. a spanning subgraph of the transfer graph which includes at most one arc going out of each vertex *represents* one valid transfer plan;
4. we limit the search to the sets of anti-branchings (which are rooted on unit vertices) in order to avoid considering equivalent solutions;
5. other properties $\chi_\phi + \chi_{\underline{D}} + \chi_{\overline{D}}$ enable us to refine the subset of transfer plans represented by the transfer graph and χ_A.

3.2.4 Using the transfer graph

The transfer graph (and its properties) can be used to represent a state in a branching algorithm which searches for a valid transfer plan. The properties can be used to separate a set of transfer plans during the branching stage – *e.g.* given a unit $k \in \mathcal{D}$ and a contact $\sigma_c \in \sigma$, the branches $\chi_\phi({}^v\phi(c)) = \{\{k\}\}$ and $\{k\} \notin \chi_\phi({}^v\phi(c))$ separate the transfer plans according to whether or not $\phi(c) = \{k\}$. The properties are always initialized with the constraints of the problem only, so that the whole set of valid transfer plans (the whole search space) is represented when the solving procedure starts. For example, for all datum units $k \in \mathcal{D}$ and all nodes $i \in \mathcal{N}$ – assuming that α refers to the first contact where $i = r_\alpha$ – we set:

- $\chi_{\underline{D}}({}^v\{k\}, i) = \chi_{\overline{D}}({}^v\{k\}, i) = 0$ **if** $k \in \mathcal{O}_i$;
- $\chi_{\underline{D}}({}^v\{k\}, i) = \alpha$ and $\chi_{\overline{D}}({}^v\{k\}, i) = m$ **if** $k \notin \mathcal{O}_i$ and $i \in \mathcal{R}$;
- $\chi_{\underline{D}}({}^v\{k\}, i) = \alpha$ and $\chi_{\overline{D}}({}^v\{k\}, i) = \infty$ otherwise.

Besides, the transfer graph can also be used to apply deductive elements locally. The properties can be updated to express new knowledge, *e.g.* value $\emptyset$ can be removed from set $\chi_\phi({}^v\phi(c))$ if it is shown that transfer $\phi(c)$ cannot be null in a dominant solution. This enables dominated solution to be removed where possible.

3.3 Deductive elements

In this section we propose several elements of deduction (based either on the dominance rules discussed in Section 3.1, or on the problem itself). Given a transfer graph, we would like to update its properties (χ_A, χ_ϕ, $\chi_{\underline{D}}$ and $\chi_{\overline{D}}$) to reduce the search space it represents (so that dominated solutions can be ignored during the search for an optimal valid transfer plan).

3.3.1 Finding non-minimal transfer plans

Let us first consider the following proposition.

Proposition 3.5

Let $k \in \mathcal{D}$ be a datum unit and $\sigma_c \in \sigma$ be a contact.

1. [*minimality*] Node r_c cannot receive multiple copies of unit k in a minimal solution, which means that it is not necessary to consider the transmission of unit k to node r_c during σ_c if it is known that r_c already possesses k when σ_c occurs, *i.e.* $c > \chi_{\overline{D}}({}^v\{k\}, r_c)$ implies that $\{k\} \notin \chi_\phi({}^v\phi(c))$.
2. [*validity*] The validity constraint ensures that node s_c possesses the unit it sends, *i.e.* $c \leq \chi_{\underline{D}}({}^v\{k\}, s_c)$ implies $\{k\} \notin \chi_\phi({}^v\phi(c))$.
3. [*earliest-date*] Node r_c can receive datum unit k only if it is allowed to possess unit k, *i.e.* $c < \chi_{\underline{D}}({}^v\{k\}, r_c)$ implies $\{k\} \notin \chi_\phi({}^v\phi(c))$.

These statements can be generalized to any transfer value. For example, in a minimal transfer plan, transfer value $\phi(c)$, *whatever its value*, cannot be forwarded to a node that already possesses this element.

Hence the following corollary.

Corollary 3.1

Let $\sigma_c \in \sigma$ be a contact, and $({}^v\phi(c), {}^vz) \in A$ be an arc out of ${}^v\phi(c)$,

1. [*minimality*] $c > \chi_{\overline{D}}({}^vz, r_c) \Longrightarrow \chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$
2. [*validity*] $c \leq \chi_{\underline{D}}({}^vz, s_c) \Longrightarrow \chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$
3. [*properties*] $c < \chi_{\underline{D}}({}^vz, r_c) \Longrightarrow \chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$

The following deduction rules aim to refine the domain χ_ϕ of each transfer, by using the information we have about other transfers. Figure 3.4 illustrates the propositions below. Definition 3.8 is introduced for the sake of clarity.

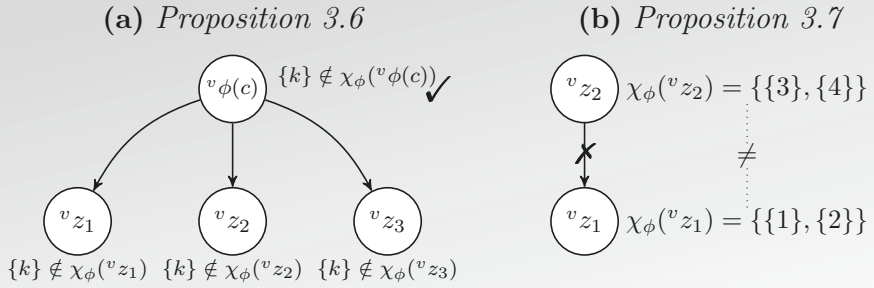


Figure 3.4 – Deduction tools for strengthening the domain χ_ϕ of each transfer.

Definition 3.8

$\Gamma : V \mapsto V$ indicates the remaining successors of each vertex, *i.e.* the set of values that transfers can still have in accordance with function χ_A:

- $\forall^v\phi(c) \in V_T$, $\Gamma(^v\phi(c)) = \{^v z \in V$ such that arc $(^v\phi(c), ^v z) \in A$ and $\texttt{true} \in \chi_A(^v\phi(c), ^v z)\}$;
- and $\forall^v\{k\} \in V_D$, we set $\Gamma(^v\{k\}) = \emptyset$ by convention.

Proposition 3.6

Given a datum unit $k \in \mathcal{D}$ and a contact $\sigma_c \in \sigma$, transfer $\phi(c)$ cannot be equal to value $\{k\}$ if none of the transfer values that may be transmitted during contact σ_c can have that value, *i.e.* $[\forall^v z \in \Gamma(^v\phi(c)), \{k\} \notin \chi_\phi(^v z)]$ implies $\{k\} \notin \chi_\phi(^v\phi(c))$.

Proposition 3.7

For any contact $\sigma_c \in \sigma$ and any transfer value z that may be transmitted during that contact – *i.e.* such that $^v z \in \Gamma(^v\phi(c))$ – transfer $\phi(c)$ cannot be equal to z if the domains of these transfer values are conflicting, *i.e.* $\chi_\phi(^v z) \cap \chi_\phi(^v\phi(c)) \subseteq \{\emptyset\}$ implies $\chi_A(^v\phi(c), ^v z) = \{\texttt{false}\}$.

These propositions are the basis for all the deduction algorithms discussed hereafter in this chapter. The aim will usually be to make use of bounds $\chi_{\underline{D}}$, $\chi_{\overline{D}}$ and Corollary 3.1 to remove arcs (to set χ_A-properties to $\{\texttt{false}\}$) in the transfer graph, so as to reduce the domain of transfers (to remove elements in χ_ϕ-properties) using Proposition 3.6. We try to show that some transfers are necessarily null in a dominant transfer plan, by proving that the recipient nodes always possess all the transfer values that the sending nodes are able to transmit in such a transfer plan.

The practical procedure for applying these deductive elements is described in Algorithm 3.1 – whose effectiveness will depend on the quality of bounds $\chi_{\underline{D}}$ and $\chi_{\overline{D}}$. We shall therefore devote the following subsection to consistency procedures designed to strengthen these bounds.

3.3.2 Elementary reasonings

In this subsection we propose algorithms designed to strengthen bounds $\chi_{\overline{D}}$ and $\chi_{\underline{D}}$. This will enable us to show that some transfers are always null in a dominant solution (*cf.* Subsection 3.3.1).

Bottom-up deductive reasoning

In this paragraph, we aim to make use of knowledge related to earlier contacts (corresponding to the vertices at the bottom of the transfer graph) in order to deduce information about later contacts (corresponding to the vertices at the top of the transfer graph).

Proposition 3.8

Let $i \in \mathcal{N}$ be a node and $t \in \{0, 1, 2, \ldots, m\}$ a time index. Let ${}^v\phi(c) \in V_T$ be a transfer vertex, and ${}^vz_1, \ldots, {}^vz_\alpha \in \Gamma({}^v\phi(c))$ its successors (transfer values $z_1, \ldots, z_\alpha \in T_\phi$ therefore correspond to what node s_c may transmit to node r_c during contact σ_c). If i possesses $z_1, z_2, \ldots,$ *and* z_α at time t, then it necessarily possesses $\phi(c)$ at time t. Whatever the value chosen for transfer $\phi(c)$ from among $z_1, \ldots, z_\alpha$, or $\emptyset$, node i possesses that element at time t. Thus $[\forall {}^vz \in \Gamma({}^v\phi(c)), t \geq \chi_{\overline{D}}({}^vz, i)]$ implies $\chi_{\overline{D}}({}^v\phi(c), i) \leq t$.

Algorithm 3.1 – MINIMALITY+VALIDITY consistency

Require: transfer graph $G_\phi = (V, A, \chi_A, \chi_\phi, \chi_{\underline{D}}, \chi_{\overline{D}})$;
Require: the vertex ${}^v\phi(c)$ to which we apply these deduction rules;

1:
2: *# consistency rules in accordance with Proposition 3.5.*
3: **for all** $k \in \mathcal{D}$ **do**
4: **if** $[c > \chi_{\overline{D}}({}^v\{k\}, r_c)] \vee [c < \chi_{\underline{D}}({}^v\{k\}, r_c)] \vee [c \leq \chi_{\underline{D}}({}^v\{k\}, s_c)]$
5: **then remove** $\{k\}$ **from** $\chi_\phi({}^v\phi(c))$;
6:
7: *# R is the union over ${}^vz \in \Gamma({}^v\phi(c))$ of sets $\chi_\phi({}^vz)$ – computed*
8: *# line 21 and used line 26.*
9: $R \leftarrow \emptyset$;
10:
11: **for all** ${}^vz \in \Gamma({}^v\phi(c))$ **do**
12:
13: *# consistency rules in accordance with Corollary 3.1.*
14: **if** $[c > \chi_{\overline{D}}({}^v\phi(c), r_c)] \vee [c > \chi_{\overline{D}}({}^vz, r_c)] \vee [c < \chi_{\underline{D}}({}^vz, r_c)]$
15: $\vee\, [c \leq \chi_{\underline{D}}({}^vz, s_c)]$ **then set** $\chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$;
16:
17: *# consistency rules in accordance with Proposition 3.7.*
18: **else if** $\forall\{k\} \in \chi_\phi({}^v\phi(c)), \{k\} \notin \chi_\phi({}^vz)$ **then**
19: **set** $\chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$;
20:
21: **else add** $\chi_\phi({}^vz)$ **to** R;
22:
23: **end for**
24:
25: *# consistency rules in accordance with Proposition 3.6.*
26: **for all** $\{k\} \in \chi_\phi({}^v\phi(c)) \backslash R$ **do remove** $\{k\}$ **from** $\chi_\phi({}^v\phi(c))$;
27:
28: *# this contact is "removed" from the model if possible.*
29: **if** $[\Gamma({}^v\phi(c)) = \emptyset] \vee [\chi_\phi({}^v\phi(c)) = \emptyset]$ **then**
30: **set** $\phi(c) = \emptyset$; and **update** the transfer graph:
31: • **set** $\chi_\phi({}^v\phi(c)) = \{\emptyset\}$;
32: • **for all** $i \in \mathcal{N}$ **do set** $\chi_{\underline{D}}({}^v\phi(c), i) = \chi_{\overline{D}}({}^v\phi(c), i) = 0$;
33: • **for all** $a \in A$ leaving or entering ${}^v\phi(c)$ **do**
34: **set** $\chi_A(a) = \{\texttt{false}\}$;
35:

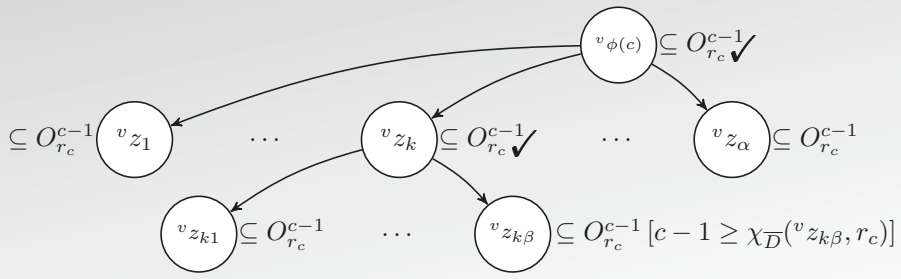


Figure 3.5 – The bottom-up procedure, *cf.* Proposition 3.8.

From a practical point of view, this leads to Algorithm 3.2. This algorithm visits a transfer vertex ${}^v\phi(c) \in V_T$ and tries to strengthen bounds $\chi_{\overline{D}}({}^vz, r_c)$, ${}^vz \in \Gamma({}^v\phi(c))$, associated with the recipient node r_c of transfer $\phi(c)$ and each transfer value $z \in T_\phi$ that node s_c could send during that transfer. If it can be shown that node r_c necessarily possesses a transfer value z at time $c-1$ (if $c-1 \geq \chi_{\overline{D}}({}^vz, r_c)$), then Corollary 3.1 enables us to remove arc $({}^v\phi(c), {}^vz)$ by setting $\chi_A({}^v\phi(c), {}^vz) = \{\texttt{false}\}$. Note that if all arcs going out of vertex ${}^v\phi(c)$ can be removed in this way, then it proves that transfer $\phi(c)$ is null in any dominant solution.

The bounds are strengthened in accordance with Proposition 3.8 within function **bottom-up**, with $i = r_c$ and $t = c-1$. ${}^vz$ refers to the vertex whose bounds must be refined. The function returns `true` if node i possesses ${}^vz$ at time t and puts a mark on all visited nodes to avoid redundant calculations. If the best known bound $\chi_{\overline{D}}({}^vz', r_c)$ of a vertex $z' \in \Gamma(z)$ is greater than $c-1$ (if a condition to deduce that $c-1 \geq \chi_{\overline{D}}({}^vz', r_c)$ is not fulfilled), we attempt to refine it recursively (line 18). This algorithm is depicted in Figure 3.5.

Remark 3.7

Although Algorithm 3.2 traverses vertices in depth-first, information is propagated from lower to upper vertices. The same outcome might also be achieved by calling function **bottom-up** with transfer vertices ${}^v\phi(1)$, ${}^v\phi(2), \ldots, {}^v\phi(c)$ in the order of the sequence. However, this would compel us to visit all the vertices "below" ${}^v\phi(c)$, whatever the situation.

Algorithm 3.2 – BOTTOM-UP consistency

Require: transfer graph $G_\phi = (V, A, \chi_A, \chi_\phi, \chi_{\underline{D}}, \chi_{\overline{D}})$;
Require: the vertex ${}^v\phi(c)$ to which we apply these deduction rules ;

unmark all vertices ; **boolean** $b \leftarrow$ `true` ;
for all ${}^vz \in \Gamma({}^v\phi(c))$ **do**
    # the recursive procedure is performed over each successor.
    $b \leftarrow b$ **and bottom-up** $({}^vz, r_c, c-1)$;

if $b =$ `true` **then**
    # it has been proved that $\phi(c)$, is possessed by r_c at time $c-1$.
    apply $\chi_{\overline{D}}({}^v\phi(c), r_c) \leq c-1$;

---

function bottom-up $({}^vz \in V, i \in \mathcal{N}, t \in \{0, \ldots, m\})$: **boolean**

    if $t \geq \chi_{\overline{D}}({}^vz, i)$ **then**
        return `true` ;

    if ${}^vz \in V_D$ **then**
        put a mark on ${}^vz$; and **return** `false` ;

    for all ${}^vz' \in \Gamma({}^vz)$ **such that** $t < \chi_{\overline{D}}({}^vz', i)$ **do**

        if ${}^vz'$ is marked **then**
            # the procedure has already failed to prove that node i
            # possesses z' at time t.
            put a mark on ${}^vz$; and **return** `false` ;

        # the procedure attempts to prove that i possesses z' at time t.
        else if ${}^vz'$ is *not* marked **and bottom-up** $({}^vz', i, t) =$ `false`
            then put a mark on ${}^vz$; and **return** `false` ;

    end for

    # according to Proposition 3.8, i possesses z at time t.
    apply $\chi_{\overline{D}}({}^vz, i) \leq t$; and **return** `true` ;

Top-down deductive reasoning

In this paragraph, we aim to make use of knowledge related to later contacts (and therefore shown at the top of the transfer graph) to deduce information about earlier contacts (located at the bottom of the transfer graph).

For example, let us consider the case where a node $i \in \mathcal{N}$ is the receiver in *only* two contacts $\sigma_{c1} = (s1, i)$ and $\sigma_{c2} = (s2, i)$ (*cf.* Figure 3.6). Then let us assume that sending nodes $s1$ and $s2$ possess two units 1 and $2 \in \mathcal{D}$, *i.e.* $O_{s1}^{c1-1} = O_{s2}^{c2-1} = \{1, 2\}$, and that node i did not possess any datum units at the outset, *i.e.* $\mathcal{O}_i = \emptyset$. Note that a datum unit is always transmitted during σ_{c1} in a strictly-active transfer plan, and that $\chi_\phi({}^v\phi(c1)) = \{\{1\}, \{2\}\}$ in the transfer graph. This remark also holds for contact σ_{c2}. As the datum units transmitted to node i are always different in a minimal transfer plan, we can deduce that node i necessarily possesses datum units 1 and 2 after these two contacts in a strictly-active minimal transfer plan, *i.e.* $\chi_{\overline{D}}({}^v\{1\}, i) \leq c2$ and $\chi_{\overline{D}}({}^v\{2\}, i) \leq c2$. We have actually shown that node i possesses at least two datum units from among subset $Z = \{1, 2\}$ after contact σ_{c2} in any dominant solution. Thus, $\phi(c1) \cup \phi(c2) = \{1, 2\}$ and $Z \subseteq O_i^{c_2}$.

The approach can be generalized to any set of transfer values.

Definition 3.9

Given a set ${}^vZ \subseteq V$ of vertices in the transfer graph, a node $i \in \mathcal{N}$, and a time index $t \in \{0, 1, \ldots, m\}$, *min-card*$({}^vZ, i, t)$ corresponds to a *lower bound* on the smallest number of transfer values $z \in Z$ that node i have after the t first contacts in any minimal strictly-active transfer plan, *i.e.* the smallest number of transfer values $z \in Z$ such that $z \subseteq O_i^t$ in such a transfer plan.

In Section 3.3.3 we will discuss how *min-card* may be evaluated.

Proposition 3.9

Let ${}^vZ \subseteq V$ be a set of vertices, $i \in \mathcal{N}$ a node, and let $t \in \{0, 1, \ldots, m\}$ denote a time index. If *min-card*$({}^vZ, i, t) = |{}^vZ|$ (*i.e.* if node i possesses at least $|{}^vZ|$ transfer values from among a set of $|{}^vZ|$ transfer values at time t), then node i possesses all the transfer values in Z at time t – *i.e.* $[\forall {}^vz \in {}^vZ,\ \chi_{\overline{D}}({}^vz, i) \leq t]$ necessarily holds.

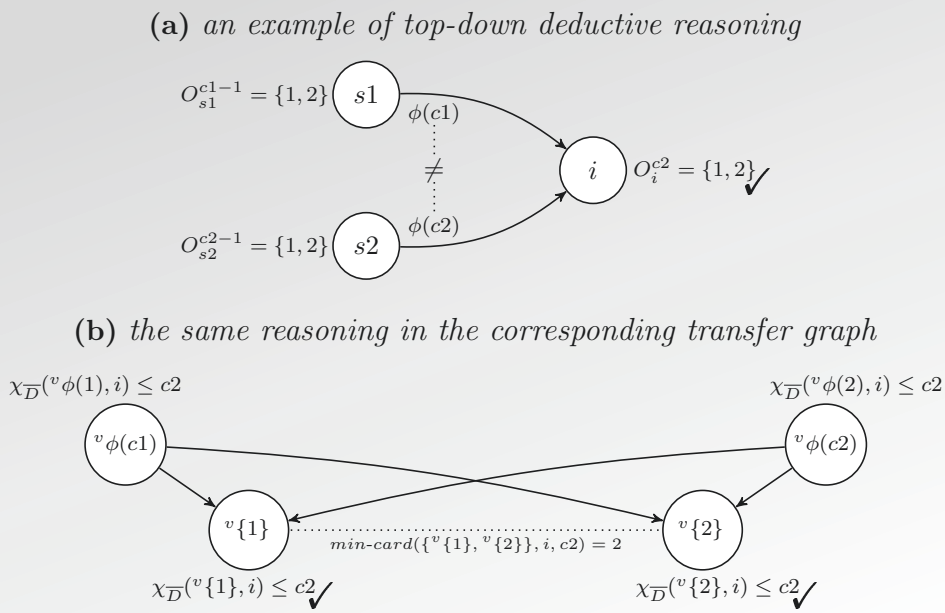


Figure 3.6 – The top-down procedure, *cf.* Proposition 3.9.

From a practical point of view, this leads to Algorithm 3.3. The procedure visits a transfer vertex ${}^v\phi(c) \in V_T$ and tries to prove that transfer $\phi(c)$ is null in any dominant solution. With this aim in view, it attempts once again to prove that any transfer value which could be transmitted during contact σ_c is already possessed by node r_c at time $c-1$ in all dominant transfer plans – that is $\forall {}^v z \in \Gamma({}^v\phi(c)),\ c-1 \geq \chi_{\overline{D}}({}^v z, r_c)$. This can sometimes be achieved using Proposition 3.9 with ${}^v Z = \Gamma({}^v\phi(c))$, $i = r_c$ and $t = c-1$, which involves evaluating $min\text{-}card(\Gamma({}^v\phi(c)), r_c, c-1)$.

The algorithm is quite straightforward. It should be noted, however, that strengthening the bound $\chi_{\overline{D}}({}^v z, r_c)$ of a vertex ${}^v z \in V$ might enable a better lower bound $min\text{-}card(\Gamma({}^v z), r_c, c-1)$ to be computed, and therefore enable Proposition 3.9 to be applied on child vertices – with ${}^v Z = \Gamma({}^v\phi(c))$, $i = r_c$, and $t = c-1$. In fact, the same deductive steps are repeated, from the upper to the lower vertices – the aim being to find matches between the subset of transfer values which are possessed by node r_c at time $c-1$ and other subsets of transfer values (corresponding to former contacts), or ideally to subsets of datum units.

Algorithm 3.3 – TOP-DOWN consistency

Require: transfer graph $G_\phi = (V, A, \chi_A, \chi_\phi, \chi_{\underline{D}}, \chi_{\overline{D}})$;
Require: the vertex ${}^v\phi(c)$ to which we apply these deduction rules;

```
# the procedure starts at vertex ^vφ(c) with i = r_c and t = c − 1.
top-down (^vφ(c), r_c, c − 1);
```

```
procedure top-down (^vφ(c) ∈ V_T, i ∈ 𝒩, t ∈ {0, 1, ..., m})

    # computing the smallest number of transfer values ^vz ∈ ^vZ
    # possessed by node i at time t.
    bound ← min-card(^vZ, i, t);

    if bound = |^vZ| then
        # node i possesses all transfer values ^vz ∈ ^vZ at time t.
        ∀^vz ∈ ^vZ, apply χ_D̄(^vz, i) ≤ t;
        ∀^vz ∈ ^vZ ∩ V_T, top-down (^vz, i, t);

    else if bound = |^vZ| − 1 and r_c = i and t = c − 1 then
        # node r_c will possess all transfer values ^vz ∈ ^vZ after σ_c.
        ∀^vz ∈ ^vZ, apply χ_D̄(^vz, i) ≤ c;
        ∀^vz ∈ ^vZ ∩ V_T, top-down (^vz, i, c);

end procedure
```

Strict-activity-based deductive reasoning

Let us now turn to the strict-activity-based deduction rule. First recall that a transfer plan is strictly-active *iif.* no transfer that might have been improving is not improving (*cf.* Definition 3.6). This means that a transfer is necessarily improving if the sending node possesses a unit that the receiving node does not possess. For example, if we look back at Figure 3.2b (given on page 53), transfer $\phi(3)$ is improving in a strictly-active minimal transfer plan, because node 3 possesses at least two units from among $\{1, 2, 3\}$ in such a solution, whereas node 4 only possesses unit 3 when contact σ_3 occurs.

Algorithm 3.4 – STRICT-ACTIVITY consistency

Require: transfer graph $G_\phi = (V, A, \chi_A, \chi_\phi, \chi_{\underline{D}}, \chi_{\overline{D}})$;
Require: the vertex ${}^v\phi(c)$ to which we apply these deduction rules ;

```
# the two bounds are evaluated – cf. Subsection 3.3.3 for details.
min-s ← min-card(V_D, s_c, c − 1) ; max-r ← max-card(V_D, r_c, c − 1) ;

# Proposition 3.9 still holds.
if min-s = |V_D| then
    for all ^v{k} ∈ V_D apply χ_D̄(^v{k}, s_c) ≤ c − 1 ;

# Proposition 3.10 is then applied.
if min-s > max-r then remove ∅ from χ_φ(^vφ(c)) ;
```

This leads to the following definition and the ensuing proposition.

Definition 3.10

Given a set ${}^vZ \subseteq V$ of vertices in the transfer graph, a node $i \in \mathcal{N}$, and a time index $t \in \{0, \ldots, m\}$, *max-card*$({}^vZ, i, t)$ corresponds to an *upper bound* on the greatest number of transfer values $z \in Z$ that node i have after the t first contacts in any minimal strictly-active transfer plan, *i.e.* the greatest number of transfer values $z \in Z$ such that $z \subseteq O_i^t$ in such a transfer plan.

In Section 3.3.3 we will discuss how *max-card* may be evaluated.

Proposition 3.10

Let $\sigma_c \in \sigma$ be a contact, and ${}^vZ \subseteq V$ a set of vertices. If node s_c always possesses more units than node r_c when contact σ_c occurs (in a minimal strictly-active transfer plan), then transfer $\phi(c)$ is necessarily improving (in such a transfer plan). *min-card*$({}^vZ, s_c, c-1) >$ *max-card*$({}^vZ, r_c, c-1)$ therefore implies $\emptyset \notin \chi_\phi({}^v\phi(c))$.

This condition is tested with ${}^vZ = V_D$ in practice, *cf.* Algorithm 3.4.

Delivery-requirement-based deductive reasoning

This paragraph looks at enforcing the delivery constraint, which states that every node $j \in \mathcal{N}$ has to obtain every unit $k \in \mathcal{D}$ before time $t = \chi_{\overline{D}}({}^{v}\{k\}, j)\}$ (except where $t = \infty$). To this end, we can utilize contacts $\sigma_c = (i, j)$ which occur before time t between a node $i \in \mathcal{N}$ and j. For $\phi(c) = \{k\}$ to be valid, other conditions have also to be fulfilled:

- $c > \chi_{\underline{D}}({}^{v}\{k\}, i)$ (node i possess unit k when contact σ_c occurs);
- $\chi_{\underline{D}}({}^{v}\{k\}, j) \leq c \leq \chi_{\overline{D}}({}^{v}\{k\}, j)$ (j is allowed to obtain k at time c);
- $\{k\} \in \chi_{\phi}({}^{v}\phi(c))$ (transfer $\phi(c) = \{k\}$ is allowed).

In consequence, if node j does not possess unit k from the outset, and if only one node $i \in \mathcal{N}$ is able to transfer k to node j on time, then an implicit constraint forces node i to obtain k, and this early enough to transmit it to node j on time. For example, in Figure 3.7, node 3 is the only one that can send datum unit 1 to recipient node 4 within time interval $\{\chi_{\underline{D}}({}^{v}\{1\}, 4) = 2, \ldots, \chi_{\overline{D}}({}^{v}\{1\}, 4) = 3\}$. So, node 3 has to obtain datum unit 1 during contact σ_1 at the latest. This can be formulated as follows.

Proposition 3.11

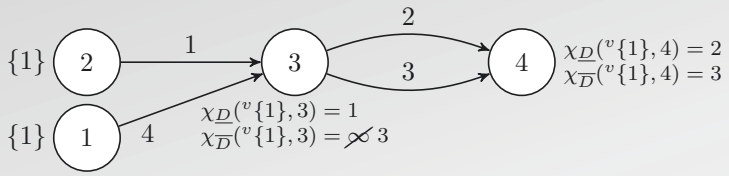
Let $k \in \mathcal{D}$ be a datum unit, and $j \in \mathcal{N}$ a node that is required to obtain k during the transfer plan – *i.e.* $k \notin \mathcal{O}_j$ and $\chi_{\overline{D}}({}^{v}\{k\}, j) \leq m$. Let then $\mathcal{N}^{j,k} = \{\, i \in \mathcal{N} \mid \exists \sigma_c = (i, j) \in \sigma$ with $c > \chi_{\underline{D}}({}^{v}\{k\}, i)$, $c \geq \chi_{\underline{D}}({}^{v}\{k\}, j)$, $c \leq \chi_{\overline{D}}({}^{v}\{k\}, j)$ and $\{k\} \in \chi_{\phi}({}^{v}\phi(c))\,\}$ denotes the set of nodes that can transmit datum unit k to node j in a valid transfer plan. If $\mathcal{N}^{j,k} = \{i\}$ is a singleton (*i.e.* if only one node i can send unit k to node j), then

$$\chi_{\overline{D}}({}^{v}\{k\}, i) < \chi_{\overline{D}}({}^{v}\{k\}, j) \text{ and } \chi_{\underline{D}}({}^{v}\{k\}, i) < \chi_{\underline{D}}({}^{v}\{k\}, j)$$

hold. On the other hand, if $\mathcal{N}^{j,k} = \emptyset$ (if there are no nodes able to send unit k to node j), then there is no solution fulfilling the set of constraints represented by the transfer graph.

For instance, in Figure 3.7a, Proposition 3.11 is applied with $k = 1$, $j = 4$, and $i = 3$. The only predecessor of node 4 is $\mathcal{N}^{4,1} = \{3\}$. Thus, $\chi_{\overline{D}}({}^{v}\{1\}, 3)$ can be adjusted to $\chi_{\overline{D}}({}^{v}\{1\}, 4) = 3$. Thereafter, as shown in Figure 3.7b, the proposition is applied with $k = 1$, $j = 4$, and $i = 1$. Contact σ_4 is no longer consistent with $\chi_{\overline{D}}({}^{v}\{1\}, 3)$, and node 2 then becomes the only node able to transmit datum unit 1 to node 3. Transfer $\phi(1)$ can even be set to $\{1\}$, since only one contact $\sigma_1 = (1, 3) \in \sigma$ exists.

(a) *Node 3 has to receive unit 1 before time 3 to forward it to node 4.*



(b) *Thus σ_4 occurs too late and transfer $\phi(1) = \{1\}$ is necessary.*

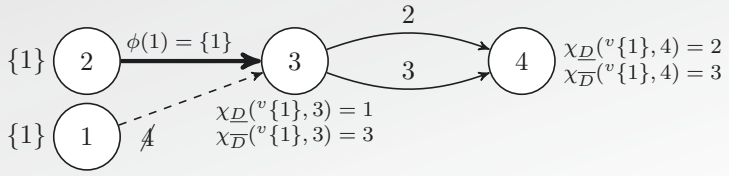


Figure 3.7 – The delivery-requirement-based deductive reasoning (Proposition 3.11).

From a practical point of view, this leads to Algorithm 3.5. The procedure ensures that bounds $\chi_{\underline{D}}$ and $\chi_{\overline{D}}$ are consistent – in the sense of Proposition 3.11 – for a given unit $k \in \mathcal{D}$ and all the nodes in $\mathcal{N}$. Note that S is a stack containing the pair of nodes $(i, j) \in \mathcal{N}^2$ such that $\mathcal{N}^{j,k} = \{i\}$, *i.e.* the nodes $\{\, j \in \mathcal{N}$ such that $k \notin \mathcal{O}_j$ and $\chi_{\overline{D}}(^v\{k\}, j) \leq m \,\}$ whose bounds $\chi_{\underline{D}}(^v\{k\}, j)$ and $\chi_{\overline{D}}(^v\{k\}, j)$ might need to be strengthened. Instruction **fail** notifies the calling function that there is no transfer plan fulfilling the set of constraints represented by the transfer graph. Therefore, if Delivery-Requirements is used within a branching algorithm, it prunes the current branch and then triggers a backtrack.

Global deductive elements

The deduction procedures described above are heuristically orchestrated in Algorithm 3.6. The vertices of the transfer graph are sequentially processed to reduce the domain of possibilities χ_ϕ associated with each transfer. If the domain of a transfer becomes a singleton, then it means that the transfer has been decided. If it becomes empty, or if the other properties are inconsistent, *e.g.* if $\exists^v z \in V$, $\exists i \in \mathcal{N} \mid \chi_{\underline{D}}(^v z, i) > \chi_{\overline{D}}(^v z, i)$, then it means that no transfer plan fulfils the constraints represented by the transfer graph.

Algorithm 3.5 – DELIVERY-REQUIREMENTS consistency

Require: transfer graph $G_\phi = (V, A, \chi_A, \chi_\phi, \chi_{\underline{D}}, \chi_{\overline{D}})$;
Require: the vertex ${}^v\phi(c)$ to which we apply these deduction rules;

 # the set of nodes whose bounds have to be checked is stored in stack S.
$S \leftarrow \{(i,j) \,|\, \mathcal{N}^{j,k} = \{i\}\}$;

 # the bounds are updated in accordance with Proposition 3.11.
while $S \neq \emptyset$ **do**

  $(i,j) \leftarrow S.back()$; $S.pop()$;
  apply $\chi_{\underline{D}}({}^v\{k\}, i) < \chi_{\underline{D}}({}^v\{k\}, j)$ **and** $\chi_{\overline{D}}({}^v\{k\}, i) < \chi_{\overline{D}}({}^v\{k\}, j)$;

  # unique transfers are even forced.
  if $\exists! \sigma_c = (i,j) \in \sigma \,|\, c > \chi_{\underline{D}}({}^v\{k\}, i)$, $c \geq \chi_{\underline{D}}({}^v\{k\}, j)$,
    $c \leq \chi_{\overline{D}}({}^v\{k\}, j)$, and $\{k\} \in \chi_\phi({}^v\phi(c))$
    then apply $\phi(c) = \{k\}$;

  # stack S is then updated if required.
  update $\mathcal{N}^{i,k}$; and **push** (z,i) **if** $\mathcal{N}^{i,k} = \{z\}$; or **fail if** $\mathcal{N}^{i,k} = \emptyset$;

end while

During each iteration $c \in \{1, 2, \ldots, m\}$, there is first an attempt to refine the bound $\chi_{\overline{D}}({}^v z, r_c)$ of every successor ${}^v z \in \Gamma({}^v\phi(c))$ of vertex ${}^v\phi(c)$ in the transfer graph (using both bottom-up and top-down deductions). This aims to prove that some transfer values which can be transferred by node s_c during contact σ_c are possessed by node r_c when the contact occurs (in any minimal and strictly-active transfer plan). Subsequently, the minimality consistency algorithm is charged with updating χ_A and χ_ϕ. The strict-activity- and the delivery-requirement-based consistency algorithms are run independently.

This way, we can often show that some transfers are null in any dominant transfer plan. The procedure can be repeated as long as changes are occurring in DELIVERY-REQUIREMENTS. Nevertheless, this is seen to be inefficient in practice (most of the transfers being fixed from the first call). We have tested Algorithm 3.6 as a *preprocessing procedure* aiming at detecting and removing fruitless contacts, so that the size m of sequence σ can be reduced before the

problem is solved. Moreover, we have tested it as a propagation procedure, whose goal is to set variables during each branching stage of a branch-and-cut algorithm (during the solving of an integer-linear-programming model which will be described in Section 3.4). However, before going any further, we must discuss how bounds *min-card* and *max-card* are computed in practice.

Algorithm 3.6 – GLOBAL-CONSISTENCY – heuristic

Require: transfer graph $G_\phi = (V, A, \chi_A, \chi_\phi, \chi_{\underline{D}}, \chi_{\overline{D}})$;
1:
2: # *the main loop aims to find which transfers are necessarily null.*
3: **for** $c : 1 \to m$ **do**
4:
5: # *we try to show that all the transfer values possessed by node* s_c
6: # *are also possessed by node* r_c *when contact* σ_c *occurs.*
7: BOTTOM-UP $(G_\phi, {}^v\phi(c))$; TOP-DOWN $(G_\phi, {}^v\phi(c))$;
8:
9: # *the strict-activity consistency procedure is run independently.*
10: STRICT-ACTIVITY $(G_\phi, {}^v\phi(c))$;
11:
12: # *the redundant transfers are finally removed in accordance*
13: # *with the minimality rule.*
14: MINIMALITY+VALIDITY $(G_\phi, {}^v\phi(c))$;
15:
16: **end for**
17:
18: # *the delivery constraints are then propagated as necessary.*
19: **while** at least one change occurs **do**
20: **for all** $k \in \mathcal{D}$ **do** DELIVERY-REQUIREMENTS (G_ϕ, k) ;
21:

3.3.3 Evaluating *min-card* and *max-card*

In this section, we will look at how to evaluate an upper and a lower bound on the number of transfer values possessed by a given node at a given time (*cf.* Definitions 3.9 and 3.10). The input data will always be one instance of the dissemination problem, one transfer graph (one subset of transfer plans), one subset ${}^{v}Z \subseteq V$ of its vertices, one node $i \in \mathcal{N}$, and finally one time index $t \in \{0, \ldots, m\}$. With regard to *max-card*, note that only the case ${}^{v}Z = V_D$ will be considered, *i.e.* we will have to find an upper bound of the number of *datum units* possessed by node i at time t (*cf.* Algorithm 3.4).

How to compute *max-card*

Let us consider the problem of evaluating $\textit{max-card}(V_D, i, t)$. We recall that the *different* datum units (represented by V_D) that a node $i \in \mathcal{N}$ can receive before time $t \in \{0, \ldots, m\}$ are associated with as many vertex-disjoint paths in the transfer graph. Each path has to start at a transfer vertex ${}^{v}\phi(c) \in V_T$ ($c \leq t$) where $r_c = i$, and end at a unit vertex ${}^{v}\{k\} \in V_D$. These paths must also fulfil the constraints which result from functions χ_A, χ_ϕ, $\chi_{\underline{D}}$ and $\chi_{\overline{D}}$ (*cf.* the summary given on page 58).

Therefore, by relaxing the constraints expressed with functions χ_ϕ, $\chi_{\underline{D}}$ and $\chi_{\overline{D}}$, the problem in hand can be reformulated as the problem of finding the greatest number of arc- and vertex-disjoint paths from a transfer vertex ${}^{v}\phi(c) \in V_T$ such that $c \leq t$ and $r_c = i$ to a unit vertex ${}^{v}\{k\} \in V_D$. Of course the search is limited to the subgraph defined by χ_A (*i.e.* the arcs $a \in A$ such that $\texttt{true} \in \chi_A(a)$). To solve this problem, we compute a maximum flow in a transportation network $G = (X, U, cap)$ comprised of the following elements (*cf.* Figure 3.8, instance Figure 3.2, page 53):

1. we consider a source vertex $src \in X$ and a sink vertex $snk \in X$;
2. with each vertex ${}^{v}z \in V$ in the transfer graph, we associate two vertices z_s and $z_e \in X$ and an arc $(z_s, z_e) \in U$ between these two vertices in the transportation network;
3. with each arc $a = ({}^{v}x, {}^{v}y) \in A$ such that $\texttt{true} \in \chi_A(a)$ in the transfer graph, we associate an arc $(x_e, y_s) \in U$ in the flow network;
4. we add an arc from the source node *src* to any vertex $z_s \in X$ which is associated with a transfer value possessed by node r_c at time t (where $\chi_{\overline{D}}({}^{v}z, i, t) \leq t$ in the transfer graph);
5. we add an arc from each unit vertex $\{k\}_e \in X$ to the sink node *snk*;

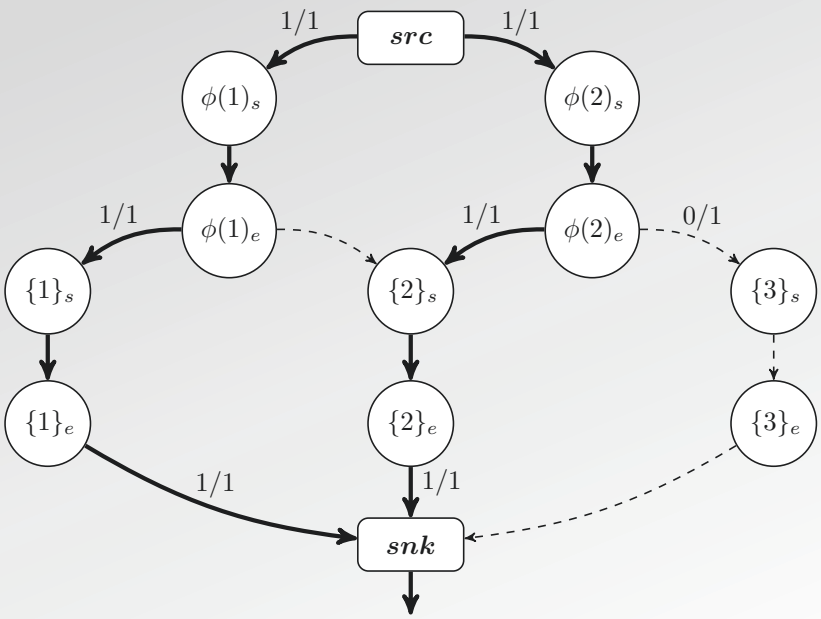


Figure 3.8 – An example of transportation network used to evaluate *max-card* (the instance is given in Figure 3.2 on page 53).

6. the capacity of each arc $a \in U$ is finally set to $cap(a) = 1$;
7. note that any node $z \in X$ that is not descendant of src, nor ascendant of snk can be removed.

Flows represent arc-disjoint paths (as all capacities are set to 1), and define a valid assignment of the transfer values possessed by node i at time t. Note that the value of each transfer $\phi(c)$, $c \in \{1, \ldots, m\}$ is unambiguous because at most one unit can flow out of each vertex $\phi(c)_e \in V$ (the paths are vertex disjoint). The constraints expressed by χ_A are fulfilled by construction.

Hence the following proposition.

Proposition 3.12

Let f_M be the maximum flow through G. $f_M = max\text{-}card\,(V_D, i, t)$ is an upper bound on the greatest number of units that node $i \in \mathcal{N}$ possesses after the first t contacts.

Finally, it is worth nothing that strict-activity and minimality constraints can be disregarded when computing $max\text{-}card({}^{v}Z, i, t)$ – because redundant and postponed transfers can only lead to a reduction in the number of datum unit received by node i.

What about *min-card* ?

Let us now turn to the problem of computing $min\text{-}card({}^{v}Z, i, t)$ (the smallest number of transfer values $z \in Z$ possessed by node i at time t in a dominant transfer plan). It might be tempting to use the same approach and to try to transform the smallest cardinal bound problem into a well-known flow-based problem. Unfortunately this does not work. The optimal solution will always be the null transfer plan in which no datum units are transferred at all, and therefore the lower bound will always be null. To get tight bounds, solutions must be required to be minimal strictly-active transfer plans (which prevents unjustified null transfers occurring). Unfortunately this constraint cannot be introduced into a flow problem. Note that it can be proved that computing an *exact* value $min\text{-}card({}^{v}Z, i, t)$ – *i.e.* the *precise* number of transfer values in ${}^{v}Z$ that node i possesses at time t in the worst case – is strongly NP-hard if solutions have to be strictly-active.

We were unable to find a satisfactory heuristic for the problem. Thus we have proposed a procedure for polynomially transforming an instance of the smallest cardinal bound problem, where ${}^{v}Z \subseteq V_D$ does not necessarily hold, into another instance where ${}^{v}Z \subseteq V_D$ holds, *i.e.* where *datum units only* need to be considered. In this case, the integer-linear-programming model defined in Section 3.4 can be adapted to our needs (with minor changes). Evaluating the exact value of *min-card* in the transformed instance gives rise to a lower bound of $min\text{-}card({}^{v}Z, i, t)$ in the original instance.

Unfortunately, the transformation is not always relevant and the method is then limited to some cases that we describe below.

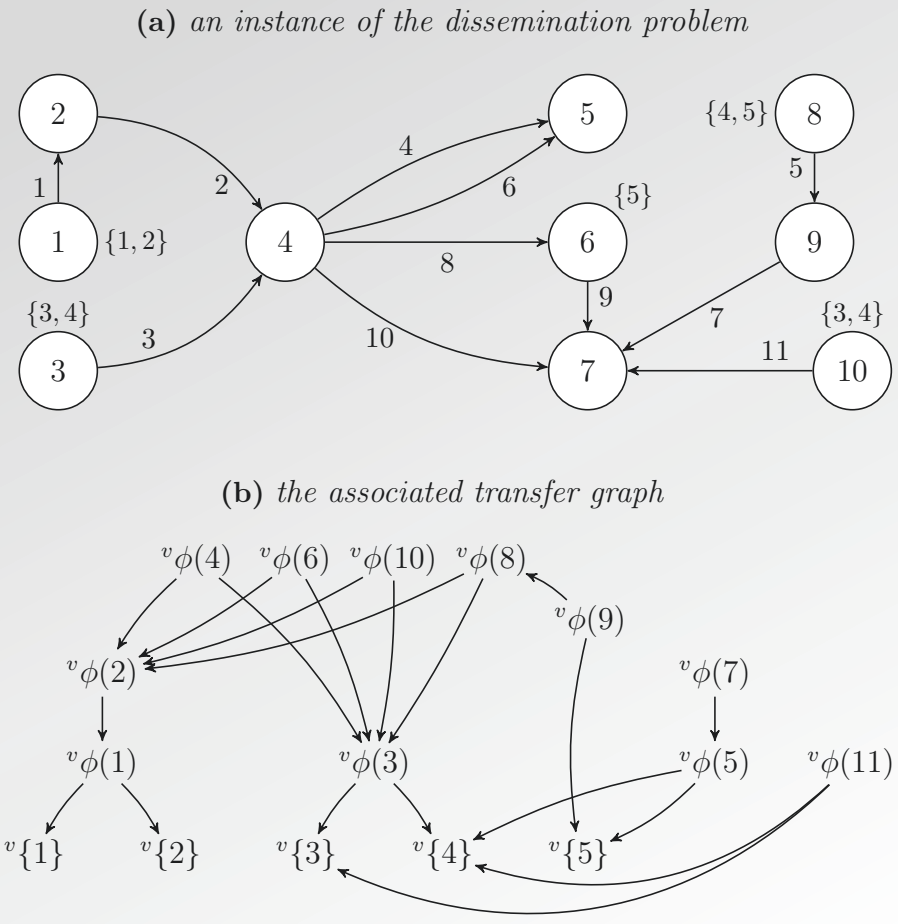


Figure 3.9 – An instance of the dissemination problem and the corresponding transfer graph.

How to perform this transformation

Let us consider the instance depicted in Figure 3.9. We would like to compute *min-card*$(\{{}^v\phi(1), {}^v\phi(3)\}, 5, 6)$, that is the smallest number of transfer values from among set $Z = \{\phi(1), \phi(3)\}$ possessed by node $i = 5$ after the first $t = 6$ contacts in a dominant solution.

As we stated above – in order to evaluate *min-card* (using integer linear programming) – we wish to reduce the original problem to an easier problem where ${}^vZ \subseteq V_D$. With this goal in mind, we propose adding a *virtual* datum unit $\{A\}$ to represent transfer value $\phi(1) \in Z$. Like $\phi(1)$, $\{A\}$ is transmitted to node 1 at time 1 via a fictitious source node s_1, while contact σ_1 is removed from the instance. Transfer $\phi(3) \in Z$ is then replaced by virtual datum unit $\{B\}$ in the same way. Finally, any contact (resp. node) whose representative arc (resp. vertex) does not belong to a journey to node $i = 5$, or which occurs after contact $\sigma_t = \sigma_6$, is removed from the instance, since it cannot help i to obtain new units on time. This instance is depicted in Figure 3.10a.

In fact, *min-card*$(\{{}^v\phi(1), {}^v\phi(3)\}, 5, 6)$ in the original instance is equal to *min-card*$(\{{}^v\{A\}, {}^v\{B\}\}, 5, 6) = 2$ in the new instance (we can compute this value since we are only considering datum units). Node 2 receives unit $\{A\}$ instead of transfer value $\phi(1)$ (during σ_1), but both represent one datum unit in the set $\{\{1\}, \{2\}\}$. Next node 4 receives $\{B\}$ instead of transfer value $\phi(3)$ (during σ_3), but both represent one unit in the set $\{\{3\}, \{4\}\}$. As the rest of the instance is not changed, units $\{A\}$ and $\{B\}$ play the same role as transfer values $\phi(1)$ and $\phi(3) \in Z$ in the original instance. Consequently, $\phi(1) \subseteq O_5^6$ and $\phi(2) \subseteq O_5^6$ hold in all dominant solutions.

In the general case, thanks to the transfer graph, the transformation can be automated using the following procedure.

1. To "replace" a transfer $\phi(c)$ by a *virtual* unit, it is sufficient to remove all descendants of vertex ${}^v\phi(c)$ in the original transfer graph, so that it becomes a leaf in the new transfer graph (*i.e.* a vertex that represents a datum unit). This virtual datum unit thus aggregates the choices that must be made regarding the removed vertices. This operation is to be repeated for all the vertices ${}^vz \in {}^vZ \cap V_T$. In this way, all the transfer values in ${}^vZ$ are units in the new instance.
2. Thereafter, the deletion of the irrelevant entities consists in removing the contacts occurring too late (*i.e.* $\{{}^v\phi(c) \in V_T \mid t < c\}$), together with the transfer values whose representative vertex is not a descendant of a vertex associated with node i in the transfer graph (*i.e.* $\{\, {}^vz \in V$ such that $\nexists {}^v\phi(c') \in V_T$, $c' \leq t$, $r_{c'} = i$ and ${}^vz$ is a descendant of ${}^v\phi(c')\,\}$).

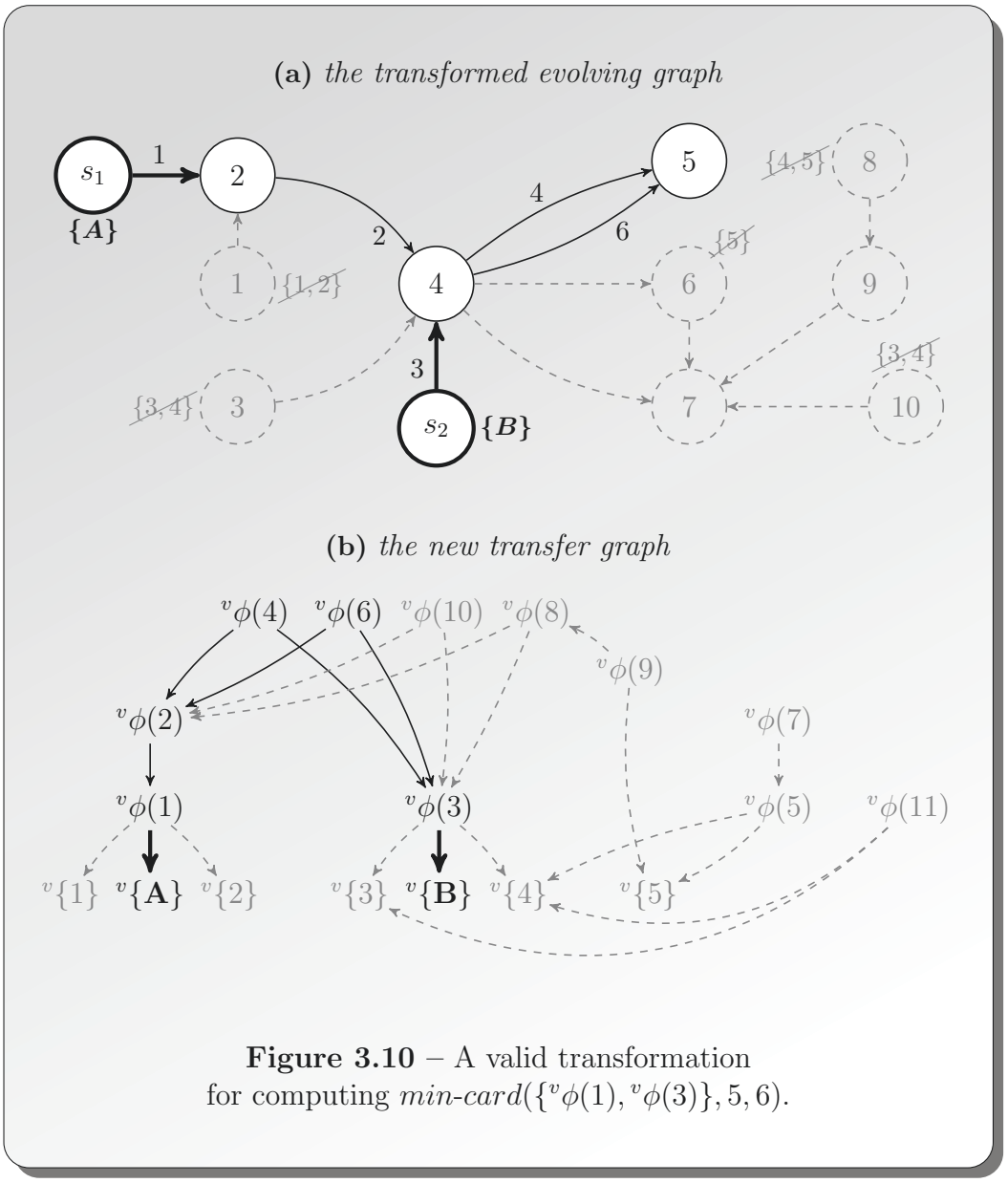


Figure 3.10 – A valid transformation for computing *min-card*$(\{^v\phi(1), ^v\phi(3)\}, 5, 6)$.

This transformation is shown in Figure 3.10b.

These operations sometimes transform other transfer vertices $^v\phi(c) \in V_T \backslash Z$ into leaves. For instance, in Figure 3.11, the transformation of transfer $\phi(3)$ into unit $\{B\}$ also transforms transfer $\phi(11)$ into unit $\{C\}$.

The limits of our approach

In some cases, not all of the successors of a vertex are removed. For example, in Figure 3.11b, the successor ${}^v\{4\}$ of vertex ${}^v\phi(5)$ is removed to transform $\phi(3)$ into $\{B\}$, while vertex ${}^v\{5\}$ is kept. In a such situation, the evaluation of *min-card* in the transformed instance is *not* a lower bound of *min-card* in the original instance. *Extra* transformations are required, because the set of choices concerning these transfers has been implicitly reduced. Some transfer plans might thus be ignored.

In this example, it would imply that $\phi(5) = \phi(7) = \{5\}$, and consequently that $\phi(8) = \phi(9)$. We would therefore find $\phi(9) = \{A\}$, $\phi(10) = \{B\}$, or vice versa, and finally conclude that $\textit{min-card}(\{{}^v\phi(1), {}^v\phi(3)\}, 7, 11) = 2$. Yet the minimal strictly-active transfer plan ϕ such that – $\phi(1) = \phi(2) = \phi(8) = \{1\}$, $\phi(3) = \phi(10) = \{3\}$, $\phi(5) = \phi(7) = \{4\}$, $\phi(9) = \{5\}$, and $\phi(11) = \emptyset$ – proves that node 7 can possess fewer than 2 units at time 11 in a dominant solution. Consequently the best bound is $\textit{min-card}(\{{}^v\phi(1), {}^v\phi(3)\}, 7, 11) = 1$, *e.g.* with $\phi(3) \subseteq O_5^{11}$ and $\phi(1) \not\subseteq O_5^{11}$.

In fact, for the bound to be computed consistently, transfer $\phi(5)$ needs to be transformed into a fourth virtual unit $\{D\}$ by removing the remaining descendant ${}^v\{5\}$ of vertex ${}^v\phi(5)$. In practice, such cases are never addressed, since they generally lead to *complex transformations* (triggered in chain) and *poor bounds*. Each transformation is actually a relaxation of some minimality constraints, in the sense that *dependent* sets of choices are being assimilated to *independent* units. For example, the transformation of transfers $\phi(3)$ and $\phi(11) \subseteq \{3, 4\}$ into virtual datum units $\{B\}$ and $\{C\}$ implicitly allows that $\phi(3) = \phi(11)$, although $r_3 = r_{11} = 10$.

The case where some transfer values in ${}^vZ$ have hierarchical relationships is not addressed either. Transforming ancestors implies deleting descendants, so that the target value $\textit{min-card}({}^vZ, i, t) = |{}^vZ|$ becomes unattainable from the beginning.

Proposition 3.13

Let ${}^vZ_T$ be the set of (virtual) datum units in the *transformed* instance (corresponding to the set of transfer values ${}^vZ$ in the original problem). Let $\textit{min-card}_T({}^vZ_T, i, t)$ refer to the smallest number of transfer values in Z_T that node i possesses at time t in a dominant transfer plan (computed in the transformed instance). $\textit{min-card}_T({}^vZ_T, i, t) = \textit{min-card}({}^vZ, i, t)$ is a lower bound of the smallest number of transfer values possessed by node i after the first t contacts in the original instance.

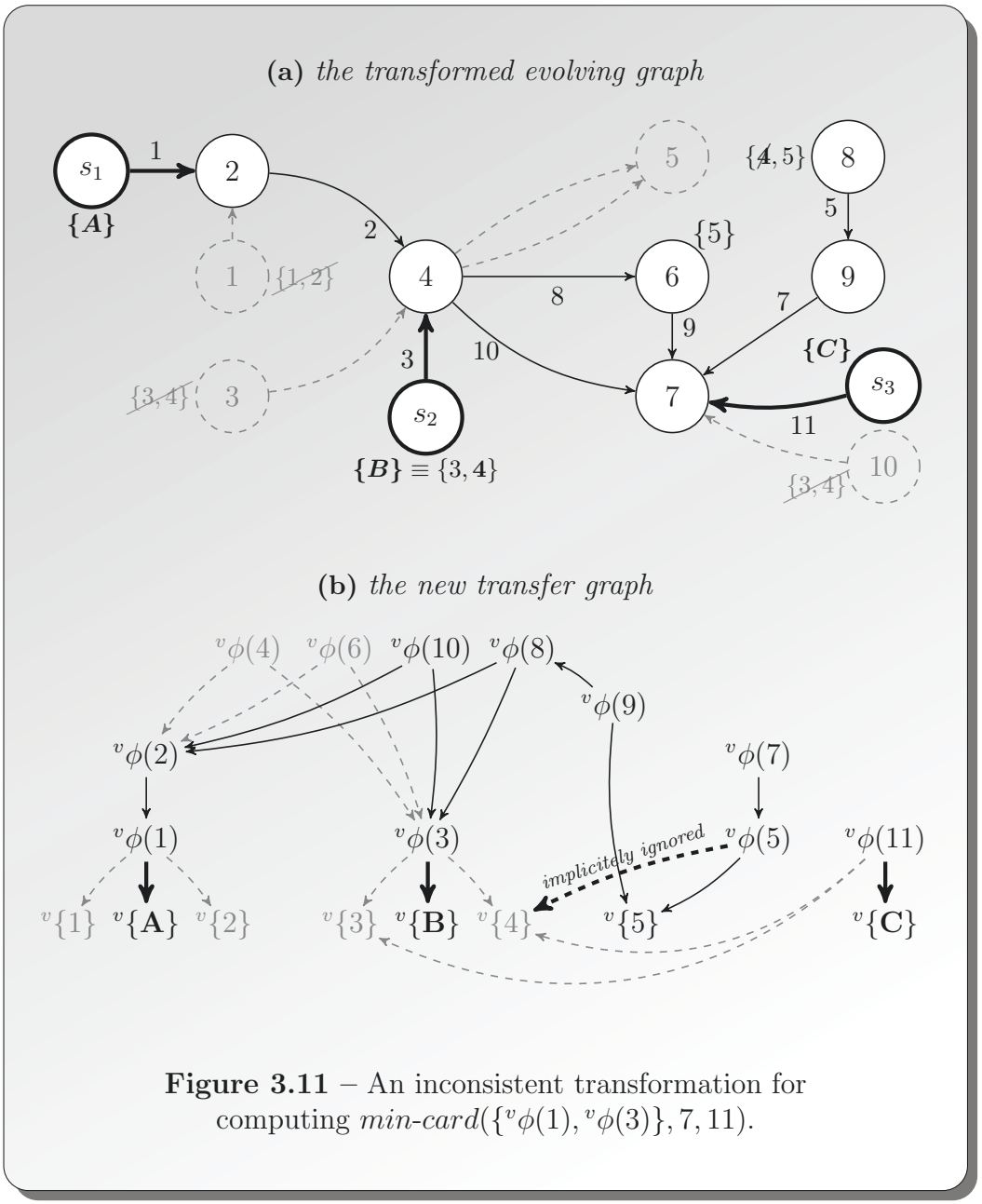


Figure 3.11 – An inconsistent transformation for computing $\textit{min-card}(\{{}^v\phi(1), {}^v\phi(3)\}, 7, 11)$.

In the following section we propose an integer-linear-programming model designed to solve the dissemination problem. Note that it is quite similar to the model we use in practice to compute $\textit{min-card}_T({}^vZ_T, i, t)$.

3.4 Solving the dissemination problem

In Section 3.1 we proposed several dominance rules which enable the search space to be reduced. Deduction procedures based on these results were then discussed in Sections 3.2 and 3.3. In this section we propose an integer-linear-programming model to solve the problem.

3.4.1 Integer linear programming

The integer-linear-programming model we propose is quite straightforward, and is based on a set of time-indexed boolean variables describing a transfer plan. For each node $i \in \mathcal{N}$, we define $\mathcal{T}_i = \{0\} \cup \{c \in \{1, 2, \ldots, m\} \,|\, r_c = i\}$, the set of time indexes at which the state of node i can change, *i.e.* at which node i can receive a datum unit. Subsequently, $\forall t \in \{0, 1, \ldots, m\}$, we define $\mathcal{T}_i(t) = \max\{t' \in \mathcal{T}_i \,|\, t' \leq t\}$. Thus, $\mathcal{T}_i(t)$ refers to the last contact occurring before time t where node i is the receiver.

The variables are defined as follows:

- $\forall k \in \mathcal{D}$, $\forall c \in \{1, \ldots, m\}$, $x_{k,c} = 1$ if datum unit k is transmitted from node s_c to node r_c during contact σ_c, and $x_{k,c} = 0$ otherwise.
- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, and $\forall t \in \mathcal{T}_i$, $y_{i,k,t} = 1$ if node i possesses unit k after contact σ_t, and $y_{i,k,t} = 0$ otherwise.
- $\forall t \in \{0, 1, \ldots, m\}$, $z_t = 1$ if there is a node $i \in \mathcal{R}$ which is not entirely served after the first t contacts, and $z_t = 0$ otherwise.

Remark 3.8

y-variables are indexed with sets $\mathcal{T}_i$ instead of set $\{0, 1, \ldots, m\}$. However, we may easily know whether a node $i \in \mathcal{N}$ possesses a datum unit $k \in \mathcal{D}$ at any index $t \in \{0, 1, \ldots, m\}$. Indeed, $y_{i,k,\mathcal{T}_i(t)} = 1$ if and only if node i possesses datum unit k after the first t contacts.

The model thus contains um x-variables, $u.(2m + n)$ y-variables and $m + 1$ z-variables.

Minimizing the dissemination length leads to the following objective.

$$\lambda^* = \min \sum_{t=0}^{m} z_t \tag{3.1}$$

Given a time index $t \in \{0, 1, \ldots, m\}$, variable z_t is null if and only if all the recipient nodes have been served at time t, *i.e.* $\forall i \in \mathcal{R}$, $\forall k \in \mathcal{D}$, $y_{i,k,\mathcal{T}_i(t)} = 1$. Hence the following constraints.

$$\forall i \in \mathcal{R},\ \forall t \in \mathcal{T}_i,\ \forall k \in \mathcal{D},\ z_t \geq 1 - y_{i,k,t} \tag{3.2}$$

Equations (3.2) can be aggregated in different ways:

$$\forall t \in \{0, \ldots, m\},\ \forall i \in \mathcal{R},\ z_t \geq 1 - \frac{1}{u} \sum_{k \in \mathcal{D}} y_{i,k,\mathcal{T}_i(t)} \tag{3.2-a}$$

$$\text{and/or } \forall t \in \{0, \ldots, m\},\ \forall k \in \mathcal{D},\ z_t \geq 1 - \frac{1}{|\mathcal{R}|} \sum_{i \in \mathcal{R}} y_{i,k,\mathcal{T}_i(t)} \tag{3.2-b}$$

$$\text{and/or } \forall t \in \{0, \ldots, m\},\ z_t \geq 1 - \frac{1}{u.|\mathcal{R}|} \sum_{i \in \mathcal{R}} \sum_{k \in \mathcal{D}} y_{i,k,\mathcal{T}_i(t)} \tag{3.2-c}$$

The efficiency of the variants will be discussed in Section 3.5.

Other constraints bind the x-variables (the decision variables) to the y- and the z-variables (the auxiliary variables). They ensure that each constraint is respected (*e.g.* the transfer plan must be valid):

- All recipients have to be served before the end of the time horizon:

$$z_m = 0 \tag{3.3}$$

- Each node $i \in \mathcal{N}$ initially possesses a subset $\mathcal{O}_i$ of datum units:

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \in \mathcal{O}_i,\ y_{i,k,0} = 1 \tag{3.4}$$

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \notin \mathcal{O}_i,\ y_{i,k,0} = 0 \tag{3.5}$$

- The transfer plan must be valid (sending nodes must possess the datum units that they transfer):

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ x_{k,c} \leq y_{s_c,k,\mathcal{T}_i(c-1)} \tag{3.6}$$

- Nodes possess a datum unit from the time they receive it:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, \ldots, m\},\ y_{r_c,k,c} \leq y_{r_c,k,\mathcal{T}_i(c-1)} + x_{k,c} \tag{3.7}$$

$$\forall k \in \mathcal{D},\ \forall c \in \{1, \ldots, m\},\ y_{r_c,k,c} \geq 1/2 \left[y_{r_c,k,\mathcal{T}_i(c-1)} + x_{k,c} \right] \tag{3.8}$$

- At most one datum unit can be transferred during each contact:

$$\forall c \in \{1, \ldots, m\},\ \sum_{k \in \mathcal{D}} x_{k,c} \leq 1 \tag{3.9}$$

3.4.2 Additional constraints

Together, constraints (3.7) and (3.8) ensure that variable $y_{r_c,k,c}$ is equal to 1 if $y_{r_c,k,\mathcal{T}_i(c-1)} = 1$ or $x_{k,c} = 1$, and to 0 otherwise – *i.e.* node r_c possesses unit k after contact σ_c if it already possessed k after contact σ_{c-1}, or if it received the unit during contact σ_c. Yet, if we seek a minimal transfer plan, as both conditions cannot be true at the same time, constraints (3.7) and (3.8) can be replaced by the following constraint:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ y_{r_c,k,\mathcal{T}_i(c-1)} + x_{k,c} = y_{r_c,k,c} \tag{3.10}$$

It is possible to strengthen the model such that transfer plans are active, but this unfortunately leads to complex equations and poor numerical results in practice. However, it is quite easy to guarantee that solutions are strictly-active. This can be achieved by ensuring that transfer plans are minimal and that any transfer $\phi(c)$ cannot be null if the sending node s_c possesses at least one datum unit that the receiving node r_c does not possess (since $\phi(c) \neq \emptyset$ and $O^c_{r_c} > O^{c-1}_{r_c}$ are equivalent in a minimal transfer plan), *i.e.*

$$\forall k \in \mathcal{D},\ \forall c \in \{1, \ldots, m\},\ y_{s_c,k,\mathcal{T}_{s_c}(c-1)} - y_{r_c,k,\mathcal{T}_{r_c}(c-1)} \leq \sum_{k' \in \mathcal{D}} x_{k',c} \tag{3.11}$$

Altogether, the model contains $1 + un + m + 4um$ constraints – including constraints (3.2) through (3.11) (except constraints (3.2-a) through (3.2-c) and constraints (3.7)+(3.8)).

Below we discuss the effectiveness of preprocessing procedures, *cf.* Section 3.3, and we compare the different models proposed in this Section.

3.5 Computational results

In this section, experimental results are reported and discussed. First of all, in Section 3.5.1, the random process used to generate the benchmark will be described. Instance classes are built with respect to the number of nodes n and the number of datum units u (the number of contacts m is chosen such that most of the instances are feasible). The models described in Section 3.4 are assessed in Section 3.5.2, while the preprocessing procedures proposed in Sections 3.2 and 3.3 are discussed in Section 3.5.3.

3.5.1 About the benchmarks

In this section, we describe a destruction/construction algorithm to generate difficult instances for the dissemination problem. The instances are available on our website [28].

Random instances are generally easy to solve. That is why we developed a procedure which attempts to increase the "hardness" of an existing instance by iteratively renewing the least relevant contacts (the contacts removed by a trivial preprocessing procedure). Usually starting with a random instance, the process is intended to generate harder and harder instances by removing irrelevant contacts, in such way that new ones can be added elsewhere in the sequence, without increasing the size of the problem (specifically the number of contacts). Note that a few random contacts are also renewed to introduce more diversity in the instances visited during the process.

At each iteration, the instance is solved by a given solver (in practice we can use the branch-and-cut algorithm defined in Section 3.4.1). After a given number of iterations, the instance that has required the most CPU-time to be solved is returned (this is assumed to be the most "difficult" instance).

The generated instances were classified by difficulty, with respect to the number of nodes n and the number of datum units u. The number of contacts m was chosen accordingly. Next the instances were solved using a variety of solvers, *e.g.* a MIP-based solver with and without a preprocessing procedure, and the union of the twenty hardest instances obtained for each solver was retained (except that the instances which could not be solved by at least one of these solvers were ignored).

184 *instances* were selected in this way, resulting in eight *classes* (named **3u10n**, **4u20n**, **4u50n**, **4u100n**, **5u50n**, **10u10n**, **50u10n**, and **100u10n**). These classes are described by the number *nbinst* of instances they contain, the number u of datum units and the number n of nodes characterizing these instances. The average number $\overline{rec}$ of recipients $i \in \mathcal{R}$, the average number $\overline{src}$ of sources $i \in \mathcal{N} \mid \mathcal{O}_i \neq \emptyset$, and the average number $\overline{m}$ of contacts of these instances are reported below. Thereafter, the eight classes were *grouped* by difficulty. The first group contains the easiest instances. The second and the third group contain the instances having many nodes but few datum units, or few nodes but many datum units, respectively.

	name	*nbinst*	u	n	$\overline{rec}$	$\overline{src}$	$\overline{m}$
1	**3u10n**	36	3	10	10	1	135
	4u20n	41	4	20	18	1	366
2	**4u50n**	26	4	50	39	1	710
	4u100n	20	4	100	87	2	1720
	5u50n	23	5	50	50	1	726
3	**10u10n**	16	10	10	6	2	197
	50u10n	16	50	10	6	2	750
	100u10n	6	100	10	7	4	2000

In the following subsections, we provide and discuss some computational results that show the efficiency of our algorithms. These computations were performed on a server equipped with 16×6 cores (each running at 2.67Ghz) and 1TB RAM. All the algorithms are implemented in `C++`. MILP are solved by CPLEX, the commercial solver developed by IBM-ILOG. Multithreading features proposed by this library are deactivated, since the use of concurrent optimizers led to unstable results. More specifically, several executions of the same computation sometimes led to different results (in terms of CPU time), which prevents us making reliable analyses. Consequently, all instances were solved by an *exact*, *sequential*, and *deterministic* algorithm, within a one-hour time-limit.

The same information is to be found in all tables of results, namely:

1. *solved* (-%) indicates the ratio of instances solved by the solver.
2. *feas* (-%) indicates the ratio of instances that remained unsolved but for which the solver found at least one feasible solution within the one-hour time-limit.

   Therefore $100\% - \textit{solved} - \textit{feas}$ (-%) indicates the ratio of instances for which the solver neither found a feasible transfer plan, nor showed the instance to be infeasible.
3. *gap* (-%) is the average relative gap between *the best lower bound* and *the best feasible solution* which have been computed during the search (this metric concerns only the *feas*% of the instances which remained unsolved but for which the solver found at least one feasible solution).

4. *cpu* (-s) indicates the average solving time for the given solver, including all instances and thus all time-limits. *cpu* therefore tends to 3600s when *solved* tends to 0.0%.

Remark 3.9

In each section, different sets of parameters are compared, and the best strategies selected. In our opinion, such a selection should be made with respect to the three *groups* rather than the eight *classes*. In this way the benchmark cannot be learned by heart.

3.5.2 About the models

Models – The most trivial model is given in Section 3.4.1. This MILP-based model is defined by equations (3.1) through (3.9), and is labelled **std** in the following. To decide which constraints from among (3.2), (3.2-a), (3.2-b) and (3.2-c) are most suitable, we compare the four cases, *cf.* Table 3.1.

There is no clear dominance between these four models. In every group, one notes that the best results per class are obtained with a *different* model (the impact of each constraint appears to be random at first glance). However we decided to use constraint (3.2-a) in the following, because it leads to the best results on average, *i.e.* across all classes.

Dominance rules – Model **std** can be refined. If transfer plans are required to be minimal, constraints (3.7)+(3.8) can be replaced by constraint (3.10), giving rise to a new model denoted as **min**. If transfer plans are required to be strictly-active too, constraint (3.11) can be added to model **min**. It gives rise to a third model, termed **min+st/act**. The numerical results obtained with these two models are reported in Table 3.2.

Model **min+st/act** is seen to outperform both **std** and **min** in the first group, while model **min** dominates the two others on bigger instances. None of them, however, gives convincing results, and limits appear as the number of nodes and, more importantly, the number of units increases.

As we will confirm below, the difficulty of an instance is better explained by the number of datum units than by the number of nodes. With a solving rate greater than 90% for all solvers, the instances in group 2 (characterized by many systems but few units) seem manageable whereas, instances of the third group (few systems but many units) seem out of reach for the moment. This being said, we could state that **min** is the best model out of the three,

because it is the only one that successfully solves some of the most difficult instances. However, this model's disappointing results when dealing with the easier classes (namely **3u10n** and **4u20n**), make us reluctant to make such a categorical statement.

These three options (**std**, **min** and **min+st/act** with constraint (3.2-a)) will therefore continue to be considered in the next section. The superiority of model **min** will then be clearly established.

3.5.3 About the preprocessing procedures

The main conclusion of the previous section is probably that standard solvers show their limits quite soon for this kind of problem.

Fortunately the deduction algorithms proposed in Section 3.3 significantly improve efficiency of solvers when applied within a preprocessing procedure whose role is to remove useless contacts or, conversely, to detect unavoidable transfers. To show this, we will now investigate different procedures that are based on Algorithm 3.6, *cf.* Section 3.3.2.

Let us recall that Algorithm 3.6 is based on a model known as the *transfer graph* that encapsulates knowledge about each transfer $\phi(c)$, $c \in \{1, \ldots, m\}$, and each state O_i^t, $i \in \mathcal{N}$, $t \in \{0, \ldots, m\}$. We are then able to apply different dominance-rule-based propagation algorithms to deduce new knowledge, and in particular to reveal the transfers that are always null (or improving) in a minimal strictly-active transfer plan. The routine processes all transfers $\phi(c)$, $c \in \{1, 2, \ldots, m\}$. First, it calls Bottom-Up and Top-Down to determine which units possessed by node s_c have been received by node r_c. Thereafter procedure Minimality+Validity uses the eponymous dominance rules to avoid fruitless transfers. Strict-Activity and Delivery-Requirements also detect transfers that are necessary, or impossible. The knowledge, thus enriched, can finally be passed on to the mathematical model. Some variables can be fixed at the root node – *e.g.* if transfer $\phi(c)$ is shown to be null in all dominant solutions, then $\forall k \in \mathcal{D}$, $x_{k,c} = 0$.

As usual, a tradeoff has to be found between the time spent running the preprocessing procedure, and the time saved in the branch-and-cut procedure as a result of the preprocessing. Actually, this is particularly true for routines Top-Down and Strict-Activity, since both often consume a significant amount of time computing *min-card* (*cf.* Section 3.3.3). To empirically find the compromise, we propose five more or less aggressive strategies, which are labelled **minimal**, **light**, **normal**, **aggressive** and **maximal**. The strategies

		std // constraint (3.2)				**std // constraint** (3.2-a)			
	name	*solved*	*feas*	*cpu*	*gap*	*solved*	*feas*	*cpu*	*gap*
1	**3u10n**	100	-	0.80	-	100	-	0.64	-
	4u20n	100	-	9.9	-	100	-	12.3	-
2	**4u50n**	100	-	39.7	-	100	-	49.6	-
	4u100n	100	-	148	-	100	-	224	-
	5u50n	91.3	8.7	520	9.2	95.7	4.3	332	12.6
3	**10u10n**	50.0	25.0	2362	21.8	68.8	12.5	1614	22.2
	50u10n	0.00	12.5	3587	2.7	0.00	6.3	3585	1.1
	100u10n	0.00	0.00	3593	-	0.00	0.00	3592	-
	avg	82.6	4.3	724	13.8	84.8	2.2	645	14.5
		std // constraint (3.2-b)				**std // constraint** (3.2-c)			
	name	*solved*	*feas*	*cpu*	*gap*	*solved*	*feas*	*cpu*	*gap*
1	**3u10n**	100	-	1.1	-	100	-	0.96	-
	4u20n	100	-	17.3	-	100	-	19.6	-
2	**4u50n**	100	-	84.1	-	100	-	50.8	-
	4u100n	100	-	110	-	100	-	165	-
	5u50n	95.7	4.3	398	14.5	95.7	4.3	392	12.5
3	**10u10n**	50.0	25.0	2087	16.5	56.3	18.8	1969	17.2
	50u10n	0.00	18.8	3586	1.8	0.00	18.8	3586	8.2
	100u10n	0.00	0.00	3589	-	0.00	0.00	3589	-
	avg	83.2	4.3	688	10.7	83.7	3.8	679	12.7

Table 3.1 – Computational results achieved with CPLEX and different models.

		min // constraint (3.2-a)				**min+st/act // const.** (3.2-a)			
	name	*solved*	*feas*	*cpu*	*gap*	*solved*	*feas*	*cpu*	*gap*
1	**3u10n**	100	-	0.91	-	100	-	0.30	-
	4u20n	100	-	14.1	-	100	-	4.2	-
2	**4u50n**	100	-	30.2	-	100	-	50.7	-
	4u100n	95.0	5.0	240	4.6	90.0	10.0	919	10.6
	5u50n	95.7	4.3	266	14.0	95.7	4.3	696	3.3
3	**10u10n**	81.3	12.5	1317	20.1	93.8	6.3	706	10.2
	50u10n	56.3	18.8	2563	2.6	0.00	6.3	3591	16.5
	100u10n	33.3	16.7	3116	0.28	0.00	0.00	3597	-

Table 3.2 – Computational results achieved with different models.

are built with the intuitive idea that consistency algorithms are more efficient on earlier contacts than on later ones, because *min-card* problems involving few transfers are much smaller. Therefore we just vary the number of contacts after which the most computationally costly routines are skipped.

These strategies are characterized by the maximum amount *pre.tl* of time which can be spent in the preprocessing procedures, the ranges *td.range* and *sa.range* of contacts to which routines Top-Down and Strict-Activity will respectively be applied, and the maximum number *dr.limit* of times the while-loop of routine Delivery-Requirements can be run. The functions evaluating *min-card* and *max-card* are limited in time by parameters *minc.tl* and *maxc.tl*, respectively. Bottom-Up and Minimality+Validity can be used without any restriction.

The five strategies are described below. Note that $td.range = 0.25$ means that top-down deductive reasoning is applied to the first quarter of transfers (from $c = 0$ to $c = \lfloor 0.25 \times m \rfloor$). The experimental results obtained using the different strategies are reported in Tables 3.3 through 3.7. The two columns *rem* and *fcd* (-%) indicate the average percentages of contacts that have been *removed* (shown to be null in all dominant solution), or alternatively *forced* (compelled to be improving or even fixed). In this way *rem* and *fcd* measure the efficiency of the procedures. They must be considered together with *prep*, the average amount of time required to execute the procedures.

	minimal	**light**	**normal**	**aggr.**	**maximal**
pre.tl (-s)	25	25	25	600	2400
do.bottom.up	yes	yes	yes	yes	yes
do.minimality	yes	yes	yes	yes	yes
td.range	-	0.50	1.00	1.00	1.00
sa.range	-	0.15	0.50	1.00	1.00
dr.limit	2	2	2	4	4
minc.tl	-	0.06s	0.06s	0.10s	0.10s
maxc.tl	-	0.20s	0.20s	0.30s	0.30s

		standard		min		min+st/act		efficiency		
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*prep*	*rem*	*fcd*
1	**3u10n**	100	0.70	100	0.78	100	0.26	0.00	15.2	-
	4u20n	100	8.6	100	11.3	100	3.3	0.01	4.9	-
2	**4u50n**	100	31.3	100	33.5	100	46.5	0.02	5.6	-
	4u100n	95.0	237	95.0	230	95.0	600	0.08	5.1	-
	5u50n	100	270	95.7	270	95.7	433	0.02	2.3	-
3	**10u10n**	62.5	1819	87.5	997	81.3	956	0.00	9.8	-
	50u10n	0.00	3589	56.3	2343	0.00	3593	0.05	5.5	-
	100u10n	0.00	3590	33.3	3208	0.00	3591	0.29	1.9	-

Table 3.3 – Computational results achieved using **minimal** preprocessings.

		standard		min		min+st/act		efficiency		
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*prep*	*rem*	*fcd*
1	**3u10n**	100	0.48	100	0.47	100	0.52	0.42	49.0	5.3
	4u20n	100	2.2	100	2.0	100	2.6	1.4	26.9	6.7
2	**4u50n**	100	9.0	100	4.1	100	17.3	2.5	21.0	6.7
	4u100n	100	32.0	100	20.1	100	275	5.7	20.3	5.6
	5u50n	100	19.6	100	19.4	100	107	2.7	13.1	7.4
3	**10u10n**	75.0	1184	93.8	482	93.8	546	0.62	19.0	6.4
	50u10n	6.3	3523	68.8	1647	0.00	3583	5.0	8.8	3.8
	100u10n	0.00	3594	33.3	3112	0.00	3591	25.3	0.76	1.3

Table 3.4 – Computational results achieved using **light** preprocessings.

		standard		min		min+st/act		efficiency		
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*prep*	*rem*	*fcd*
1	**3u10n**	100	1.8	100	1.9	100	2.0	1.8	62.8	8.9
	4u20n	100	6.5	100	6.1	100	6.8	5.8	34.3	10.6
2	**4u50n**	100	16.7	100	10.8	100	24.9	9.8	23.6	11.2
	4u100n	100	60.0	100	35.3	100	271	21.7	20.3	9.4
	5u50n	100	39.0	100	15.5	100	171	11.5	13.3	12.0
3	**10u10n**	81.3	1017	100	107	100	313	3.5	14.4	12.3
	50u10n	0.00	3590	62.5	1721	0.00	3581	25.1	4.9	5.9
	100u10n	0.00	3590	16.7	3267	0.00	3587	25.3	0.76	1.3

Table 3.5 – Computational results achieved using **normal** preprocessings.

		standard		min		min+st/act		efficiency		
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*prep*	*rem*	*fcd*
1	**3u10n**	100	3.0	100	3.1	100	3.2	3.0	64.9	12.1
	4u20n	100	17.0	100	17.1	100	17.7	16.4	37.9	11.4
2	**4u50n**	100	42.2	100	36.2	100	49.0	35.0	24.9	12.2
	4u100n	100	170	100	137	100	322	128	21.6	9.8
	5u50n	100	74.5	100	54.2	100	173	48.8	13.4	12.5
3	**10u10n**	75.0	1020	93.8	464	100	277	12.9	14.6	15.5
	50u10n	0.00	3589	68.8	1691	0.00	3584	156	8.8	8.9
	100u10n	0.00	3592	66.7	3305	0.00	3591	601	1.6	1.8

Table 3.6 – Computational results achieved using **aggressive** preprocessings.

		standard		min		min+st/act		efficiency		
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*prep*	*rem*	*fcd*
1	**3u10n**	100	3.5	100	3.5	100	3.4	3.5	65.1	12.1
	4u20n	100	25.6	100	25.0	100	25.4	25.0	40.2	11.4
2	**4u50n**	100	64.1	100	58.3	100	66.9	57.1	25.9	12.2
	4u100n	100	233	100	215	100	319	206	21.9	9.8
	5u50n	100	109	100	87.1	100	196	84.0	13.4	12.5
3	**10u10n**	75.0	1029	93.8	533	100	337	19.7	14.7	15.5
	50u10n	0.00	3593	68.8	1724	0.00	3590	195	8.8	8.9
	100u10n	0.00	3600	0.00	3597	0.00	3600	2402	2.0	1.8

Table 3.7 – Computational results achieved using **maximal** preprocessings.

		selected		computational results				performances		
	name	*model*	*prep.*	*solved*	*feas*	*cpu*	*gap*	Δ_{rem}	Δ_{fcd}	Δ_{prep}
1	**3u10n**	min	light	100	-	0.47	-	75.3	44.0	11.9
	4u20n	min	light	100	-	2.0	-	66.9	58.1	5.6
2	**4u50n**	min	light	100	-	4.1	-	81.2	54.6	4.4
	4u100n	min	light	100	-	20.1	-	92.7	57.1	2.8
	5u50n	min	light	100	-	19.4	-	97.8	59.0	3.2
3	**10u10n**	min	agg.	93.8	6.3	464	10.17	99.6	100	65.6
	50u10n	min	agg.	68.8	6.3	1691	1.06	100	99.9	79.7
	100u10n	min	agg.	66.7	0.00	3305	-	81.1	97.7	25.0

$\Delta_{rem} = {}^{rem(prep.)}/_{rem(max.)}$ (×100) • $\Delta_{fcd} = {}^{fcd(prep.)}/_{fcd(max.)}$ (×100) • $\Delta_{prep} = {}^{prep(prep.)}/_{prep(max.)}$ (×100)

Table 3.8 – The best preprocessing strategies.

Remark 3.10

The most aggressive strategy – **maximal** – is characterized by a specific propagation algorithm, which is executed before procedure Bottom-Up for all contact indexes, that is at the beginning of Algorithm 3.6's main loop. It attempts to prove that node r_c has necessarily obtained all the units (the whole datum) in accordance with Proposition 3.9, by testing whether $\textit{min-card}(V_D, r_c, c-1) = u$. If so – combined with Bottom-Up and Minimality+Validity – this results in the deletion of contact σ_c. Although this procedure is inefficient in practice, we retained it as part of **maximal**, because it is only designed to upper-bound the number of deductions that can be done with our approach.

Minimal – In this strategy, only the most basic procedures are run, *e.g.* no transfer can be forced (since Strict-Activity is deactivated). The number of null transfers detected is low. Although the amount *prep* of time required to preprocess the instance is insignificant, and almost linear with respect to the number of contacts, this strategy does not improve the behaviour of the solver (the results reported in Tables 3.1/3.2 and 3.3 are similar). In fact we may suppose that CPLEX's preprocessing engine already makes such kinds of deductions from the MILP model.

Light/Normal/Aggressive – The effects of the preprocessing procedures become significant when Top-Down and Strict-Activity are acting, *e.g.* if we look at the first class in Table 3.4, almost half of the contacts have been removed and about 5% of the transfers have been forced in only 0.4 seconds. The efficiency of the preprocessing procedures is remarkable on the first two groups, but unfortunately collapses when the number of datum units is large (*rem* and *fcd* drop when u is greater than 4 or 5).

This behaviour may be explained by the fact that both Top-Down and Strict-Activity need strongly NP-Hard problems (evaluating *min-card*) solved (which is far from straightforward). These problems are characterized by fewer contacts, some datum units, and only one recipient. In fact, we try to solve smaller problems in order to deduce information about the original master problem. Unfortunately, these problems are much more difficult when the number of units is larger (for all transfers, there are too many options), and the CPU time required to run preprocessing procedures explodes – *e.g.* see column *prep* in Table 3.7.

Maximal – This strategy is really *much* too aggressive. The computational overhead of the overall preprocessing procedure is unjustified. This remark is highlighted in Table 3.8, where the experimental results of the best strategies are summarised. In the three last columns are given different ratios between these strategies and **maximal**. Thus, if we consider class **5u50n**, almost all contacts removed (resp. about 60% of the contacts forced) by **maximal**, are also removed (resp. forced) by strategy **light**, with a computational cost that is around 30 times less. So strategy **maximal** should not be used.

Note in addition that model **min** is clearly dominant when preprocessing procedures are used – except for **10u10n** which is better solved with model **min+st/act**.

3.6 Conclusion

First it will be remarked that the dissemination problem can be solved more efficiently by using the deductive elements proposed in Section 3.3. We have shown how these can be used within a preprocessing procedure designed to simplify the instances. Given the positive results achieved, we also attempted to apply these deduction rules dynamically, by propagating the constraints during the branching stage, as a constraints-programming engine would do. However, this gave poor results that are not worth reporting here.

Such an approach falls within the well-known framework called constraint propagation, a powerful tool of *constraint programming* that is unsuitable for linear programming. Implementing Algorithm 3.6 with linear programming seemed unnatural and excessively complex. Typically, matching the variables of our model and the properties of the transfer graph (χ_A, χ_ϕ, $\chi_{\underline{D}}$ and $\chi_{\overline{D}}$) is not at all a straightforward matter. This prevents us from making full use of the information obtained using the deduction procedures in the model – *e.g.* what kind of relationships is it that binds the arcs of the transfer graph (or more specifically function χ_A) and the x-variables ?

In addition, while the notion of *minimal transfer plan* is seen to be well integrated into the model (through constraint (3.10)), the concept of *strictly-active transfer plan* continues to yield disappointing results. To tackle these problems, we now propose a new constraint-programming-based model.

4 Constraint programming

Integer linear programming has been seen to be an unsuitable framework for taking full advantage of the dominance rules proposed in Chapter 3. In particular, formulating the concept of strictly-active transfer plan in the linear program has yielded mitigated numerical results (as opposed to the notion of minimal transfer plan, which has already produced promising results). To address this issue, we propose to use *constraint programming* in hope that constraint propagation mechanisms will enable every dominance rule to be leveraged more effectively. First we will propose a new program modelling the dissemination problem, and then a branching algorithm for obtaining a solution (*cf.* Section 4.1). Next we will propose additional features to refine this procedure (*cf.* Section 4.2).

Contents

4.1 Modelling the dissemination problem 98
4.1.1 Constraint programming model 98
4.1.2 Branching algorithm 100
4.2 Additional features 102
4.2.1 Lower bounds 102
4.2.2 Symmetry-breaking techniques 104
4.3 Computational results 112
4.3.1 About the model 112
4.3.2 The additional features 114
4.4 Conclusion 117

4.1 Modelling the dissemination problem

In this section, we first propose a constraint programming model for solving the dissemination problem, and then a branching algorithm for finding valid transfer plans (this procedure will be refined in Section 4.2).

4.1.1 Constraint programming model

The model proposed below is directly inspired by the linear model described in Section 3.4. However, since non-linear expressions can now be used, many constraints will be significantly simplified.

As a reminder, for each node $i \in \mathcal{N}$ is defined $\mathcal{T}_i$, the set of time indexes at which the state of i can change, *i.e.* $\mathcal{T}_i = \{0\} \cup \{c \in \{1, 2, \ldots, m\} \mid r_c = i\}$. In addition, $\forall t \in \{0, 1, \ldots, m\}$, we refer by $\mathcal{T}_i(t)$ to the last contact occurring before time t where node i is the receiver, *i.e.* $\mathcal{T}_i(t) = \max\{t' \in \mathcal{T}_i \mid t' \leq t\}$.

The variables are defined as follows:

- $\forall k \in \mathcal{D}$, $\forall c \in \{1, \ldots, m\}$, $x_{k,c} = 1$ if datum unit k is transmitted from node s_c to node r_c during contact σ_c, and $x_{k,c} = 0$ otherwise.
- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, and $\forall t \in \mathcal{T}_i$, $y_{i,k,t} = 1$ if node i possesses unit k after contact σ_t, and $y_{i,k,t} = 0$ otherwise. See Remark 3.8 on page 82.

- $\forall c \in \{1, 2, \ldots, m\}$, $a_c = 1$ if transfer $\phi(c)$ is improving (something new is transmitted to node r_c), and $a_c = 0$ otherwise.
- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, variable $\lambda_{i,k}$ represents the delivery length associated with datum unit k and node i – *i.e.* the date from which node i stores datum unit k. Its domain is $\mathcal{T}_i \cup \{\infty\}$. Note that $\lambda_{i,k} = \infty$ means that node i does not recover unit k during the transfer plan. In practice we can take $\infty = m + 1$.
- $\forall i \in \mathcal{R}$, variable $\lambda_i = \max_{k \in \mathcal{D}} \{\lambda_{i,k}\}$ then represents the delivery length of recipient node i – *i.e.* the date from which node i stores all the datum units. Its domain is $\mathcal{T}_i$ (because i has to be served).
- Finally, variable $\lambda = \max_{i \in \mathcal{R}} \{\lambda_i\}$ represents the dissemination length of the transfer plan. Its domain is therefore $\bigcup_{i \in \mathcal{R}} \mathcal{T}_i \subseteq \{0, 1, \ldots, m\}$.

Minimizing the dissemination length leads to the following objective.

$$\lambda^* = \min \lambda \tag{4.1}$$

The constraints are written as follows:

- Each node $i \in \mathcal{N}$ initially possesses a subset $\mathcal{O}_i$ of datum units:

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \in \mathcal{O}_i,\ y_{i,k,0} = 1 \tag{3.4}$$
$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \notin \mathcal{O}_i,\ y_{i,k,0} = 0 \tag{3.5}$$

- The transfer plan must be valid (sending nodes must possess the datum units that they transmit):

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ x_{k,c} \leq y_{s_c,k,\mathcal{T}_i(c-1)} \tag{3.6}$$

- Nodes possess a datum unit from the time they receive it:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ y_{r_c,k,\mathcal{T}_i(c-1)} + x_{k,c} = y_{r_c,k,c} \tag{3.10}$$

- At most one datum unit can be transferred during each contact:

$$\forall c \in \{1, 2, \ldots, m\},\ \sum_{k \in \mathcal{D}} x_{k,c} = a_c \tag{4.2}$$

- $\lambda_{i,k}$ is the delivery length associated with datum unit k and node i:

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D},\ \forall t \in \mathcal{T}_i,\ \begin{cases} y_{i,k,t} = 0 \iff \lambda_{i,k} > t \\ y_{i,k,t} = 1 \iff \lambda_{i,k} \leq t \end{cases} \tag{4.3}$$

- The transfer plan must be strictly-active:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},$$
$$\textbf{if } [\, y_{s_c,k,\mathcal{T}_{s_c}(c-1)} > y_{r_c,k,\mathcal{T}_{r_c}(c-1)} \,] \textbf{ then } [\, a_c = 1 \,] \tag{4.4}$$

Remark 4.1

As a reminder, constraint (3.10) ensures that node r_c possesses a datum unit k after contact σ_c if it possessed k after contact σ_{c-1}, or if it received k during contact σ_c. Nevertheless, the constraint prevents both of these conditions being true at the same time, and *so ensures that the solution is minimal.*

Altogether, the model thus contains um x-variables, $u.(2m + n)$ y-variables, $nu + |\mathcal{R}| + 1$ λ-variables, m a-variables, and $un + m + 5um$ constraints.

The model is solved by a branch-and-bound procedure. The efficiency of such an algorithm is heavily dependent on the "branching variables" that are selected, as well as on the quality of the bounds computed at every node of the search tree. For this reason we propose a branching heuristic in the next section. Lower bounds for the problem will be introduced in Section 2.3.

4.1.2 Branching algorithm

In order to prune branches as early as possible and obtain a fast convergence, it is necessary to find good upper bounds quickly. To this end, priority should be given to the most promising branches, so that the better solutions (whose values are upper bounds) are visited sooner. This is why we propose setting transfers *sequentially* and *heuristically*, *i.e.* the x-variables are set from $x_{k,1}$ ($\forall k \in \mathcal{D}$) to $x_{k,m}$. This also makes it easy to develop *ad hoc* procedures that reduce the size of enumerations (*cf.* Sections 4.2.1 and 4.2.2).

In practical terms, at each node of the search tree, the solver selects the smallest index $c \in \{1, 2, \ldots, m\}$ for which the value of transfer $\phi(c)$ has not yet been decided, then creates one branch per possible value – *i.e.* it creates one branch for each unit $k \in \mathcal{D}$ such that variable $x_{k,c}$ is not fixed, then sets $x_{k,c} = 1$ and $x_{k',c} = 0$ for all $k' \in \mathcal{D}\backslash\{k\}$. If variable a_c is not fixed to 1, then an extra branch is created for the null transfer. To this end the whole set of $x_{k,c}$ variables ($\forall k \in \mathcal{D}$) is set to 0.

Therefore, except for the branch corresponding to the null transfer, each branch is associated with exactly one datum unit $k \in \mathcal{D}$, and can be referred to as B_k unambiguously.

The order in which these branches are visited is heuristic (and the result of many empirical tests conducted by A. Groud, L. Leichtnam, and myself).

1. First of all, we seek to identify the most "critical" transfers in terms of *feasibility*. In particular – as the fewer the remaining opportunities for node r_c to receive a datum unit $k \in \mathcal{D}$, the more urgent the transfer of k becomes – we give priority to the branches B_k for which criterion

$$|\{\, t \in \{c, \ldots, m\} \text{ such that } r_t = r_c \text{ and } x_{k,t} \text{ is not set to } 0 \,\}|$$

   is the lowest.

2. *In case of a tie*, we seek to balance the dissemination of all the datum units. Therefore we prioritize the branches B_k whose criterion

$$|\{\, t \in \{1, 2, \ldots, m\} \text{ such that } r_t \neq r_c \text{ and } x_{k,t} \text{ is not set to } 0 \,\}|$$

   is the highest. This criterion is actually based on the heuristic proposed by Belblidia *et al.* in [5].

3. If there is one, the branch corresponding to the null transfer is always considered as a last resort.

In practice, this heuristic has given promising results. It unfortunately fails on some instances of medium and large size. When the number of backtracks starts to soar, we therefore switch to the solver's built-in algorithm[1] and start the search again from scratch. The default algorithm seems better to prove the optimality of a solution – *i.e.* to "close" the nodes in the search tree.

These failures occur because pure depth-first searches (like ours) generally yield an initial solution quite fast – but do not move easily from one area of the search space to another, since they are often unable to recover sufficiently quickly from bad decisions made early on in the process. For this reason we devote the following sections to *ad hoc* methods designed to counterbalance this familiar drawback of depth-first searches. Note that the implementation of these *ad hoc* methods is significantly facilitated by the fact that transfers are set sequentially.

[1] The behaviour of this algorithm will not be described in the present chapter (see the online documentation of the IBM-Ilog CP Optimizer for more details).

4.2 Additional features

In this section, we propose additional features that can easily be integrated into the branching algorithm proposed in Section 4.1.2. We first propose two lower bounds for the problem, then we describe three techniques for breaking the symmetries that are inherent in our approach.

4.2.1 Lower bounds

In this subsection we propose some lower bounds of the delivery length $\lambda_i(\phi)$ corresponding to the different recipient nodes $i \in \mathcal{R}$, and consequently some lower bounds of the dissemination length $\lambda(\phi)$ of a transfer plan ϕ.

Let us first consider the following proposition.

Proposition 4.1

Let $i \in \mathcal{R}$ be a recipient node and $\alpha = u - |\mathcal{O}_i|$ be the number of datum units that node i has to obtain during a transfer plan. Let $\sigma_x \in \sigma$ be the α^{th} contact $\sigma_c = (s_c, i) \in \sigma$ during which a datum unit $k \in \mathcal{D}\backslash\mathcal{O}_i$ can be transferred to node i – *i.e.* such that variable $x_{k,c}$ is not set to 0. If this index does not exist (*e.g.* if it remains less than α contacts fulfilling the condition), we consider that $x = \infty$. x is a valid lower bound for $\lambda_i(\phi)$ and $\lambda(\phi)$ – *i.e.* $\lambda \geq \lambda_i \geq x$.

This lower bound can be enhanced by checking whether a valid assignment between available transfers and required datum units exists. This point may be illustrated by considering a recipient node $i \in \mathcal{R}$ that must receive units 1, 2, 3 ($\alpha = 3$) and for which the following contacts are available:

	domains			
σ_c	$x_{1,c}$	$x_{2,c}$	$x_{3,c}$	
$\sigma_3 = (s_3, i)$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\rightarrow \phi(3) \in \{\emptyset, \{1\}, \{2\}, \{3\}\}$
$\sigma_5 = (s_5, i)$	$\{0,1\}$	$\{0\}$	$\{0\}$	$\rightarrow \phi(5) \in \{\emptyset, \{1\}\}$
$\sigma_8 = (s_8, i)$	$\{0,1\}$	$\{0\}$	$\{0\}$	$\rightarrow \phi(8) \in \{\emptyset, \{1\}\}$
$\sigma_9 = (s_9, i)$	$\{0\}$	$\{0,1\}$	$\{0,1\}$	$\rightarrow \phi(9) \in \{\emptyset, \{2\}, \{3\}\}$

Any contact σ_c, $c \in \{3, 5, 8, 9\}$, is such that there exists a unit $k \in \mathcal{D}\backslash\mathcal{O}_i$ for which variable $x_{k,c}$ is not set to 0. Thus Proposition 4.1 states that delivery length $\lambda_i(\phi) \geq 8$. However, it is clear that there is no transfer plan enabling node i to receive datum units 1, 2 and 3 using only contacts σ_3, σ_5 and σ_8.

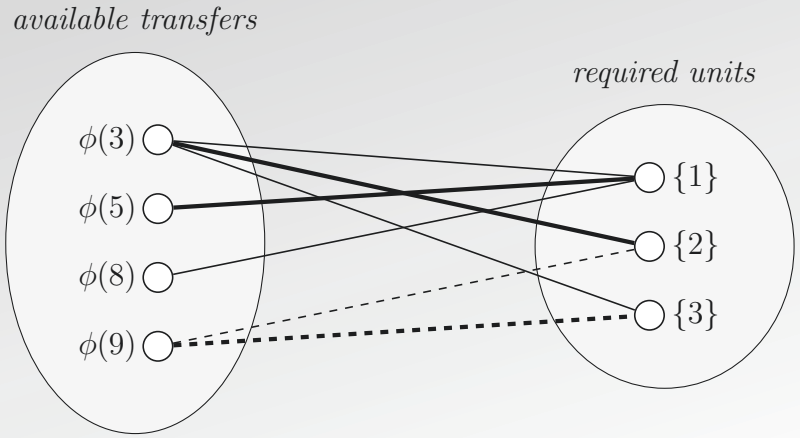


Figure 4.1 – The assignment problem that must be solved to compute the lower bound.

In the best case, node i receives one unit from among $\{2, 3\}$ during contact σ_3 and unit 1 during contact σ_5 or σ_8. Even so, node i still has to receive at least one datum unit from among $\{2, 3\}$ during contact σ_9. In fact, delivery length $\lambda_i(\phi)$ is necessarily greater than or equal to 9, since there is no valid assignment of transfers $\phi(3)$, $\phi(5)$ and $\phi(8)$ enabling node i to receive datum units $\{1, 2, 3\}$, *cf.* Figure 4.1. Hence the following proposition.

Proposition 4.2

Let $i \in \mathcal{R}$ be a recipient, and $\sigma^i \subseteq \sigma$ be the subsequence of contacts that is built by considering only contacts $\sigma_c = (s_c, i) \in \sigma$ during which a unit $k \in \mathcal{D} \backslash \mathcal{O}_i$ can be transferred to node i (such that variable $x_{k,c}$ is not yet fixed to 0). Let $\sigma_x \in \sigma^i$ denote the first contact from which – by adding the contacts of σ^i one after the other – there exists an assignment of the transfers which leads to the delivery of all the units to node i. If such an index x does not exist, then we consider that $x = \infty$. x is a lower bound for $\lambda_i(\phi)$ and $\lambda(\phi)$, *i.e.* $\lambda \geq \lambda_i \geq x$.

The computational investment required to compute this lower bound gives a particularly good return when the number of datum units is high, but the weaker bound defined in Proposition 4.1 usually gives good results too – *cf.* Section 4.3 for a comprehensive comparison between these lower bounds.

Another lower bound can be derived from the proof of polynomiality that we proposed in Section 2.2.2 for the case – called the delivery problem – where there is only one recipient node, *i.e.* $|\mathcal{R}| = 1$. Given a recipient node $i \in \mathcal{R}$, the optimal solution which can be computed for the problem where node i is the unique recipient, is a lower bound for delivery length $\lambda_i(\phi)$ in the original problem. Unfortunately this lower bound is weak in practice. Transforming the problem into a flow problem implies relaxing the "identity" of the datum units (a unit of flow is not associated with a specific unit). This prevents us using information collected during the search (*e.g.* the values of the variables) to constrain the admissible flows.

4.2.2 Symmetry-breaking techniques

The efficiency of the search can be improved by breaking the symmetries that are inherent in our approach. The search space we consider contains a large number of equivalent transfer plans that should be ignored. For this purpose we propose new dominances rules.

Symmetric transfer plans

Let us consider the example on the top of Figure 4.2. At time $t = 3$, datum units 1 and 2 share the same sources (nodes 1, 2 and 3) – *i.e.* $\forall i \in \mathcal{N}$, $1 \in O_i^3$ if and only if $2 \in O_i^3$. Thus, the role of these datum units can be swapped in the rest of the sequence, as shown in the bottom half of the figure. In other words, the sub-branchings (in the evolving graph) corresponding to units 1 and 2, and associated with the contacts occurring after σ_3, can be swapped. The dissemination length is not affected by this operation. So it is sufficient to consider one option ($\phi(4) = \{1\}$ or $\phi(4) = \{2\}$) out of the two to solve the dissemination problem, *e.g.* we can arbitrarily decide to transmit the datum unit with the lowest index first.

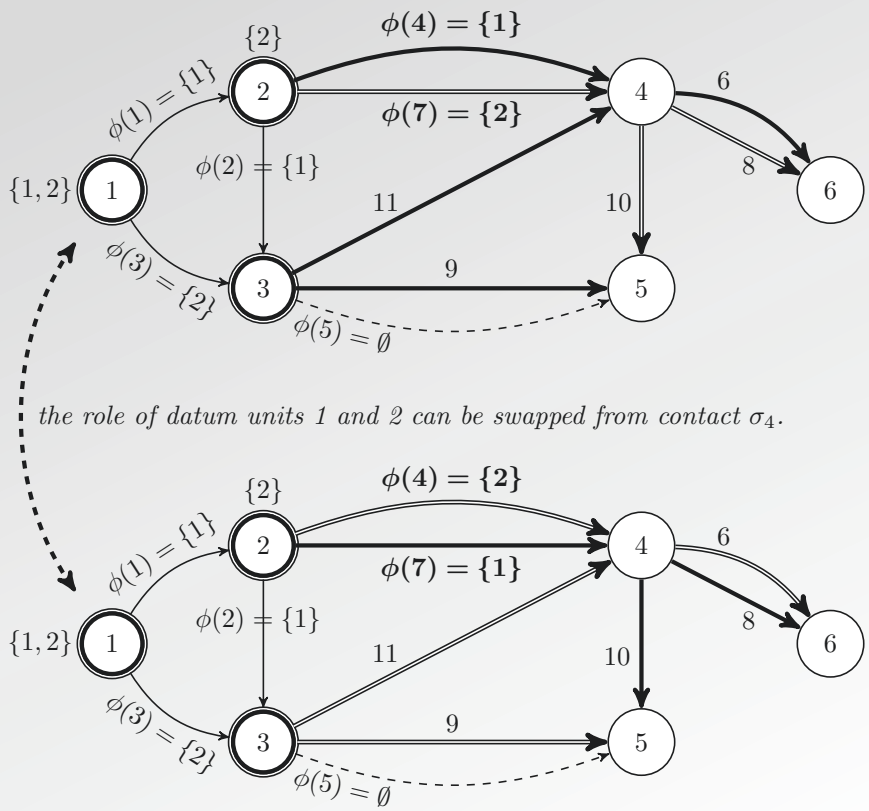


Figure 4.2 – A symmetry appears between datum units 1 and 2 when we try to set transfer $\phi(4)$.

This can be formalized as follows.

Definition 4.1 – partial transfer plan

Let $t \in \{0, 1, \ldots, m\}$ be a time index. A *partial transfer plan* ϕ^t of *length* t is a partial function from $\{1, \ldots, m\}$ to $T_\phi = \{\emptyset, \{1\}, \ldots, \{u\}\}$, whose domain X is at least $\{1, 2, \ldots, t\}$ (*i.e.* $\{1, \ldots, t\} \subseteq X$). Note that X can be empty if $t = 0$. The states O_i^x (*cf.* equation (1.1) on page 9) are still defined for any node $i \in \mathcal{N}$ and any $x \in \{0, \ldots, t\}$. The delivery length $\lambda_i(\phi^t)$ of a node $i \in \mathcal{N}$ becomes $\lambda_i(\phi^t) = \min\{x \in \{0, \ldots, t\} \,|\, O_i^x = \mathcal{D}\}$, or $\lambda_i(\phi^t) = \infty$ if such an index does not exist. The dissemination length remains $\lambda(\phi^t) = \max_{i \in \mathcal{R}} \{\lambda_i(\phi^t)\}$.

Definition 4.2 – extension of a partial transfer plan

Let ϕ^t be a partial transfer plan of length $t \in \{0, 1, \ldots, m\}$, and X refer to its domain. The *extension* $\Lambda(\phi^t)$ of ϕ^t denotes the set of valid transfer plans ϕ such that $\forall c \in X$, $\phi(c) = \phi^t(c)$.

Proposition 4.3

Let $t \in \{0, 1, \ldots, m\}$ be a time index, and ϕ^t be a partial transfer plan of length t. Let k_1 and $k_2 \in \mathcal{D}$ be two datum units (such that $k_1 < k_2$). If k_1 and k_2 are possessed by the same nodes after the first t contacts – *i.e.* $k_1 \in O_i^t(\phi^t)$ if and only if $k_2 \in O_i^t(\phi^t)$ for all nodes $i \in \mathcal{N}$ – then the set Φ_1 of transfer plans $\phi_1 \in \Lambda(\phi^t)$ such that $\phi_1(t+1) = \{k_1\}$ *dominates* the set Φ_2 of transfer plans $\phi_2 \in \Lambda(\phi^t)$ such that $\phi_2(t+1) = \{k_2\}$.

Proof. Let ϕ_2 be a valid transfer plan in Φ_2. We prove there exists a transfer plan better than or equivalent to ϕ_2 in subset Φ_1. To this end, we consider a copy ϕ_1 of ϕ_2, where we set $\phi_1(c) = \{k_1\}$ for all $c \in \{t, \ldots, m\} \,|\, \phi_2(c) = \{k_2\}$, and $\phi_1(c) = \{k_2\}$ for all $c \in \{t, \ldots, m\} \,|\, \phi_2(c) = \{k_1\}$. Transfer plan ϕ_1 thus belongs to Φ_1 (the role of k_1 and k_2 have been swapped after σ_t). Let us now prove that ϕ_1 has the same dissemination length as ϕ_2. First, it will be noted that $k \in O_i^c(\phi_2)$ and $k \in O_i^c(\phi_1)$ are obviously equivalent for any *other* unit $k \in \mathcal{D} \backslash \{k_1, k_2\}$, any node $i \in \mathcal{N}$, and at any time $c \in \{0, \ldots, m\}$. We would like to show that $k_1 \in O_i^c(\phi_2)$ is equivalent to $k_2 \in O_i^c(\phi_1)$. This is obvious in the case of $c \leq t$. We then assume it holds at a time $c > t$, and we consider the situation at time $c + 1$.

1. **If** $i \neq r_{c+1}$, **then**
   $k_1 \in O_i^{c+1}(\phi_2) \Rightarrow k_1 \in O_i^c(\phi_2) \Rightarrow k_2 \in O_i^c(\phi_1) \Rightarrow k_2 \in O_i^{c+1}(\phi_1)$.
2. **If** $i = r_{c+1}$, **then**
   $k_1 \in O_i^{c+1}(\phi_2) \Rightarrow k_1 \in O_i^c(\phi_2) \cup \phi_2(c+1)$ **and so:**
   (a) **if** $k_1 \in O_i^c(\phi_2)$, **then** $k_2 \in O_i^c(\phi_1) \Rightarrow k_2 \in O_i^{c+1}(\phi_1)$;
   (b) **if** $\{k_1\} = \phi_2(c+1)$, **then** $\{k_2\} = \phi_1(c+1) \Rightarrow k_2 \in O_i^{c+1}(\phi_1)$.

In the same way, we can prove that $k_2 \in O_i^{c+1}(\phi_1) \Rightarrow k_1 \in O_i^{c+1}(\phi_2)$ – which demonstrates the result by recurrence. Finally we can prove that $k_2 \in O_i^c(\phi_2)$ is equivalent to $k_1 \in O_i^c(\phi_1)$. Therefore $O_i^\lambda(\phi_1) = O_i^\lambda(\phi_2)$ always holds at time $\lambda = \lambda(\phi_2)$ and consequently $\lambda(\phi_1) = \lambda(\phi_2)$. □

In practice – Proposition 4.3 can be applied at each node of the search tree. Given a sequential branching algorithm like that described in Section 4.1.2, every node defines one partial transfer plan ϕ^t of size t (t being the number of transfers that have been set during the search, from the root to the node we consider in the search tree). Thus, if $T = \{\, k \in \mathcal{D} \,|\, x_{k,t+1}$ is not set to $0 \,\}$ denotes the set of units that could be transferred during contact σ_{t+1}, we can check whether two units k_1 and $k_2 \in T$ (with $k_1 < k_2$) have the same source nodes. If so, the proposition allows us to remove k_2 from T (variable $x_{k_2,t+1}$ can be set to 0). As the branch where $x_{k_1,t+1} = 1$ (*i.e.* the subset of transfer plans $\Phi_1 \subseteq \Lambda(\phi^t)$ such that $\phi(t+1) = \{k_1\}$) will be explored, we can ignore the branch where $x_{k_2,t+1} = 1$ (the set of transfer plans $\Phi_2 \subseteq \Lambda(\phi^t)$ such that $\phi(t+1) = \{k_2\}$) can be ignored.

Remark 4.2

In the trivial case where only one source node $s \in \mathcal{N}$ possesses the whole datum at the outset – *i.e.* $\mathcal{O}_s = \{1, 2, \ldots, u\}$ and $\forall i \in \mathcal{N} \backslash \{s\}$, $\mathcal{O}_i = \emptyset$ – this proposition means we only need to consider one transfer plan out of $u!$ during the search. For each transfer plan, that is for each subset of u arc-disjoint branchings of the evolving graph, we can find $u!$ symmetrical transfer plans by permuting the assignments of each datum unit to the branchings. Proposition 4.3 ensures that the branchings are assigned to the units according to their indexes (other permutations are ignored).

Consecutive transfers

Let us now turn to the example shown on the left half of Figure 4.3. Node 3 is the recipient in contacts $\sigma_1 = (1, 3)$ and $\sigma_2 = (2, 3)$, but has no opportunity to transmit data to a third node between these contacts. Therefore, the order in which datum units are sent to node 3 during these contacts is not relevant (if we assume that both are improving) – *i.e.* $O_2^3 = \{1, 2\}$ if $\phi(1) = \{1\}$ and $\phi(2) = \{2\}$, or if $\phi(1) = \{2\}$ and $\phi(2) = \{1\}$. So, to prevent such symmetries from expanding the search space, we propose considering only transfer plans where the datum units are transmitted in the order of their indexes, *i.e.* such that $\phi(1) = \{1\}$ and $\phi(2) = \{2\}$. This can be formalized as follows.

Proposition 4.4

Let $i \in \mathcal{N}$ be a node. Let k_1, $k_2 \in \mathcal{D}$ denote two datum units such that $k_1 < k_2$. One assumes there are two contacts σ_{x1} and $\sigma_{x2} \in \sigma$ $(x1 < x2)$ with $r_{x1} = r_{x2} = i$, between which node i has no opportunity to transfer data, *i.e.* $\nexists y \in \{x1, \ldots, x2\}$ with $s_y = i$. Among the valid transfer plans such that

$$k_1 \in O_{s_{x1}}^{x1-1} \cap O_{s_{x2}}^{x2-1} \backslash O_{r_{x2}}^{x2-1} \text{ and } k_2 \in O_{s_{x1}}^{x1-1} \cap O_{s_{x2}}^{x2-1},$$

the subset Φ_1 of transfer plans ϕ_1 with $\phi_1(x1) = \{k_1\}$ and $\phi_1(x2) = \{k_2\}$ *dominates* the subset Φ_2 of transfer plans ϕ_2 where $\phi_2(x1) = \{k_2\}$ and $\phi_2(x2) = \{k_1\}$ (the unit with the lowest index is transferred first).

Proof. Let $\phi_2 \in \Phi_2$ be a transfer plan such that $k_1 \in O_{s_{x1}}^{x1-1} \cap O_{s_{x2}}^{x2-1} \backslash O_{r_{x2}}^{x2-1}$; $k_2 \in O_{s_{x1}}^{x1-1} \cap O_{s_{x2}}^{x2-1}$; $\phi_2(x1) = \{k_2\}$; and $\phi_2(x2) = \{k_1\}$. We build a transfer plan $\phi_1 \in \Phi_1$ by copying ϕ_2, and by setting $\phi_1(x1) = \{k_1\}$ and $\phi_1(x2) = \{k_2\}$. Note that ϕ_1 is valid, since ϕ_2 is valid, $k_1 \in O_{s_{x1}}^{x1-1}$ and $k_2 \in O_{s_{x2}}^{x2-1}$. Moreover, we have $\lambda(\phi_1) = \lambda(\phi_2) \geq x2$ since $k_1 \notin O_{r_{x2}}^{x2-1}$. □

In practice – Proposition 4.4 can be applied every time we have to decide the value of a transfer $\phi(x1)$ for which there exists a contact $\sigma_{x2} \in \sigma$, $x1 < x2$, such that $r_{x1} = r_{x2}$ and $\nexists y \in \{x1, x1 + 1, \ldots, x2\} \,|\, s_y = r_{x1}$ (in a sequential branching algorithm like that proposed in section 4.1.2). Note that the first $x1-1$ transfers are then fixed, and the set $T = \{\, k \in \mathcal{D} \,|\, x_{k,x1}$ is not set to $0\,\}$ of units that can be transmitted during contact σ_{x1} is given. Thus, in every branch $k_2 \in T$ (associated with decision $\phi(x1) = \{k_2\}$), transfer $\phi(x2) = \{k_1\}$ can be blocked if $k_2 \in O_{s_{x2}}^{x2-1}$.

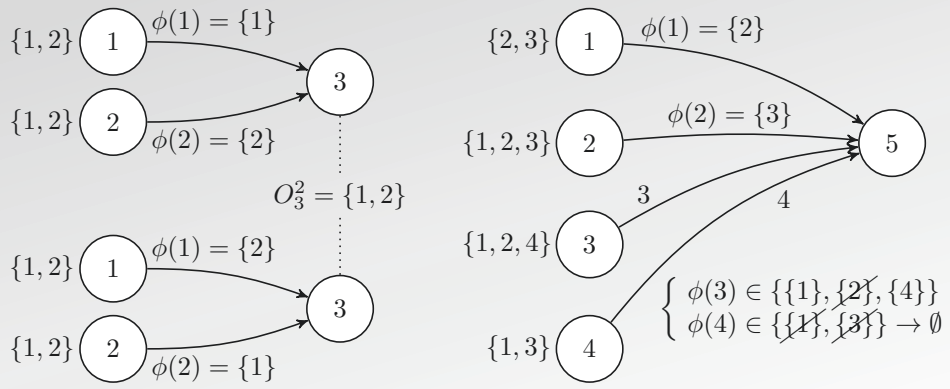


Figure 4.3 – When consecutive contacts occur, datum units can be sent in the order of their index.

The other constraints cannot be violated:

1. assertions $k_1 \in O_{s_{x1}}^{x1-1}$ and $k_2 \in O_{s_{x1}}^{x1-1}$ already hold;
2. $k_1 \in O_{s_{x2}}^{x2-1}$ is required for transfer $\phi(x2) = \{k_1\}$ to be valid;
3. and $k_1 \notin O_{r_{x2}}^{x2-1}$ is also necessary for that transfer to be minimal.

This can be *automated* simply by adding the following constraint in each of these branches:

$$\forall k_1 \in \{1, 2, \ldots, k_2 - 1\} \cap T, \quad [\, y_{s,k_2,t} = 1 \Longrightarrow x_{k_1,x2} = 0 \,] \\ (\text{with } s = s_{x2} \text{ and } t = \mathcal{T}_s(x2 - 1))$$

The use of such constraints cannot be avoided, since the state $O_{s_{x2}}^{x2-1}$ is not fixed when the value of transfer $\phi(x1)$ is being decided.

Moreover, when several contacts σ_{x2} fulfil the above conditions, we need to post the constraints for each possibility. If we look at the instance shown on the half right of Figure 4.3, $x2 = 3$ and $x2 = 4$ both need to be considered when we set $\phi(2) = \{3\}$. Finally, note that the case where $\phi(3) = \{1\}$ cannot be ignored, because no swap will be possible between transfers $\phi(2)$ and $\phi(3)$ (node 3 will not be in possession of datum unit 3 when σ_3 will occur).

Further research could include trying to detect other symmetry patterns, *e.g.* we could attempt to detect that the transfer plans such that $\phi(1) = \{2\}$, $\phi(2) = \{1\}$, $\phi(3) = \{4\}$ and $\phi(4) = \{3\}$ maximize the number of datum units received by node 5 throughout these four contacts, while ensuring that as far as possible the units are transferred in the order of their indexes. This way, more transfers could be arbitrarily set, and fewer transfer plans would need to be considered during the search.

Nogood recording

The final technique for breaking symmetries that we propose in this chapter involves registering the state of the nodes visited throughout the search. This enables branches to be pruned by detecting dominance relationships among transfer plans.

Different sequences of transfers can give rise to a similar dissemination – *i.e.* it can exist two partial transfer plans ϕ_1^t and ϕ_2^t of length $t \in \{1, \ldots, m\}$ such that $\forall i \in \mathcal{N}$, $O_i^t(\phi_2^t) \subseteq O_i^t(\phi_1^t)$. Where this is the case, the set of transfer plans $\phi_1 \in \Lambda(\phi_1^t)$ dominates the set of transfer plans $\phi_2 \in \Lambda(\phi_2^t)$ since, from any transfer plan $\phi_2 \in \Lambda(\phi_2^t)$, we can consider a better or equivalent transfer plan by taking the first t contacts of ϕ_1^t, and keeping the last $m - t$ contacts of ϕ_2. This can be formalized as follows.

Proposition 4.5

Let ϕ_1^t and ϕ_2^t be two valid partial transfer plans of length $t \in \{1, 2, \ldots, \min\{m, \lambda(\phi_1^t)\}\}$. If $O_i^t(\phi_2^t) \subseteq O_i^t(\phi_1^t)$ holds for all the nodes $i \in \mathcal{N}$, then $\Lambda(\phi_1^t)$ *dominates* $\Lambda(\phi_2^t)$.

Proof. Let ϕ_2 be a transfer plan in $\Lambda(\phi_2^t)$. We build a transfer plan $\phi_1 \in \Lambda(\phi_1^t)$ by copying the first t transfers of ϕ_1^t, and by completing with the last $m - t$ transfers of ϕ_2 – *i.e.* $\forall c \in \{1, \ldots, t\}$, $\phi_1(c) = \phi_1^t(c)$, and $\forall c \in \{t+1, \ldots, m\}$, $\phi_1(c) = \phi_2(c)$. It is assumed that $O_i^t(\phi_2) \subseteq O_i^t(\phi_1^t) = O_i^t(\phi_1)$ for all the nodes $i \in \mathcal{N}$. We thus consider that $O_i^c(\phi_2) \subseteq O_i^c(\phi_1)$ for a time index $c \in \{t+1, \ldots, m-1\}$, and let us examine the situation at index $c + 1$.

1. **If** $i \neq r_{c+1}$, **then**
   $O_i^{c+1}(\phi_2) = O_i^c(\phi_2) \subseteq O_i^c(\phi_1) = O_i^{c+1}(\phi_1)$.
2. **If** $i = r_{c+1}$, **then**
   $O_i^{c+1}(\phi_2) = O_i^c(\phi_2) \cup \phi_2(c+1) \subseteq O_i^c(\phi_1) \cup \phi_1(c+1) = O_i^{c+1}(\phi_1)$.

Consequently, we have proved by recurrence that $O_i^c(\phi_2) \subseteq O_i^c(\phi_1)$ holds for all indexes $c \in \{t, \ldots, m\}$. The validity of ϕ_1^t and ϕ_2 therefore results in the validity of ϕ_1. Moreover – since $t \leq \lambda(\phi_1^t)$ – we also have $t \leq \lambda(\phi_1) = \lambda$ and $O_r^\lambda(\phi_2) \subseteq O_r^\lambda(\phi_1)$ for all the recipient nodes $r \in \mathcal{R}$. So, $\lambda(\phi_1) \leq \lambda(\phi_2)$. □

In practice – at every node of the search tree, and after having set the new transfer $\phi_2^t(t)$, we also save the corresponding global-state

$$\mathrm{S}_t^{curr} = \{O_i^t \,|\, i \in \{1, 2, \ldots, n\}\}$$

in a list L_t. In short, S_t^{curr} saves the units possessed by each node at the end the first t transfers of the current partial transfer plan ϕ_2^t. Every list L_t then contains the global states associated with all *visited* partial transfer plans of length $t \in \{1, 2, \ldots, m\}$. From Proposition 4.5, we know that a node can be pruned if there exists a dominant state $\mathrm{S}_t^{dom} \in \mathrm{L}_t$ (corresponding to a visited partial transfer plan ϕ_1^t) such that $\forall i \in \mathcal{N}$, $O_i^t(\mathrm{S}_t^{curr}) \subseteq O_i^t(\mathrm{S}_t^{dom})$.

Such a symmetry-breaking technique falls within the well-known *nogood recording* framework, which was first introduced by Schiex and Verfaillie [39]. The major drawback of such an approach is the huge number of global states that can be generated, and thus recorded. An easy way of managing this is to implement lists L_t as truncated heaps sorted according to the "cardinal" $|\mathrm{S}_t| = \sum_{i \in \mathcal{N}} |O_i^t(\mathrm{S}_t)|$ of the states $\mathrm{S}_t \in \mathrm{L}_t$. The intuitive idea is to prioritize the global states that are most likely to include other global states.

Remark 4.3

If the global states are represented with bit vectors, testing whether two sets S_t^1 and S_t^2 are such that $\mathrm{S}_t^1 \subseteq \mathrm{S}_t^2$ is trivial using boolean operations. This computational efficiency makes it possible to store several thousand global states in each list L_t, $t \in \{1, \ldots, m\}$, and thereby to improve the performances of the solver drastically, *cf.* Section 4.3 for more details.

Three methods have been proposed to break symmetries inherent in our approach. The first two techniques can be qualified as *proactive*, as both are intended to avoid generating symmetric solutions during the search, whereas the third is more *reactive* insofar as it aims to detect (and prune) symmetric branches after they have been generated.

4.3 Computational results

In this section numerical results are reported and discussed. All experiments were performed in the same conditions as for Chapter 3 (the same instances, the same machine, the same criteria, and still with a one-hour time limit), *cf.* Section 3.5.1. We used CP-Optimizer, the constraint-programming engine developped by IBM-ILOG. Note that the best results achieved with integer linear programming will serve as a reference for subsequent comparisons.

The model described in Section 4.1 is discussed in Section 4.3.1, and the features proposed in Section 4.2 are evaluated in Section 4.3.2.

4.3.1 About the model

Let us discuss the computational results reported in Table 4.1. These results were obtained using three different algorithms. First **min** consists in solving the constraint programming model defined by constraints (3.4) through (4.3) (the transfer plan is therefore required to be minimal). Then, **min+st/act** follows the same approach, but by also considering constraint (4.4) (here the transfer plan is required to be strictly-active, in addition of being minimal). Finally, procedure **prep+min+st/act** consists in calling the preprocessing procedure defined in Chapter 3 (we only consider the strategies reported in Table 3.8, page 93), followed by **min+st/act**.

The experimental results reported in Table 4.1 may appear disappointing at first glance, especially compared to those reported in Table 3.8. However, it should be remarked that there is a significant gap between the two model **min** and **min+st/act**. This shows that we were successful in implementing the dynamical deduction procedures that were discussed in the conclusion of Chapter 3 – *i.e.* we managed to integrate fully into the solver the concept of strictly-active transfer plan. This was made possible through the addition of a-variables, and through non-linear constraints (4.4).

Moreover, there is also a clear gap between algorithms **min+st/act** and **prep+min+st/act**, showing that the preprocessing procedures we proposed outperform the consistency algorithms implemented in CP Optimizer.

		min			**min+st/act**			**prep+min+st/act**		
	name	*solved*	*feas*	*cpu*	*solved*	*feas*	*cpu*	*solved*	*feas*	*cpu*
1	**3u10n**	100	-	3.1	100	-	4.1	100	-	0.84
	4u20n	100	-	12.3	100	-	10.2	100	-	5.5
2	**4u50n**	100	-	339	100	-	60.2	100	-	18.5
	4u100n	80.0	20.0	818	100	-	196	95.0	5.0	333
	5u50n	69.6	30.4	1316	91.3	8.7	555	100	-	173
3	**10u10n**	18.8	50.0	3027	18.8	50.0	2964	37.5	37.5	2269
	50u10n	0.00	31.3	3602	0.00	0.00	3601	0.00	0.00	3601
	100u10n	0.00	33.3	3604	0.00	0.00	3603	0.00	0.00	3603

Table 4.1 – The results achieved with CP Optimizer and different models.

4.3.2 The additional features

In fact, to get the most out of constraint programming, a custom branching algorithm often needs to be implemented. The strategy proposed in Section 4.1.2 is quite easy to implement with CP-Optimizer. As a reminder, transfers are set in the order of the sequence. The search is guided in accordance with a heuristic. If the number of backtracks starts to become unreasonably large, the search is restarted from scratch with the built-in algorithm. In this case, the solver is notified that the x-variables are the decision variables and that they should be given priority during the branching. Experimental results are reported in Table 4.2, columns **none**. Note that the proportion of instances successfully solved is significantly better, but is still not comparable to results obtained with integer linear programming.

To obtain competitive results, we must consider the *ad hoc* tools described in Section 4.2 – namely the weak (**wlb**) and the strong (**slb**) lower bounds, the proactive symmetry-breaking techniques (**sym**) and/or the nogood-recording (**ngr**), *cf.* Tables 4.2 through 4.4.

Lower bounds – **wlb** and **slb** have positive effects on the efficiency of our algorithm. They enable the solver to prune some branches earlier. The weak lower bound provides better results for the first two groups – *i.e.* the smallest instances – than for the third group; and, conversely, the strong lower bound provides better results for the third group – *i.e.* the hardest instances – than for the first two groups. As is often the case, there is a balance to be struck between on the one hand heavy computations but tight bounds, and on the other light computations but weak bounds.

Symmetry-breaking techniques – **ngr** and **sym** enable the search space to be significantly reduced at a low computational cost. The numerical results show that both operate well across all classes.

In summary, algorithm **sym+ngr+wlb** should be used on the first two groups. **sym+ngr+slb** is more suitable for large instances, *cf.* Table 4.5.

		none		sym		sym+ngr		sym+ngr+wlb	
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*
1	**3u10n**	100	0.46	100	0.47	100	0.46	100	0.39
	4u20n	100	1.8	100	1.7	100	1.7	100	1.3
2	**4u50n**	100	6.3	100	3.2	100	3.2	100	2.7
	4u100n	100	198	100	160	100	58.5	100	88.5
	5u50n	100	51.2	100	33.7	100	39.4	100	20.7
3	**10u10n**	43.8	2236	50.0	1807	81.3	812	87.5	462
	50u10n	43.8	2106	43.8	2105	43.8	2102	75.0	1029
	100u10n	83.3	1184	83.3	1184	83.3	1189	100	702

Table 4.2 – Computational results obtained with CP Optimizer – *part*-1.

		sym+ngr+slb		sym+wlb		sym+slb		ngr	
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*
1	**3u10n**	100	0.59	100	0.38	100	0.59	100	0.46
	4u20n	100	1.8	100	1.3	100	1.8	100	1.7
2	**4u50n**	100	3.6	100	2.6	100	3.7	100	5.2
	4u100n	100	216	100	181	100	174	100	107
	5u50n	100	73.1	100	12.6	100	66.0	100	54.2
3	**10u10n**	100	36.1	68.8	1135	93.8	379	43.8	2143
	50u10n	87.5	640	75.0	1032	75.0	1061	43.8	2107
	100u10n	100	1220	100	731	100	1291	83.3	1183

Table 4.3 – Computational results obtained with CP Optimizer – *part*-2.

		ngr+wlb		ngr+slb		wlb		slb	
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*
1	**3u10n**	100	0.38	100	0.59	100	0.38	100	0.60
	4u20n	100	1.3	100	2.0	100	1.5	100	2.2
2	**4u50n**	100	4.3	100	7.7	100	5.6	100	11.9
	4u100n	100	84.6	100	279	100	154	100	395
	5u50n	100	46.7	100	241	100	29.9	95.7	245
3	**10u10n**	50.0	1807	68.8	1425	43.8	2091	62.5	1725
	50u10n	62.5	1689	62.5	1497	62.5	1662	62.5	1495
	100u10n	83.3	1187	83.3	1431	83.3	1187	83.3	1473

Table 4.4 – Computational results obtained with CP Optimizer – *part*-3.

	name	*algorithm*	*solved*	*feas*	*cpu*
1	**3u10n**	sym+ngr+wlb	100	-	0.39
	4u20n	sym+ngr+wlb	100	-	1.3
2	**4u50n**	sym+ngr+wlb	100	-	2.7
	4u100n	sym+ngr+wlb	100	-	88.5
	5u50n	sym+ngr+wlb	100	-	20.7
3	**10u10n**	sym+ngr+slb	100	-	36.1
	50u10n	sym+ngr+slb	87.5	12.0	640
	100u10n	sym+ngr+slb	100	-	1220

Table 4.5 – The algorithms that obtained the best results with CP Optimizer.

4.4 Conclusion

In this chapter we proposed an algorithm based on constraint programming for solving the *dissemination problem*. It outperforms our previous algorithm, with a better heuristic branching algorithm and some extensions – *i.e.* some lower bounds and some symmetry breaking techniques.

Table 4.6 shows the best computational results achieved so far, for every class and across Chapters 3 and 4.

In the next chapter we plan to investigate several methods for solving the *dissemination problem* in an uncertain context. This implies studying *robust optimization* in order to find transfer plans which remain valid when not all transfers are successful.

	algorithm		***ad hoc* procedures**		**results**		
name	*solver*	*model*	*preprocessing*	*features*	*solved*	*feas*	*cpu*
3u10n	CPLEX	min+st/act	minimal	-	100	-	0.26
4u20n	CPO	min+st/act	light	sym+wlb	100	-	1.3
4u50n	CPO	min+st/act	light	sym+wlb	100	-	2.6
4u100n	CPLEX	min	light	-	100	-	20.1
5u50n	CPO	min+st/act	light	sym+wlb	100	-	12.6
10u10n	CPO	min+st/act	aggressive	sym+ngr+slb	100	-	36.1
50u10n	CPO	min+st/act	normal	sym+ngr+slb	87.5	12.5	540
100u10n	CPO	min+st/act	normal	sym+wlb	100	-	129

Table 4.6 – The best results achieved so far.

5 Robust optimization

Robust optimization is an approach to optimization problems affected by uncertainty. In this chapter one will seek a transfer plan which ensures the delivery of each datum unit to all recipients when a given number $\Gamma \leq m$ of transfers might fail. First we will formalize this new problem, focusing particularly on the differences with the original problem (*cf.* Section 5.1). Thereafter we will propose a necessary and sufficient condition for a transfer plan to be "robust". This will finally lead to a branching algorithm (*cf.* Section 5.2), implemented in practice with constraint programming (*cf.* Section 5.3).

Contents

5.1 The robust dissemination problem 121
5.1.1 Formal description 121
5.1.2 Robustness and evolving graphs 122
5.2 Robust optimization 123
5.2.1 Necessary and sufficient condition for a transfer plan to be Γ-robust 124
5.2.2 Enumeration procedure 127
5.3 A constraint programming approach 130
5.3.1 Model 130
5.3.2 Additional features 133
5.4 Preliminary results 138
5.4.1 About the benchmark 138
5.4.2 Numerical results 140
5.5 Conclusion 142

In the previous chapters, the dissemination problem has been addressed for the case where transfers cannot fail. We now propose to tackle a variant where failures might occur. We would like to find transfer plans during which every datum unit is correctly delivered to all recipient nodes, *regardless* the failures scenario. Of course, by "failure" we mean anything which prevents a transfer to terminate successfully – *e.g.* link problems, battery depletions, ... In this way, we hope to cover a wide range of real applications. In practice, we will assume that at most $\Gamma \in \mathbb{N}$ transmissions can fail during a scenario, following the well known approach proposed by Bertsimas and Sim [6].

The present chapter is organized as follows. In Section 5.1, we extend the formulation of dissemination problem to the robust optimization case. Then, in Section 5.2, we propose an enumeration algorithm to find all valid robust solutions. This algorithm is integrated into a constraint-programming solver (like in Chapter 4) in Section 5.3, and finally assessed in Section 5.4.

5.1 The robust dissemination problem

In this section, we generalize the formulation of the dissemination problem to the case where one considers failures. In addition, we show that these two problems (the robust and the non-robust problems) are different.

5.1.1 Formal description

The *robustness* criteria may depend on numerous parameters (the intended applications, the environment, ...). In this chapter, we consider a parameter, $\Gamma \in \mathbb{N}$, to characterize the minimum "level" of robustness the solution has to guarantee. The problem is to compute a "Γ-robust" transfer plan minimizing the dissemination length, *i.e.* a transfer plan ensuring that the recipient nodes are served if at most Γ failures occur.

Formally, a *scenario* of Γ failures is a function $\mathtt{S} : \{1, 2, \ldots, m\} \mapsto \{0, 1\}$ such that $\sum_{c=1}^{m} \mathtt{S}(c) = \Gamma$. In fact, $\mathtt{S}(c) = 1$ indicates that contact $\sigma_c \in \sigma$ has failed. Therefore, given a scenario $\mathtt{S}$ and a transfer plan ϕ, we can compute the *realization* of the *states* $O_i^t(\mathtt{S}, \phi)$ (for all nodes $i \in \mathcal{N}$ and all time indexes $t \in \{0, 1, \ldots, m\}$) as follows:

$$\begin{array}{l} (1)\ \forall i \in \mathcal{N},\ O_i^0 = \mathcal{O}_i, \\ (2)\ \forall c \in \{1, 2, ..., m\} \,|\, \mathtt{S}(c) = 1,\ \forall i \in \mathcal{N},\ O_i^c = O_i^{c-1}, \\ (3)\ \forall c \in \{1, 2, ..., m\} \,|\, \mathtt{S}(c) = 0,\ O_{r_c}^c = O_{r_c}^{c-1} \cup [\, \phi(c) \cap O_{s_c}^{c-1} \,], \\ (4)\ \forall c \in \{1, 2, ..., m\} \,|\, \mathtt{S}(c) = 0,\ \forall i \in \mathcal{N} \backslash \{r_c\},\ O_i^c = O_i^{c-1}. \end{array} \tag{5.1}$$

Consequently state $O_i^t(\mathtt{S}, \phi)$ denotes the set of datum units possessed by node i at time t when transfer plan ϕ is used during scenario $\mathtt{S}$.

The delivery length $\lambda_i^\Gamma(\phi)$ of node $i \in \mathcal{N}$ must now indicate the smallest time index at which one can guarantee that i possesses all the datum units, regardless the contacts that fail. Thus, we will now consider:

$$\lambda_i^\Gamma(\phi) = \min \{\, t \in \{0, 1, \ldots, m\} \,|\, O_i^t(\mathtt{S}, \phi) = \mathcal{D} \text{ for any scenario } \mathtt{S} \text{ of at most } \Gamma \text{ failures} \,\}$$

In this way, the dissemination length $\lambda^\Gamma(\phi) = \max_{i \in \mathcal{R}} \{\lambda_i^\Gamma(\phi)\}$ indicates the time index at which we can guarantee that all the recipient nodes possess all the datum units (regardless the contacts that fail).

The *robust dissemination problem* is to find a transfer plan ϕ minimizing the dissemination length $\lambda^\Gamma(\phi)$. Note that a transfer plan ϕ is Γ-robust when $\lambda^\Gamma(\phi) \neq \infty$. Only such solutions are admissible for the robust dissemination

problem, so an instance can be *feasible* with a given value of Γ, but *infeasible* with a greater value of that parameter. In particular, some feasible instances of the dissemination problem ($\Gamma = 0$) may become infeasible with respect to the robust dissemination problem ($\Gamma \geq 1$). However, Γ-robust transfer plans are always p-robust for all $0 \leq p \leq \Gamma$.

The robust dissemination problem is NP-Hard in the strong sense, since the dissemination problem (a strongly NP-Hard problem) is a special case of the robust dissemination problem where $\Gamma = 0$.

5.1.2 Robustness and evolving graphs

Instances of both problems (the dissemination and the robust dissemination problems) can be described by evolving graphs. Let us remind that these are multigraphs whose vertices represent nodes and whose arcs represent a set of connections between these nodes. Each arc is labelled with time intervals which indicate when the corresponding link is active. To appropriately take account of time constraints, the notion of *path* must be replaced by the notion of *journey*, *i.e.* an ordered set of arcs having *increasing* labels. For our needs, each contact is thus represented by an arc whose label is given by its position in the sequence – *cf.* Figure 1.1 on page 11.

Actually, a set of arc-disjoint branchings (in the evolving graph) that are rooted on the source nodes of a given unit $k \in \mathcal{D}$, and which globally covers all the recipient nodes, defines a possible store-forward routing to disseminate unit k. For example, in Figure 1.1, the bold arcs form a set of branchings to disseminate datum unit 1 from nodes 1 and 2 to all the other nodes. In the following, such a set of branchings is just named a *covering branching*, since this actually is one branching if we consider a virtual root node to transmit unit k to the source nodes (1 and 2) at time 0.

Therefore – as shown in Section 2.3.1 – solving the dissemination problem ($\Gamma = 0$) can be seen as finding u arc-disjoint such covering branchings – *i.e.* *one* per datum unit (see the branchings with bold and double arcs).

An intuitive extension to the robust case consists in searching for exactly $\Gamma + 1$ arc-disjoint covering branchings *per datum unit*. The resulting transfer plan (built from the $(\Gamma + 1) \times u$ covering branchings) is Γ-robust. It defines how $\Gamma + 1$ copies of each datum unit can be routed (along independent and contact-disjoint journeys) to the recipient nodes. This way, Γ failures cannot be enough to prevent a given node to receive a given datum unit.

However, it is worth nothing that this approach is over protective and the existence of $(\Gamma+1)\times u$ covering branchings is a sufficient, but not necessary, condition for a Γ-robust solution to exist.

Proposition 5.1

The existence of $(\Gamma+1)\times u$ mutually-arc-disjoint covering branchings in the evolving graph (*i.e.* $\Gamma+1$ branchings per datum unit) is a sufficient, but not necessary, condition for a Γ-robust transfer plan to exist.

Proof. The condition is sufficient, since Γ failures are not enough to invalidate $\Gamma+1$ arc-disjoint branchings. To prove it is not necessary, we then consider the evolving graph depicted in Figure 2.7 on page 39. We assume that there is only one datum unit, whose the only source is node t, and that all nodes are recipient – *i.e.*

$$\begin{cases} \mathcal{R}=\mathcal{N}=\{t,a,b,\ldots,f\}\,;\mathcal{D}=\{1\}\,; \\ \mathcal{O}_t=\{1\}\,;\text{and }\forall i\in\{a,b,\ldots,f\},\,\mathcal{O}_i=\emptyset. \end{cases}$$

The valid transfer plan ϕ such that $\forall c\in\{1,\ldots,12\}$, $\phi(c)=\{1\}$ is 1-robust – *i.e.* no single failure can prevent the correct delivery of unit 1. Yet we have already shown that there does not exist two arc-disjoint covering branchings rooted at node t (the source) in this evolving graph. □

Note that this proposition justifies the present chapter. Solving the robust case is **not** equivalent to solve the initial problem by considering $\Gamma+1$ times more datum units.

5.2 Robust optimization

In this section, we propose a branching algorithm to enumerate all Γ-robust transfer plans. This procedure is based on a necessary and sufficient condition that will be discussed in the first place. Moreover, we propose a polynomial-time algorithm to check that a given transfer plan is Γ-robust. This branching algorithm will be incorporated into a branch-and-bound procedure described in Section 5.3 (and developed with a constraint programming approach).

5.2.1 Necessary and sufficient condition for a transfer plan to be Γ-robust

In the worst case, all the failures prevent the same node $r \in \mathcal{R}$ to receive the same datum unit $k \in \mathcal{D}$, *i.e.* the failures break all the journeys enabling r to receive k. Therefore, to prove that a given valid transfer plan ϕ is Γ-robust, one can show that for all datum units $k \in \mathcal{D}$, and all recipient nodes $r \in \mathcal{R}$, the minimum number of contacts which must fail to prevent r from receiving unit k, is strictly-greater than Γ. To this end, for each datum unit $k \in \mathcal{D}$ and each node $r \in \mathcal{R}$, we consider the transportation network $G(\phi, k, r) = (X, U)$ built as follows:

1. First, we add a source vertex $src \in X$ and a sink node $snk \in X$.
2. For each node $i \in \mathcal{N}$, we add $\mu_i + 1$ nodes $\{i_0, i_1, \ldots, i_{\mu_i}\} \subseteq X$, where $\mu_i = |\{\, \sigma_c \in \sigma \,|\, r_c = i \,\}|$ is the number of contacts whose receiver is i.
3. Next, we add one arc $(src, i_0) \in U$ for each source node $i \in \mathcal{N} \,|\, k \in \mathcal{O}_i$, and one arc $(r_{\mu_r}, snk) \in U$. All those arcs have an infinite capacity.
4. For each node $i \in \mathcal{N}$, we add arcs $\{(i_0, i_1), (i_1, i_2), \ldots, (i_{\mu_i - 1}, i_{\mu_i})\}$ with infinite capacities.
5. For each contact $\sigma_c = (i, j) \in \sigma$ such that $\phi(c) = \{k\}$ – assuming that node i is the receiver of $x \in \{0, \ldots, c-1\}$ contacts before σ_c – and that σ_c is the y^{th} ($y \in \{1, 2, \ldots, m\}$) contact where node j is the receiver – we add an arc $(i_x, j_y) \in U$ of capacity 1.

Such a graph has been proposed in Section 2.2.2 to prove that the non-robust dissemination problem *with one recipient* is polynomial and can be reduced to the solving of a max-flow problem. The graph has been slightly simplified as we only consider one datum unit at a time. In Figure 5.1b, we report the flow network $G(\phi, k = 1, r = 5)$ associated with the transfer plan ϕ described in Figure 5.1a. Of course this network can be simplified using the procedure defined on page 33 (*cf.* Figure 2.6).

Following the proof of Section 2.2.2, we can prove that a *src*-*snk* flow in $G(\phi, k, r)$ describes a set of arc-disjoint journeys to transmit unit k, from one or several sources, to node r. These journeys are obtained from the saturated arcs representing a contact – *i.e.* the arcs having a flow and a capacity of 1. Each unit of flow entering vertex *snk* actually correspond to a journey from a source node to r. Thus, we can assert that Γ failures cannot be enough to prevent node r from receiving unit k during transfer plan ϕ if the maximum amount of flow traversing $G(\phi, k, r)$ is greater or equal to $\Gamma + 1$ – *i.e.* if there are at least $\Gamma + 1$ arc-disjoint journeys to deliver unit k to node r.

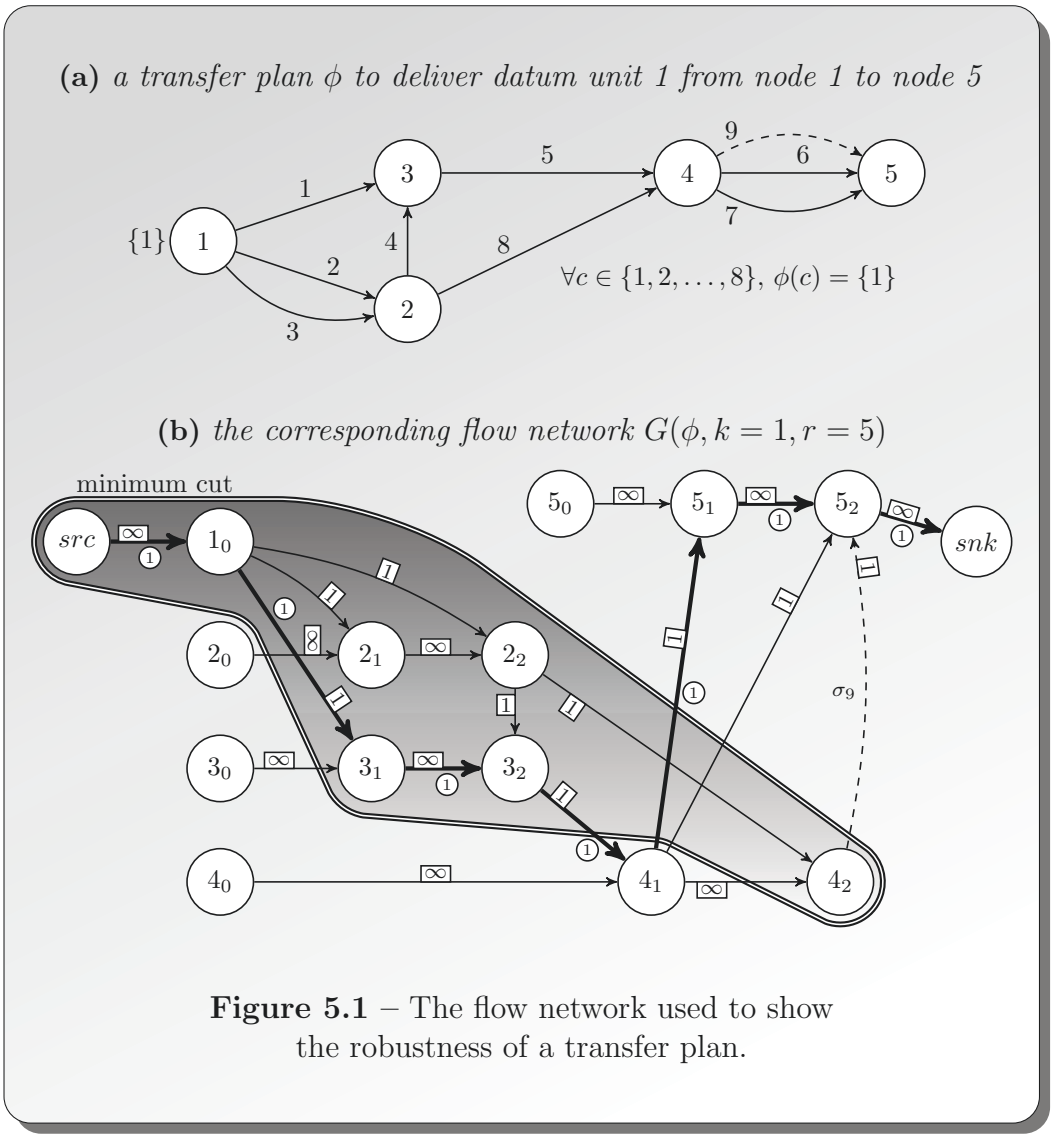


Figure 5.1 – The flow network used to show the robustness of a transfer plan.

In Figure 5.1, only one unit of flow can traverse $G(\phi, 1, 5)$, *i.e.* the transfer plan is not robust to failures. The minimum cut (depicted in gray) computed by Ford-Fulkerson [24] algorithm informs us that node 5 cannot receive datum unit 1 if contact $\sigma_5 = (3, 4)$ fails. This contact is represented by the only arc – namely $(3_2, 4_1)$ – which is *in* the minimum cut (all the journeys from nodes 1 to 5 actually use contact σ_5). On the other hand, adding one contact $(4, 5)$ at time 9 (with $\phi(9) = \{1\}$) would improve the robustness of the solution – *cf.* the augmenting path $(src, 1_0, 2_1, 2_2, 4_2, 5_2, snk)$. The arc-disjoint journeys would then be $(\sigma_1, \sigma_5, \sigma_6)$ and $(\sigma_2, \sigma_8, \sigma_9)$.

Formally, this results in the following proposition.

Proposition 5.2

Let ϕ be a valid transfer plan. ϕ is Γ-robust if and only if, for all datum units $k \in \mathcal{D}$, and all recipient nodes $r \in \mathcal{R}$, there exists a *src-snk* flow of value $\Gamma + 1$ in transportation network $G(\phi, k, r)$.

Proof. In this proof, we assume that the maximum flow is finite. Indeed, the maximum flow is infinite (and then greater than $\Gamma + 1$) if and only if node r is a source ($k \in \mathcal{O}_r$) – *i.e.* if and only if even an infinite number of failure is not sufficient to prevent r to possess k at the end of the sequence.

The condition is obviously sufficient, since Γ failures cannot be sufficient to invalidate the $\Gamma + 1$ journeys enabling r to receive k which correspond to a flow having a value of $\Gamma + 1$. To show the condition is necessary, we apply the max-flow/min-cut theorem, which states that the maximum amount of flow passing from *src* to *snk* in $G(\phi, k, r)$ is equal to the minimum capacity of a *src-snk* cut. Only arcs representing a contact (with a capacity of 1) can belong to such a cut (others have an infinite capacity). Thus, if the capacity of the minimum cut does not exceed Γ, then it exists a scenario with at most Γ failures during which node r will not be able to receive unit k. The failures are given by the minimum cut. Removing all these arcs prevents any flow to pass from *src* to *snk* – *i.e.* this breaks all the journeys enabling r to receive unit k in the evolving graph. □

On a practical level, Proposition 5.2 comes to a polynomial time algorithm to check whether a transfer plan is Γ-robust. For each datum unit and each recipient, we solve a max-flow problem in order to verify that there exist at least $\Gamma + 1$ mutually arc-disjoint journeys linking a source of this datum unit to this recipient. This test runs in $O(u.|\mathcal{R}|.(\Gamma + 1).(n + m))$ time. For each pair $(k, r) \in \mathcal{D} \times \mathcal{R}$ we need to find $\Gamma + 1$ augmenting paths in a graph that contains at most $1 + n + 2m$ arcs.

This result is the basis of the enumeration procedure that we propose in the following section. It enumerates the transfers plans where $\Gamma + 1$ journeys link the sources and each recipient node.

5.2.2 Enumeration procedure

The most naive method to enumerate the set of all Γ-robust transfer plans, consists in enumerating the set of all valid transfer plans, and in keeping only Γ-robust solutions – *i.e.* the transfer plans which guarantee that every node receives the whole datum in any scenario of at most Γ failures. To this end, we can use the branching algorithm proposed in Section 4.1.2 (on page 100). Transfers are set in the order of the sequence σ, from $\phi(1)$ to $\phi(m)$. At each node of the search tree, the first transfer $\phi(c)$ that is not yet set is selected, and one branch is created for each possible value – *i.e.* one branch for each unit $k \in O_{s_c}^{c-1}$, and one branch for the null transfer. The algorithm proposed in Section 5.2.1 is finally used to filter non-robust solutions.

Unfortunately, such an approach compels us to explore the whole search space. To avoid this, we propose to generalize the dominance rules proposed in Section 3.1 (on page 46).

Robust-minimal transfer plans

In Chapter 3, a 0-robust transfer plan ϕ is said to be *minimal* if every transfer $\phi(c)$ is either improving (node r_c receives a new datum unit, $|O_{r_c}^{c-1}| < |O_{r_c}^{c}|$) or null (no datum unit is transmitted, $\phi(c) = \emptyset$) – *i.e.* if no node receives the same datum unit more than once. The set of such transfer plans was shown to be dominant where $\Gamma = 0$. Although this dominance rule no longer holds where $\Gamma > 0$, it can be generalised. Note that the main idea is still to avoid fruitless transfers.

Definition 5.1

Transfer $\phi(c) = \{k\}$, $c \in \{1, 2, \ldots, m\}$, $k \in \mathcal{D}$, is *robust-improving* if and only if it improves the "level" of robustness associated with datum unit k and node r_c – *i.e.* the minimum number of failures required to prevent node r_c from receiving datum unit k – without, however, exceeding Γ.

In practice, a transfer $\phi(c) = \{k\}$ is then robust-improving if and only if:

1. the maximum amount of flow passing from *src* to *snk* in transportation network $G(\phi, k, r_c)$ is larger when we consider transfer $\phi(c)$, than when we only consider the $c-1$ first contacts (*i.e.* when the capacity of the arcs which correspond to a contact σ_t with $t \geq c$, is set to 0);
2. the value of such a flow does not exceed $\Gamma + 1$.

Of course, an improving transfer is also robust-improving.

Definition 5.2 – robust-minimal transfer plan

A transfer plan ϕ is *robust-minimal* if all its transfers are null or robust-improving.

In Figure 5.1, transfer $\phi(6)$ is robust-improving, since it "opens" the first journey to node 5. Transfer $\phi(7)$ is not robust-improving, because adding or removing arc $(4_1, 5_2)$ does not increase the amount of flow that can traverse $G(\phi, 1, 5)$. If we set $\phi(7) = \emptyset$, ϕ becomes robust-minimal. It remains robust-minimal if we consider $\phi(9) = \{1\}$ (since $\phi(9)$ is robust-improving).

Proposition 5.3

The set of *robust-minimal* transfer plans is dominant.

Proof. Let ϕ be a valid non-robust-minimal transfer plan. So there exists at least one transfer $\phi(c)$ which is neither null, nor robust-improving. Therefore the transfer plan ϕ' – obtained by copying ϕ and by setting $\phi'(c) = \emptyset$ – has the same dissemination length than ϕ – *i.e.* $\lambda^{\Gamma}(\phi') = \lambda^{\Gamma}(\phi)$. This process is to be repeated as long as the new transfer plan is not robust-minimal. □

Robust-strictly-active transfer plans

Given the definition of a robust-improving transfer, the concept of *strictly-active* transfer plan can be generalised straightforwardly. The idea remains to prevent postponement of improving (robust-improving) transfers.

Definition 5.3 – robust-strictly-active transfer plan

A transfer plan ϕ is *robust-strictly-active* if and only if all transfers are robust-improving when possible – *i.e.* $\forall c \in \{1, \ldots, m\}$, if $\exists k \in O_{s_c}^{c-1}$ such that $\phi(c) = \{k\}$ is robust-improving, then $\phi(c)$ is robust-improving.

Proposition 5.4

The set of *strictly-active* transfer plans is dominant.

Proof. Let ϕ be a non-robust-strictly-active transfer plan – *i.e.* there exist a non-robust-improving transfer $\phi(c)$, $c \in \{1, \ldots, m\}$, and a datum unit $k \in \mathcal{D}$ such that $\phi(c) = \{k\}$ would be robust-improving. Let then ϕ' be the transfer plan obtained by copying ϕ, and by setting $\phi'(c) = \{k\}$. The dissemination length of ϕ' is always better than or equal to the dissemination length of ϕ – *i.e.* $\lambda^{\Gamma}(\phi') \leq \lambda^{\Gamma}(\phi)$. Of course, this process must be repeated as long as the new transfer plan is not robust-strictly-active. □

Proposition 5.5

The set of *robust-minimal* and *robust-strictly-active* transfer plans is also dominant (by combining the above proofs).

Enumerating robust transfer plans

Proposition 5.5 states that enumerating robust-minimal and robust-strictly-active transfer plans is sufficient to solve the robust dissemination problem (*i.e.* other transfer plans can be ignored). Therefore – if we use a sequential branching algorithm – we can skip all the branches that correspond neither to a null transfer, nor to a robust-improving transfer.

At each node of the search tree, the earliest transfer $\phi(c)$ that is not yet set is selected, and one branch is created for each possible transfer $\phi(c) = \{k\}$ ($k \in O_{s_c}^{c-1}$) that "opens" a new journey from a source of k to node r_c. If no such a transfer exists, then $\phi(c)$ is set to $\emptyset$. A transfer plan then is Γ-robust as soon as every recipient node has received $\Gamma + 1$ copies of each unit.

If we keep the flows (used to check that all transfers are robust-improving) from one node to another during the search, testing whether a given transfer is robust-improving then takes $O(n + m)$ time, because only one iteration of the Ford-Fulkerson [24] algorithm is required. At most one new journey can be found. Indeed, a single transfer cannot enable two or more new journeys to be found at the same time.

Remark 5.1

In practice, we store only one flow network in memory. The graph defined in Section 5.2.1 has a structure that essentially depends on the instance we consider. In particular, only the capacities vary between two graphs $G(\phi_1, k_1, r_1)$ and $G(\phi_2, k_2, r_2)$ (the capacities only depends on the partial transfer plan, the datum unit and the node we consider). Therefore only the capacities – associated with each unit $k \in \mathcal{D}$ and each node $i \in \mathcal{N}$ – and the resulting flows must be stored in a reversible (*i.e.* backtrack-able) data structure. The structure of the graph itself can be static.

5.3 A constraint programming approach

In this section, we implement the scheme proposed in Section 5.2 for solving the robust dissemination problem. We use constraint programming, and take advantage of pre-implemented tools to easily develop a complete branch-and-bound procedure. Moreover, we propose *ad hoc* methods (*e.g.* lower bounds) to speed up the solving of the problem.

5.3.1 Model

The model proposed below is an update of the model proposed in Chapter 4 for the case $\Gamma = 0$. All transfers are required to be robust-improving. Thus, every recipient node $r \in \mathcal{R}$ must receive exactly $\Gamma + 1$ copies of each datum unit $k \in \mathcal{D}$. At the outset, we assume that r possesses $\Gamma + 1$ (resp. 0) copies of k if $k \in \mathcal{O}_r$ (resp. $k \notin \mathcal{O}_r$).

As a reminder, for each node $i \in \mathcal{N}$ is defined $\mathcal{T}_i$, the set of time indexes at which the state of i can change, *i.e.* $\mathcal{T}_i = \{0\} \cup \{c \in \{1, 2, \ldots, m\} \,|\, r_c = i\}$. In addition, $\forall t \in \{0, 1, \ldots, m\}$, we refer by $\mathcal{T}_i(t)$ to the last contact occurring before time t where node i is the receiver, *i.e.* $\mathcal{T}_i(t) = \max\{t' \in \mathcal{T}_i \,|\, t' \leq t\}$.

The variables of our model are defined as follows:

- $\forall k \in \mathcal{D}$, $\forall c \in \{1, \ldots, m\}$, $x_{k,c} = 1$ if datum unit k is transmitted from node s_c to node r_c during contact σ_c, and $x_{k,c} = 0$ otherwise.
- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, $\forall t \in \mathcal{T}_i$, $y_{i,k,t}$ indicates the number of copies of datum unit k possessed by node i after contact σ_t. Thus, the domain of these variables is $\{0, 1, \ldots, \Gamma + 1\}$. See Remark 3.8 on page 82.

- $\forall c \in \{1, \ldots, m\}$, $a_c = 1$ if transfer $\phi(c)$ is robust-improving, and $a_c = 0$ otherwise.
- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, variable $\lambda_{i,k}$ represents the delivery length associated with datum unit k and node i – *i.e.* the date from at node i possesses $\Gamma + 1$ copies of datum unit k. Its domain is $\mathcal{T}_i \cup \{\infty\}$. $\lambda_{i,k} = \infty$ means that node i does not recover unit k during the transfer plan. In practice we consider that $\infty = m + 1$.
- $\forall i \in \mathcal{R}$, variable $\lambda_i = \max_{k \in \mathcal{D}} \{\lambda_{i,k}\}$ then represents the delivery length of recipient node i – *i.e.* the date from which node i possesses all datum units in any scenario of at most Γ failures. Its domain is $\mathcal{T}_i$ (because i has to be served).
- Finally, variable $\lambda = \max_{i \in \mathcal{R}} \{\lambda_i\}$ represents the dissemination length of the transfer plan. Its domain is therefore $\bigcup_{i \in \mathcal{R}} \mathcal{T}_i \subseteq \{0, 1, \ldots, m\}$.

Minimizing the dissemination length leads to the following objective.

$$\lambda^* = \min \lambda \tag{4.1}$$

The constraints are written as follows (note that these constraints alone are **not** sufficient to ensure that the transfer plan is Γ-robust):

- Each node $i \in \mathcal{N}$ initially possesses a subset $\mathcal{O}_i$ of datum units:

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \in \mathcal{O}_i,\ y_{i,k,0} = \Gamma + 1 \tag{5.2}$$
$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \notin \mathcal{O}_i,\ y_{i,k,0} = 0 \tag{3.5}$$

- The transfer plan must be valid (sending nodes must possess the datum units that they transmit):

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ x_{k,c} \leq y_{s_c,k,\mathcal{T}_i(c-1)} \tag{3.6}$$

- Nodes possess a datum unit from the time they receive it:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ y_{r_c,k,\mathcal{T}_i(c-1)} + x_{k,c} = y_{r_c,k,c} \tag{3.10}$$

- At most one datum unit can be transferred during each contact:

$$\forall c \in \{1, 2, \ldots, m\},\ \sum_{k \in \mathcal{D}} x_{k,c} = a_c \tag{4.2}$$

- $\lambda_{i,k}$ is the delivery length associated with datum unit k and node i:

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D},\ \forall t \in \mathcal{T}_i,\ \begin{cases} y_{i,k,t} < \Gamma + 1 \iff \lambda_{i,k} > t \\ y_{i,k,t} = \Gamma + 1 \iff \lambda_{i,k} \leq t \end{cases} \tag{5.3}$$

Implicit constraints

No constraints explicitly ensures that the transfer plan is Γ-robust. Actually, this is implicitly done through the branching stage. As mentioned in Section 5.2.2, to solve the problem, we sequentially set the transfers. It corresponds to set x-variables from $x_{k,1}$ ($\forall k \in \mathcal{D}$) to $x_{k,m}$.

1. At each node of the search tree, the earliest transfer $\phi(c)$ which is not yet set – *i.e.* the smallest index $c \in \{1, \ldots, m\}$ such that $\exists k \in \mathcal{D}$ where $x_{k,c}$ is not yet fixed – is selected.
2. Next, one branch is created for each robust-improving transfer – *i.e.* for each datum unit $k \in O_{s_c}^{c-1}$, if transfer $\phi(c) = \{k\}$ is robust-improving, then one adds a branch where $x_{k,c} = 1$ and $\forall p \in \mathcal{D} \backslash \{k\}$, $x_{p,c} = 0$.
3. If no robust-improving transfers have been found, then $\phi(c)$ is set to $\emptyset$ – *i.e.* $\forall k \in \mathcal{D}$, we set $x_{k,c} = 0$.

To check whether a given transfer is robust-improving, we use the test defined in Section 5.2.2. Note that the capacities of the transportation network are given by the x-variables for transfers occurring before $\phi(c)$, and must be set to 0 for other transfers.

In this way, we ensure that the transfer plan is $\Gamma-$*robust* (because other constraints ensure that each recipient receives $\Gamma + 1$ copies of each unit).

In addition, this approach ensures that the transfer plan is *robust-minimal* (every transfer is either null or robust-improving), and *robust-strictly-active* (because robust-improving transfers cannot be postponed).

Remark 5.2

A transfer $\phi(c) = \{k\}$ is necessarily robust-improving if there exist more arc-disjoint journeys from a source of datum unit k to node s_c than from a source of k to node r_c. Consequently, in such a case, no flow has to be computed and one can set $a_c = 1$. In practice, note that constraint (4.4) can be used (*cf.* page 100).

5.3.2 Additional features

As regards the robust dissemination problem, key concepts and models have been discussed. However, like in Section 4.2, *ad hoc* algorithms can speed up the solving process. Below we propose an additional propagation procedure, and adaptations of the techniques developed for the case where $\Gamma = 0$.

Look-ahead

In some circumstances, the propagation engine can reveal necessary transfers even before the sequential branching algorithm have to chose a value for those transfers (by combining some constraints of the model). The corresponding x-variables are then set to 1. For example, on Figure 5.2a, the solver might deduce that $\phi(6) = \{1\}$ and $\phi(9) = \{1\}$ at the start (since there are only two contacts left for sending the two required copies of unit 1 to node 6).

In the following, we study such a situation. Formally, we assume the first $t \in \{0, \dots, m-1\}$ transfers have been set. Thus, $\phi(t+1)$ is the first unfixed transfer. We then consider a transfer $\phi(c) = \{k\}$, $c \in \{t+2, \dots, m\}$, $k \in \mathcal{D}$, that has been set by a third propagation procedure.

Unfortunately, we cannot guarantee, at this stage, that transfer $\phi(c)$ will necessarily be robust-improving, or even that a dominant solution such that transfers $\phi(c)$ is robust-improving exists. This may depend on some transfers $\phi(x)$, $x \in \{t+1, \dots, c-1\}$, which are not yet set. However, we can already check that $\phi(c)$ has a chance of being robust-improving. To this end, we use the transportation network $G(\phi, k, r = r_c)$ (*cf.* Section 5.2.1) restricted to:

1. all contacts σ_x, $x \leq t$, such that $x_{k,x} = 1$, *i.e.* such that $\phi(x) = \{k\}$;
2. contacts σ_x, $x \in \{t+1, \dots, c\}$, such that $r_x = r_c$ and $x_{k,x} = 1$ (because we must show that these contacts can be robust-improving);
3. contacts σ_x, $x \in \{t+1, \dots, c-1\}$, such that $r_x \neq r_c$ and $x_{k,x}$ is not set to 0, *i.e.* such that $\phi(x)$ might/must be equal to $\{k\}$.

In fact, we consider the transportation graph associated with the hypothetical transfer plan ϕ where all the available contacts are leveraged for transmitting unit k. Obviously, if we cannot find enough arc-disjoint journeys with such a transfer plan – that is, if we cannot find one augmenting path in this flow network for each contact σ_x, $x \in \{1, 2, \dots, c\}$, such that $r_x = r_c$ and $x_{k,x} = 1$ – then no dominant solution can be computed (developed) from this node of the search tree (the transfers towards node r_c cannot *all* be robust-improving at the same time). If so, a backtrack must be triggered immediately.

If we look back at Figure 5.2a, the corresponding transportation network is given in Figure 5.2b. Contacts σ_x, $x \in \{1, 2, 3, 4, 5, 7, 8\}$, may be used for transferring unit 1 to node 6, so the capacity of the arcs associated with these contacts is 1. Conversely, contact σ_{10} will be used for sending datum unit 2 to node 5, so the capacity of arc $(3_1, 5_2)$ is set to 0 (this arc is removed from the flow network). We can find an augmenting path for transfer $\phi(6)$ and for transfer $\phi(9)$ (the two transfers fixed in advance) : $[\, src, 1_0, 2_3, 4_1, 6_1, 6_2, snk\,]$ and $[\, src, 1_0, 2_1, 2_2, 2_3, 5_1, 6_2, snk\,]$. Thus we cannot prune this branch.

Remark 5.3

Note that contacts σ_x, $x \in \{t+1, \ldots, c-1\}$, such that $\boldsymbol{r_x = r_c}$ and $x_{k,x}$ is still free (neither set to 0, nor set to 1), are ignored (we set the capacity of the corresponding arcs to 0). In fact, considerings these contacts would over-constrain the flow, because we would have to find augmenting paths for transfers that are not required to be robust-improving.

In practice, it may be hard to prune a node in such a way, because there may be many available contacts, and therefore potentially many augmenting paths. Nevertheless, computing these maximum flows also provides valuable information regarding the possible transfers. More specifically, any arc that belongs to a minimum cut in the transportation network described above is necessary for the flow to be maximum – *i.e.* for the transfers fixed in advance to be robust-improving.

In Figure 5.2b, the minimum cut given by the hatched vertices – namely $\{(2_3, 4_1), (2_3, 5_1)\}$ – reveals that we must use contacts σ_5 and σ_7 to transmit datum unit 1 to nodes 4 and 5 respectively. Thus, $x_{1,5} = 1$ and $x_{1,7} = 1$.

Proposition 5.6

Let $M = |\{\, \sigma_x \in \{1, \ldots, c\}$ such that $r_x = r_c$ and $x_{k,x} = 1\,\}|$ refer to the required value of a maximum flow in $G(\phi, k, r_c)$ (when only considering the contacts listed above).

1. The flow value cannot exceed M (by construction).
2. If the value of a maximum flow is M, then every arc belonging to a minimum cut of $G(\phi, k, r_c)$ corresponds to a contact during which datum unit k must be transmitted.
3. Finally, if the value of a maximum flow is lower than M, then the current branch cannot lead to a dominant solution.

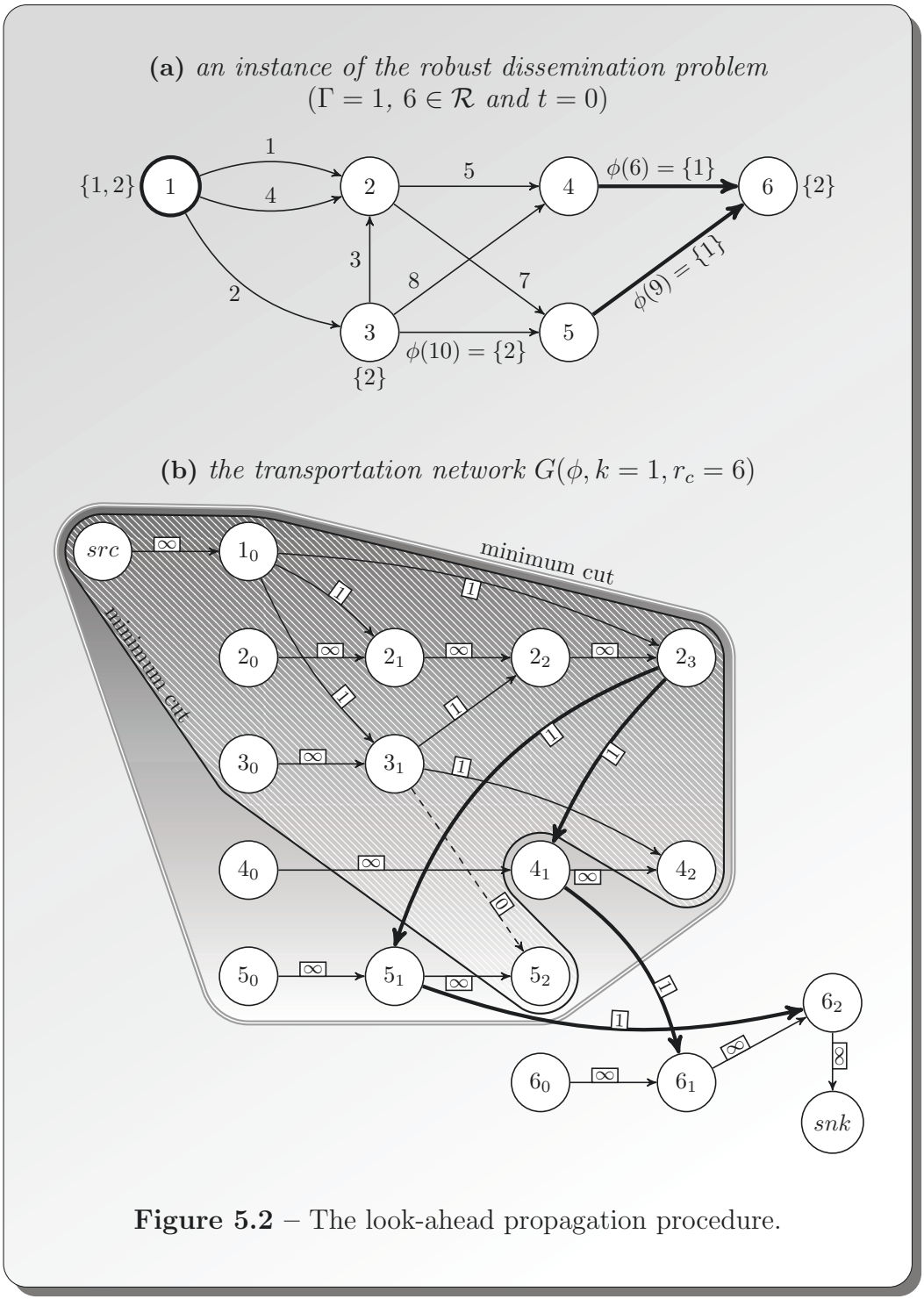


Figure 5.2 – The look-ahead propagation procedure.

Proof. (item.2) Let C be a minimum cut in $G(\phi, k, r_c)$. Let $a \in C$ be an arc of that cut. We assume that there exists a dominant transfer plan (developed from the current node in the search tree) such that the contact corresponding to arc a is not leveraged for transferring datum unit k. Therefore, there is a flow of value M in $G(\phi, k, r_c) - \{a\}$ (*i.e.* the graph obtained after removing arc a in $G(\phi, k, r_c)$). This flow is maximum. Yet, $C - \{a\}$ is a cut of capacity $M - 1$ in graph $G(\phi, k, r_c) - \{a\}$ – which contradicts the max-flow/min-cut theorem. □

In practice, we enumerate the minimum cuts with the recursive algorithm proposed by Balcioglu and Wood [4]. Therefore, this propagation procedure runs in $O\left(M(n+m) + K(n+m)^2\right)$ time, with $M = |\{\, \sigma_x \in \{1, \ldots, c\}$ such that $r_x = r_c$ and $x_{k,x} = 1 \,\}|$. K denotes the number of maximum cuts in the flow network. As K may be huge, recursion depth should be limited.

Lower bounds and symmetry-breaking techniques

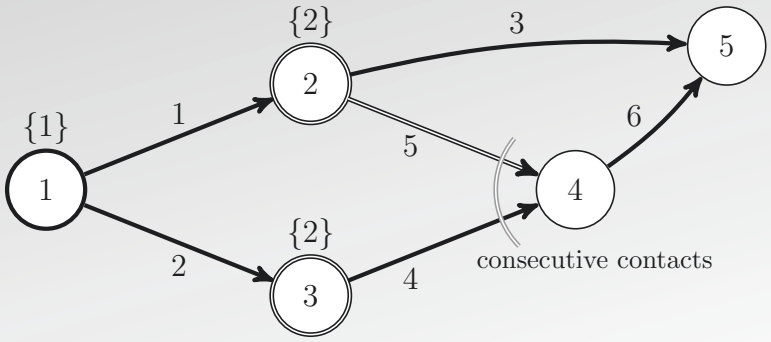
Almost all *ad hoc* methods proposed in Section 4.2 can be generalised to the robust case straightforwardly. Actually, almost all proposed methods remain applicable on condition that we consider the datum units *and* their copies – *i.e.* on condition that we correctly take into account that each recipient has to receive $\Gamma + 1$ copies of each datum unit, instead of one.

For example, the first lower bound becomes:

Proposition 5.7

Let $i \in \mathcal{R}$ be a recipient node, and $\alpha = (\Gamma+1) \times (u - |\mathcal{O}_i|)$ be the number of datum units (including copies) that i has to receive during a transfer plan. Let $\sigma_x \in \sigma$ denote the α^{th} contact $\sigma_c = (s_c, i)$ during which a unit $k \in \mathcal{D} \backslash \mathcal{O}_i$ can be transferred to node i (*i.e.* such that variable $x_{k,c}$ is not set to 0). If this index does not exist (if it remains less than α contacts fulfilling the condition), we consider that $x = \infty$. x is a lower bound for $\lambda_i^{\Gamma}(\phi)$ and $\lambda^{\Gamma}(\phi)$ ($\lambda \geq \lambda_i \geq x$ in the model).

For further details, we refer to Section 4.2 at page 102. Note that one can still use the lower bound defined in Proposition 4.1; the lower bound defined in Proposition 4.2; the symmetry-breaking technique defined in Proposition 4.3; and the nogood-recording technique defined in Propositon 4.5 – that is everything but the symmetry-breaking technique defined in Proposition 4.4 (*cf.* Figure 5.3 and Remark 5.4 for more details).



$$\phi(1) = \phi(2) = \phi(3) = \phi(4) = \phi(6) = \{1\} \text{ and } \phi(5) = \{2\}$$

Figure 5.3 – Symmetry-breaking techniques, consecutive contacts.

Remark 5.4

In Proposition 4.4, page 108, we study the situation where a node is the receiving node in several contacts, without having any possibility to send data to a third node between these contacts. We propose to enforce that datum units are transferred, in such a situation, in order of their index. For instance, in Figure 4.3, page 109 ($\Gamma = 0$), considering

$$[\,\phi(1) = \{1\} \text{ and } \phi(2) = \{2\}\,] \textbf{ or } [\,\phi(1) = \{2\} \text{ and } \phi(2) = \{1\}\,]$$

leads to equivalent states (all nodes possess the same units). Therefore, considering one option among the two is sufficient to solve the problem. Unfortunately, in Figure 5.3 ($\Gamma > 0$), the "symmetric" transfer plan

$$[\,\phi(4) = \{2\} \text{ and } \phi(5) = \{1\}\,]$$

does not lead to the same state – because transfer $\phi(6) = \{1\}$ no longer is robust-improving (the two journeys associated with node 5 and datum unit 1 relies on contact σ_1).

5.4 Preliminary results

In this section, experimental results are reported and discussed. First of all, we describe our benchmark. The latter is built from the instances generated for the non-robust dissemination problem. Next we study the results obtained with the different algorithms described in Section 5.3. All experiments were performed in the same conditions as for Chapters 3 and 4 – *i.e.* on the same machine, with the same parameters and the same one-hour time limit.

5.4.1 About the benchmark

In order to generate hard instances, we used the instances generated for the dissemination problem, *cf.* Section 3.5.1 at page 85.

Given Propositions 5.1 and 5.2, one can expect an instance of the robust dissemination problem to be as difficult as an instance of the dissemination problem characterised by about Γ times more units (if we keep constant the number of nodes and the number of contacts). Consequently, we propose to reuse the instances generated for the dissemination problem, by reducing the number of datum units in such a way that the old number of datum units is of the same order of magnitude as $(\Gamma + 1) \times u$ in the new instance. Actually, we must find a well-known compromise:

1. between a larger value of $(\Gamma + 1) \times u$ – feasible transfer plans are harder to be found, but proofs of infeasibility are usually easier;
2. and a smaller value of $(\Gamma + 1) \times u$ – feasible transfer plans are easier to be found, but proofs of infeasibility may be harder.

This compromise must ensure that it is difficult to prove that a transfer plan is optimal.

The new benchmark is described in Table 5.1. As a reminder, the classes are characterised by the number *nbinst* of instances it contains, the required level of robustness Γ, the number of datum units u, and the number of nodes n of these instances. The average number $\overline{rec}$ of recipients $i \in \mathcal{R}$, the average number $\overline{src}$ of source nodes $i \in \mathcal{N}$ such that $\mathcal{O}_i \neq \emptyset$, and the average number $\overline{m}$ of contacts in every class are also reported. In column *from*, we indicate the classe(s) (originally used for the dissemination problem) from which the new instances are built (for the robust dissemination problem).

name	*from*	*nbinst*	Γ	u	n	$\overline{rec}$	$\overline{src}$	$\overline{m}$
1r2u20n	4u20n	41	1	2	20	18	1	366
1r2u50n	4u50n+5u50n	49	1	2	50	44	1	717
1r2u100n	4u100n	20	1	2	100	87	2	1720
1r5u10n	10u10n	16	1	5	10	6	2	197
1r25u10n	50u10n	16	1	25	10	6	2	750
1r50u10n	100u10n	6	1	50	10	7	4	2000
2r2u50n	2r5u50s	23	2	2	50	50	1	726
2r3u10n	10u10n	16	2	3	10	6	2	197
2r17u10n	50u10n	16	2	17	10	6	2	750
2r34u10n	100u10n	6	2	34	10	7	4	2000
3r2u10n	10u10n	16	3	2	10	6	2	197
3r13u10n	50u10n	16	3	13	10	6	2	750
3r25u10n	100u10n	6	3	25	10	7	4	2000

Table 5.1 – The benchmark generated for the robust dissemination problem.

5.4.2 Numerical results

Let us have a look at column **none** in Table 5.2. This contains the numerical results obtained by considering the model of Section 5.3.1, and the branching algorithm described in Section 5.2.2, that is the minimum needed for solving the robust dissemination problem (with no additional features).

We note that only 13% of the benchmark has been solved to optimality, but feasible (robust) transfer plans have been found for approximatively 77% of the remaining instances. It shows that the benchmark is quite challenging, and that the solver (namely CP-Optimizer) can run into difficulties, even for small instances.

To achieve better results, we need to consider the additional propagation algorithms proposed in Section 5.3.2:

- the weak (**wlb**) or the strong (**slb**) lower bounds;
- the symmetry-breaking technique (**sym**);
- the nogood-recording (**ngr**);
- and/or the look-ahead procedure (**la**).

In practice, by activating each feature one by one, we noticed that the weak lower bound is the element that has the most impact on the performances of the solver – *cf.* column **wlb** in Table 5.2. However, we may wonder whether the strong lower bound is really relevant for this problem, because we observe that the percentage of instances solved proven to optimality tends to be worse in column **slb** than in column **wlb**.

Nogood-recording and symmetry-breaking techniques have also appeared to be very effective. Using these two features (together with the weak lower bound) enable more than 70% of the benchmark to be solved – see columns **wlb+ngr** and **wlb+ngr+sym** in Table 5.3. The impact of the look-ahead procedure is certainly more limited, see column **wlb+ngr+sym+la**, but is worth mentioning.

Note finally that the best numerical results were achieved with a variant of the look-ahead procedure, termed "weak-look-ahead". This consists in not executing the look-ahead algorithm more than once, for a *same* transfer in a *same* branch – *i.e.* we check that a transfer fixed in advance has a chance to be robust-improving *only* the first time we detect it. The results are reported in Table 5.4.

	none			wlb			slb		
	solved	*feas*	*cpu*	*solved*	*feas*	*cpu*	*solved*	*feas*	*cpu*
1r2u20n	73.2	26.8	1171	78.0	22.0	882	75.6	24.4	1259
1r2u50n	12.2	83.7	3169	14.3	81.6	3096	14.3	81.6	3133
1r2u100n	20.0	65.0	2881	25.0	60.0	2701	25.0	60.0	2701
1r5u10n	6.3	62.5	3377	31.3	37.5	2477	31.3	37.5	2484
1r25u10n	0.00	93.8	3601	43.8	50.0	2027	43.8	50.0	2046
1r50u10n	0.00	100	3604	50.0	50.0	1811	50.0	50.0	1850
2r2u50n	0.00	60.9	3601	21.7	39.1	2933	34.8	43.5	2349
2r3u10n	12.5	75.0	3303	50.0	37.5	1918	50.0	37.5	1979
2r17u10n	0.00	93.8	3601	50.0	43.8	1818	50.0	43.8	1963
2r34u10n	0.00	100	3603	50.0	50.0	1808	50.0	50.0	1828
3r2u10n	43.8	56.3	2111	68.8	31.3	1239	56.3	37.5	1670
3r13u10n	0.00	87.5	3601	56.3	31.3	1640	50.0	37.5	1825
3r25u10n	0.00	100	3602	50.0	50.0	1807	50.0	50.0	1821
average	12.9	77.3	3171	45.3	44.9	2012	44.7	46.4	2070

Table 5.2 – Computational results obtained with CP Optimizer – *part*-1.

	wlb+ngr			wlb+ngr+sym			wlb+ngr+sym+la		
	solved	*feas*	*cpu*	*solved*	*feas*	*cpu*	*solved*	*feas*	*cpu*
1r2u20n	100	-	0.41	100	-	0.31	100	-	0.23
1r2u50n	67.3	30.6	1310	73.5	24.5	1144	81.6	16.3	857
1r2u100n	55.0	45.0	1622	60.0	40.0	1506	65.0	35.0	1366
1r5u10n	43.8	37.5	2090	56.3	25.0	1586	56.3	25.0	1581
1r25u10n	43.8	50.0	2027	50.0	37.5	1814	50.0	37.5	1814
1r50u10n	50.0	50.0	1810	83.3	16.7	620	83.3	16.7	620
2r2u50n	34.8	34.8	2400	34.8	34.8	2369	43.5	30.4	2336
2r3u10n	87.5	12.5	798	93.8	6.3	311	93.8	6.3	273
2r17u10n	50.0	43.8	1816	68.8	25.0	1180	68.8	25.0	1185
2r34u10n	50.0	50.0	1807	66.7	33.3	1216	66.7	33.3	1211
3r2u10n	100	-	0.28	100	-	0.20	100	-	0.20
3r13u10n	56.3	31.3	1603	62.5	25.0	1351	62.5	25.0	1351
3r25u10n	50.0	50.0	1807	66.7	33.3	1207	66.7	33.3	1208
average	60.6	33.5	1469	70.5	23.2	1100	72.2	21.8	1062

Table 5.3 – Computational results obtained with CP Optimizer – *part*-2.

5.5 Conclusion

In this chapter, we addressed the problem of finding a Γ-robust transfer plan, *i.e.* a valid transfer plan which guarantees that the recipient nodes correctly receive all datum units, even if some transfers (at most Γ) fail. We hope that this approach will manage to cover a wide range of constraints that are to be found in real applications.

We proposed an algorithm based on constraint programming. This relies on a necessary and sufficient condition for a transfer plan to be Γ-robust, *cf.* Proposition 5.2. Note, moreover, that this algorithm is an adaptation of the procedure proposed in Chapter 4 for the original dissemination problem.

Finally, promising results were reported. It appeared that specific *ad hoc* propagation algorithms – *e.g.* some lower bounds – are *required* for the model to be efficiently solved (even more than before).

	wlb+ngr+sym+wla		
	solved	*feas*	*cpu*
1r2u20n	100	-	0.20
1r2u50n	81.6	16.3	835
1r2u100n	60.0	40.0	1502
1r5u10n	56.3	25.0	1582
1r25u10n	50.0	37.5	1815
1r50u10n	83.3	16.7	620
2r2u50n	39.1	30.4	2354
2r3u10n	93.8	6.3	251
2r17u10n	68.8	25.0	1150
2r34u10n	83.3	16.7	1025
3r2u10n	100	-	0.20
3r13u10n	62.5	25.0	1351
3r25u10n	66.7	33.3	1207
average	72.7	20.9	1053

Table 5.4 – The best results achieved with CP-optimizer.

6 Conclusion and perspectives

In this thesis, we addressed the problem of making use of knowledge about node mobility when information must be routed throughout an *intermittently-connected network*, *i.e.* a delay-tolerant network (DTN) or a system of systems. We sought *store-forward* routings – termed transfer plans – which enable a set of recipient nodes to receive some data from a set of source nodes, when a sequence of *contacts* (an opportunity for two nodes to communicate) can be reliably estimated. In practice, this took the shape of a combinatorial problem, termed *the dissemination problem*.

Our contributions are organised into three parts.

1. First, we formally defined the "dissemination problem". This step was essential since the literature was lacking a clear and unified framework for such problems. In addition, the literature did not properly consider the case where *identified* and *indivisible* pieces of data (termed *datum units*) need to be routed from *one or several* sources, to *one or several* recipients. Next we proved that the problem is NP-Hard in the strong sense. We also highlighted some polynomial cases. Another interesting point is that the dissemination problem was shown to be equivalent to finding mutually arc-disjoint branchings in an evolving graph. In fact, all of this corresponds to Chapters 1 and 2.
2. The second part comprises more technical elements. More specifically, we proposed two solving schemes. The first one relies on integer-linear programming, and was defined in Chapter 3. The second one relies on constraint programming, and was defined in Chapter 4. Both of them

are based on the dominance rules that we proposed beforehand. These were leveraged to build efficient preprocessing procedures, and to define extra constraints. In addition, we proposed specific *ad hoc* propagation algorithms for the constraint-programming model to be more efficiently solved. For example, one of these algorithms is based on a lower bound of the dissemination length. Note that constraint programming has led to the best experimental results on a self-generated benchmark.

3. Finally, we study a variant of the dissemination problem – termed the *robust dissemination problem* – where we need to find Γ-robust transfer plans, *i.e.* valid transfer plans which guarantee that the recipient nodes correctly receive all datum units, even if any Γ transfers fail. We hope that this approach will manage to cover a wide range of constraints of real applications. To tackle this problem, we proposed a necessary and sufficient condition for a valid transfer plan to be Γ-robust. Thereafter we adapted the solving scheme based on constraint programming (and proposed for the original problem) in order to take account of this new constraint. All of this is discussed in Chapter 5.

This work is still underway.

Concerning the robust dissemination problem, we still have to propose a more specific benchmark, with few but well-characterized classes and groups (like we did for the dissemination problem). For this purpose, we propose to follow the promising approach discussed in Section 5.4.1.

At the same time, we must develop/implement a preprocessing procedure for the robust dissemination problem. We are currently adapting the one we proposed for the initial dissemination problem to that end.

Finally, we think that further research on robust optimization would help prediction errors to be more effectively managed in practice (which seems to us to be a crucial point in a real context). For example, instead of considering that some contacts may fail, we could also envision that contacts occur in an uncertain order, or even that some nodes may be destroyed. Nodes deployed in hostile (*e.g.* military) environments are indeed prone to such risks.

This being said, many other constraints are also worth considering – *e.g.* buffers and/or batteries limitations, transmission and/or propagation delays, interferences, *etc.*

Bibliography

[1] I. F. Akyildiz, O. B. Akan, C. Chen, J. Fang, and W. Su. InterPlaNetary Internet: state-of-the-art and research challenges. *Computer Networks*, 43(2):75–112, 2003.

[2] J. Alonso and K. Fall. A Linear Programming Formulation of Flows over Time with Piecewise Constant Capacity and Transit Times. Technical report, Intel Research, Berkeley, 2003.

[3] E. Altman, G. Neglia, F. De Pellegrini, and D. Miorandi. Decentralized Stochastic Control of Delay Tolerant Networks. In *The 28th Conference on Computer Communications, IEEE INFOCOM 2009*, pages 1134–1142. IEEE, 2009.

[4] A. Balcioglu and K. R. Wood. Enumerating Near-Min S-T Cuts. In D. L. Woodruff, editor, *Network Interdiction and Stochastic Integer Programming*, volume 22 of *Operations Research / Computer Science Interfaces Series*, pages 21–49. Springer US, 2003.

[5] N. Belblidia, M. Dias De Amorim, L. H. M. K. Costa, J. Leguay, and V. Conan. PACS: Chopping and shuffling large contents for faster opportunistic dissemination. In *8th International Conference on Wireless On-Demand Network Systems and Services, WONS 2011*, pages 9–16. IEEE, 2011.

[6] D. Bertsimas and M. Sim. The Price of Robustness. *Operations Research*, 52(1):35–53, 2004.

[7] S. Bhadra and A. Ferreira. Computing multicast trees in dynamic networks and the complexity of connected components in evolving graphs. *Journal of Internet Services and Applications*, 3(3):269–275, 2012.

[8] R. Bocquillon and A. Jouglet. Data Transfer in Delay-Tolerant Networks. In *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA'2013*, pages 355–359. IEEE, 2013.

[9] R. Bocquillon and A. Jouglet. Minimizing the dissemination length in the one-datum-unit data transfer problem. In *2013 Sixth Multidisciplinary International Conference on Scheduling, Theory and Applications, MISTA'13*, 2013.

[10] R. Bocquillon and A. Jouglet. A constraint-programming-based approach to solve the data dissemination problem. *submitted*, 2015.

[11] R. Bocquillon and A. Jouglet. Modeling elements and solving techniques for the data dissemination problem. *submitted*, 2015.

[12] R. Bocquillon, A. Jouglet, and J. Carlier. The data transfer problem in a system of systems. *European Journal of Operational Research*, 244(2):392–403, 2015.

[13] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations, 2001.

[14] B. Burns, O. Brock, and B. N. Levine. MV routing and capacity building in disruption tolerant networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 398–408. IEEE, 2003.

[15] J. Cheriyan and M. R. Salavatipour. Hardness and approximation results for packing steiner trees. *Algorithmica*, 45(1):21–43, 2006.

[16] The delay-tolerant networking research group. `http://www.dtnrg.org/`.

[17] J. Edmonds. *Combinatorial algorithms*, chapter Edge-disjoint branchings, pages 91–96. Algorithmic Press, New York, NY, USA, R. Rustin ed., 1972.

[18] J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19(2):248–264, 1972.

[19] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[20] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM 2003*, page 27, New York, NY, USA, 2003. ACM Press.

[21] A. Ferreira. On models and algorithms for dynamic communication networks: The case for evolving graphs. In *4e rencontres francophones sur les Aspects Algorithmiques des Télécommunications, ALGOTEL 2002*, pages 155–161. INRIA Press, 2002.

[22] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.

[23] A. Ferreira and A. Jarry. Complexity of Minimum Spanning Tree in Evolving Graphs and the Minimum-Energy Broadcast Routing Problem. In *Proceedings of WiOpt'04, Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2004.

[24] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[26] R. Handorean, C. Gill, and G.-C. Roman. Accommodating Transient Connectivity in Ad Hoc and Mobile Settings. In *Pervasive Computing, proceedings of the Second International Conference, PERVASIVE 2004*, volume 3001 of *Lecture Notes in Computer Science*, pages 305–322. Springer Berlin Heidelberg, 2004.

[27] D. Hay and P. Giaccone. Optimal routing and scheduling for deterministic delay tolerant networks. In *2009 Sixth International Conference on Wireless On-Demand Network Systems and Services*, pages 27–34. IEEE, 2009.

[28] Instances of the dissemination problem. `https://www.hds.utc.fr/~rbocquil/dokuwiki/_media/dp_instances.zip`.

[29] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. *ACM SIGCOMM Computer Communication Review*, 34(4):145, 2004.

[30] M. Jamshidi. *System of Systems Engineering: Principles and Applications*. Boca Raton, Taylor & Francis, 2008.

[31] A. Jouglet and J. Carlier. Dominance rules in combinatorial optimization problems. *European Journal of Operational Research*, 212(3):433–444, 2011.

[32] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking. *ACM SIGARCH Computer Architecture News*, 30(5):96, 2002.

[33] J. LeBrun and C.-N. Chuah. Bluetooth content distribution stations on public transit. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking, MobiShare 2006*, page 63, New York, NY, USA, 2006. ACM Press.

[34] A. Lindgren, A. Doria, and O. Schelén. Probabilistic Routing in Intermittently Connected Networks. *SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, 2003.

[35] S. Merugu, M. Ammar, and E. Zegura. Routing in Space and Time in Networks with Predictable Mobility. Technical report, Georgia Institute of Technology, 2004.

[36] M. Mongiovi, A. K. Singh, X. Yan, B. Zong, and K. Psounis. Efficient multicasting for delay tolerant networks using graph indexing. In *2012 Proceedings IEEE INFOCOM*, pages 1386–1394. IEEE, 2012.

[37] Labex MS2T "Control of Technological Systems-of-Systems". `https://www.hds.utc.fr/labex-ms2t-484/`.

[38] A. Pentland, R. Fletcher, and A. Hasson. DakNet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, 2004.

[39] T. Schiex and G. Verfaillie. Nogood Recording for static and dynamic constraint satisfaction problems. In *Proceedings of 1993 IEEE Conference on Tools with Al (TAI-93)*, pages 48–55. IEEE Comput. Soc. Press, 1993.

[40] Y. Shiloach. Edge-disjoint branching in directed multigraphs. *Information Processing Letters*, 8(1):24–27, 1979.

[41] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks*. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010.

[42] A. Vahdat and D. Becker. Epidemic Routing for Partially Connected Ad Hoc Networks. Technical report, Duke University, 2000.

[43] A. Voyiatzis. A Survey of Delay- and Disruption-Tolerant Networking Applications. *Journal of Internet Engineering*, 5(1), 2012.

[44] W. D. Wood, Lloyd Ivancic, W. M. Eddy, D. Stewart, J. Northam, C. Jackson, and A. da Silva Curiel. Use of the Delay-Tolerant Networking Bundle Protocol from Space. In *59th International Astronautical Congress and Exhibition*, Glasgow, Scotland, United Kingdom, 2008.

[45] B. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[46] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *IEEE Communications Surveys & Tutorials*, 8(1):24–37, 2006.

[47] W. Zhao, M. Ammar, and E. Zegura. Multicasting in delay tolerant networks. In *Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking, WDTN 2005*, pages 268–275, New York, New York, USA, 2005. ACM Press.