A constraint-programming-based approach for solving the data dissemination problem

Ronan Bocquillon, Antoine Jouglet

A constraint-programming-based approach for solving the data dissemination problem

Ronan Bocquillon, Antoine Jouglet

Sorbonne Universités, Université de Technologie de Compiègne, CNRS, Heudiasyc UMR 7253, CS 60319, 60203 Compiègne cedex

{ *ronan.bocquillon ; antoine.jouglet* } *@hds.utc.fr*

Abstract

Systems of mobile Systems (SoS) are intermittently connected networks that use store-carry-forward routing for data transfers. Heterogeneous independent systems collaborate and exchange data to achieve a common goal. Data transfers are only possible where systems are close enough to each other, when a so-called *contact* occurs. During a contact, a sending system can transmit to a receiving system a fixed amount of data held in an internal buffer. We assume that the trajectories of component systems are predictable and consequently that a sequence of contacts may be considered. The dissemination problem is finding a transfer plan such that a set of data can be completely transferred from a given subset of source systems to all the recipient systems. In this paper we propose a new constraint-programming-based algorithm for solving this problem. Computational results show that this approach is an improvement on the integer-linear-programming-based approach that we proposed in a previous paper.

Keywords: combinatorial optimization, constraint programming, lower bounds, symmetry-breaking techniques, systems of systems, data transfer problem

1. Introduction

Mo Jamshidi has defined *Systems of Systems* (SoS) as large-scale integrated systems which are heterogeneous and independently operable, but which are networked together for a common goal [1]. To achieve this goal, the (sub)systems must collaborate and share data. When the systems are mobile and intermittently connected, they must opportunistically make use of contacts that arise when two systems are close enough to each other. This sort of opportunistic use of contacts becomes critical when contact durations are relatively short with respect to the volume of information being routed through the SoS. Via a series of separate contacts, elements of data are transferred from one system to another until their final destination is reached (this is called *store-carry-forward* routing).

Data transfer problems within systems of systems are addressed in the literature, in both opportunistic [2] and delay-tolerant networking [3]. In general, the mobility of systems is assumed to be non-deterministic, although realistic predictions can be made in many applications, *e.g.* satellite networks (where the trajectories of the component systems obey straightforward dynamic laws), data transfers in public transportation systems [4], and data transfers between drones.

In this paper we look at how knowledge about collaboration opportunities [5, 6] may be harnessed when routing data from sources to destinations inside a given time horizon. In a previous paper [7] we proposed several dominance rules, different deduction procedures, a promising preprocessing algorithm, and an integer-linear-programming model for solving this *dissemination problem* (formalized in Section 2). Here we extend this previous work and propose a constraint-programming-based algorithm to solve the problem more efficiently. For a state-of-the-art we refer the reader to [7].

The remainder of the paper is organized as follows. In Section 2, we briefly restate the problem. In Sections 3 and 4, we propose a constraint-programming-based algorithm for solving it. In Section 5 we assess this new approach and compare it with the approach we proposed in [7]. The paper ends with a conclusion and prospects.

2. The dissemination problem

In this section we first recall the statement of the dissemination problem, then summarize the dominance rules we proposed in [7]. These dominance rules enable the search space to be significantly reduced.

2.1. Formal description

First of all, we consider a set $\mathcal{N} = \{1, 2, \ldots, q\}$ of q interacting mobile systems, the *nodes*, and one *datum* $\mathcal{D} = \{1, 2, \ldots, u\}$ of u datum units. Each *datum unit* represents a unitary, indivisible fragment of data. Each node $i \in \mathcal{N}$ possesses a subset $O_i \subseteq \mathcal{D}$ of datum units from the outset. Subset $\mathcal{R} \subseteq \mathcal{N}$ defines the nodes wishing to obtain the datum $\mathcal{D}$ (all the datum units) inside the given time horizon. In this paper the term *source nodes* will refer to the nodes $i \in \mathcal{N}$ such that $O_i \neq \emptyset$. The nodes in $\mathcal{R}$ are termed the *recipients*.

To ensure the dissemination of the datum $\mathcal{D}$, nodes may exchange datum units whenever they are close enough to communicate. A communication opportunity is known as a *contact*. We assume that these contacts are perfectly known, or easily predictable at any time, because 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 transmit 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 $\sigma_c = (s_c, r_c) \in \sigma$.

A *transfer plan* $\phi : \{1, 2, \ldots, m\} \mapsto \{\emptyset, \{1\}, \{2\}, \ldots, \{u\}\}$ 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 transferred during contact σ_c. Hereinafter, T_ϕ denotes the target set $\{\emptyset, \{1\}, \{2\}, \ldots, \{u\}\}$ of ϕ. A transfer plan ϕ has a corresponding set of states $O_i^t \subseteq \mathcal{D}$, defined for each time index $t \in \{0, 1, \ldots, m\}$ and each node $i \in \mathcal{N}$, such that:

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

Each state O_i^t therefore 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 start). The transfer plan is *valid* where nodes transmit only datum units that they possess, *i.e.*

$$\forall \sigma_c \in \sigma,\ \phi(c) \in \{\emptyset\} \cup \{\{k\} \,|\, k \in O_{s_c}^{c-1}\} \tag{2}$$

(a) *instance definition*

$$\begin{cases} \mathcal{N} = \mathcal{R} = \{1, 2, \ldots, 6\}; \mathcal{D} = \{1, 2\} \\ O_1 = \{1, 2\}; O_2 = \{1\}; O_3 = \cdots = O_6 = \emptyset \\ \sigma = [(1, 6), (6, 1), (6, 5), \ldots, (5, 6)] \end{cases}$$

(b) *a solution of 3 arc-disjoint branchings*

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

(c) *evolving graph [9]*

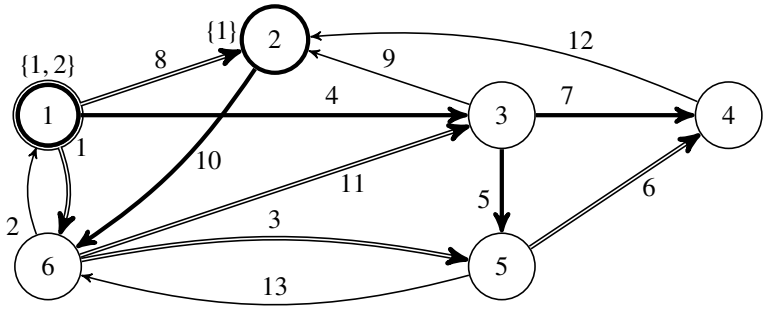

Figure 1: An instance of the problem, a solution and the corresponding evolving graph.

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

This problem is known to be NP-Hard in the strong sense, but it can be polynomially solved if $u = 1$ or $|\mathcal{R}| = 1$ (see [8] for a comprehensive study on the complexity of this problem).

Note that instances of the problem can be described by evolving graphs, *i.e.* multigraphs whose vertices represent nodes and whose arcs represent connections between these nodes. Each arc is labelled with time intervals which indicate when the link is active. To properly 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, each contact is thus represented by an arc whose label is given by its position in the sequence.

In Figure 1, $[\sigma_1 = (1, 6), \sigma_3 = (6, 5)]$ is a journey, since $3 \geq 1$. This represents the fact that node 6 can forward the unit it has received from node 1 at time 1 to node 5 at time 3. However, $[\sigma_{13} = (5, 6), \sigma_1 = (1, 6)]$ is not a journey because $1 < 13$. Given a datum unit $k \in \mathcal{D}$, a journey $[(i, u), \ldots, (v, j)]$ between a source node $i \in \mathcal{N} \mid k \in O_i$ and a recipient node $j \in \mathcal{R}$ thus represents a store-carry-forward routing to transmit unit k from i to j. Solving the dissemination problem is equivalent to finding a set of arc-disjoint branchings in this graph (each unit has a corresponding set of branchings rooted on its sources and covering the recipients) [8, 9], *cf.* Figure 1.

2.2. Dominance rules

We will mainly use two dominance rules that are recalled below.

First, since network failures are ignored, it will never be necessary to transmit the same datum unit more than once to the same node during the transfer plan. Hence the following definitions and the ensuing dominance rule.

Definition 2.1. *The transfer occurring at time c with respect to the 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 2.2. *The transfer occurring at time c with respect to transfer plan ϕ is said to be improving if and only if the receiver obtains a new datum unit during σ_c, i.e. $|O^{c-1}_{r_c}| < |O^{c}_{r_c}|$.*

Definition 2.3. *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 2.1. *The set of* minimal *transfer plans is dominant.*

In addition, as we are optimizing the dissemination length, there is no need to delay improving transfers. Therefore we define a strictly-active transfer plan as follows.

Definition 2.4. *A transfer plan ϕ is said to be* strictly-active *if no non-improving transfer could have been improving, i.e. $\forall c \in \{1, 2, \ldots, m\}$,* if $\exists k \in \mathcal{D}, k \in O^{c-1}_{s_c}, k \notin O^{c-1}_{r_c}$, then $|O^{c-1}_{r_c}| < |O^{c}_{r_c}|$.

Proposition 2.2. *The set of* strictly-active *transfer plans is dominant.*

Proposition 2.3. *The set of* minimal *and* strictly-active *transfer plans is also dominant.*

In addition to these dominance rules, in [7] we proposed different *preprocessing algorithms* designed to reduce the search space and to deduce new constraints. These include, for example, algorithms for proving that some contacts are useless in all dominant solutions (the corresponding transfers $\phi(c)$ are therefore set to $\emptyset$). In some cases it is possible to identify datum units that must necessarily be transferred. Unfortunately, using these kinds of algorithms in a *dynamic* context, that is during the execution of the branching algorithm, has been seen to be inefficient within an integer-linear-programming-based procedure. This calls instead for constraint propagation, *i.e.* a powerful *constraint programming* tool that is unsuitable for linear programming.

In the following section, therefore, we propose a new constraint programming model for solving the problem more efficiently. In using constraint propagation (and other mechanisms) we are seeking to perform more effective deductions.

3. From integer linear programming to constraint programming

In this section we first propose a constraint programming model for addressing the data dissemination problem, and then a branching algorithm for obtaining a solution (this procedure will subsequently be refined in Section 4).

3.1. Constraint programming model

To start with, for each node $i \in \mathcal{N}$, we define $\mathcal{T}_i = \{0\} \cup \{c \in \{1, \ldots, m\} \mid r_c = i\}$, the set of time indexes where the state of node i can change (*i.e.* where node i can receive one datum unit). Let $\mathcal{T}_i(t)$, $t \in \{0, 1, \ldots, m\}$ refer to the last contact occurring before time t and where 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:

- $\forall k \in \mathcal{D}$, $\forall c \in \{1, 2, \ldots, m\}$,

$$x_{k,c} = \begin{cases} 1 & \text{if datum unit } k \text{ is transmitted from } s_c \text{ to } r_c \text{ during contact } \sigma_c, \\ 0 & \text{else.} \end{cases}$$

- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, $\forall t \in \mathcal{T}_i$,

$$y_{i,k,t} = \begin{cases} 1 & \text{if node } i \text{ possesses datum unit } k \text{ after contact } \sigma_t, \\ 0 & \text{else.} \end{cases}$$

- $\forall c \in \{1, 2, \ldots, m\}$,

$$a_c = \begin{cases} 1 & \text{if transfer } \phi(c) \text{ is improving,} \\ 0 & \text{else.} \end{cases}$$

- $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{D}$, variable $\lambda_{i,k}$ represents the delivery length which is associated with unit k and node i, *i.e.* the date from which node i possesses datum unit k. Its domain is $\mathcal{T}_i \cup \{\infty\}$ ($\lambda_{i,k} = \infty$ means that node i does not receive unit k during the transfer plan). In practice, we can take $\infty = m + 1$.

- $\forall i \in \mathcal{R}$, variable $\lambda_i = \max_{k \in \mathcal{D}} \{\lambda_{i,k}\}$ represents the delivery length of recipient node i, *i.e.* the date from which node i possesses all the datum units. Its domain is $\mathcal{T}_i$ (i has to be served).

- Variable $\lambda = \max_{i \in \mathcal{R}} \{\lambda_i\}$ represents the dissemination length of the transfer plan. Its domain is therefore $\bigcup_{i \in \mathcal{R}} \mathcal{T}_i \subseteq \{0, 1, \ldots, m\}$.

Remark 3.1. *y-variables are indexed with sets $\mathcal{T}_i$, instead of set $\{0, \ldots, m\}$. However, it is easy to know whether a node $i \in \mathcal{N}$ possesses a datum unit $k \in \mathcal{D}$ at any index $t \in \{0, 1, \ldots, m\}$.*

$$y_{i,k,\mathcal{T}_i(t)} = \begin{cases} 1 & \text{if node } i \text{ possesses datum unit } k \text{ after the } t \text{ first contacts,} \\ 0 & \text{else.} \end{cases}$$

Minimizing the dissemination length gives the following **objective** function:

$$\lambda^* = \min \lambda \tag{3}$$

The **constraints** are written as follows:

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

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \in O_i,\ y_{i,k,0} = 1 \tag{4}$$
$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D} \,|\, k \notin O_i,\ y_{i,k,0} = 0 \tag{5}$$

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

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ x_{k,c} \leq y_{s_c,k,\mathcal{T}_i(c-1)} \tag{6}$$

- Nodes are in possession of a datum unit from the time they receive it:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ y_{r_c,k,\mathcal{T}_i(c-1)} + x_{k,c} = y_{r_c,k,c} \tag{7}$$

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

$$\forall c \in \{1, 2, \ldots, m\},\ \sum_{k \in \mathcal{D}} x_{k,c} = a_c \tag{8}$$

- Variable $\lambda_{i,k}$ is the delivery length corresponding to datum unit k and node i:

$$\forall i \in \mathcal{N},\ \forall k \in \mathcal{D},\ \forall t \in \mathcal{T}_i,\ \begin{cases} y_{i,k,t} = 0 \iff \lambda_{i,k} > t \\ y_{i,k,t} = 1 \iff \lambda_{i,k} \leq t \end{cases} \tag{9}$$

- The transfer plan must be strictly-active:

$$\forall k \in \mathcal{D},\ \forall c \in \{1, 2, \ldots, m\},\ y_{s_c,k,\mathcal{T}_{s_c}(c-1)} > y_{r_c,k,\mathcal{T}_{r_c}(c-1)} \Longrightarrow a_c = 1 \tag{10}$$

Remark 3.2. *Note that constraint* (7) *ensures that node r_c possesses a datum unit k after contact σ_c if it possessed unit k after contact σ_{c-1}, or if it received unit k during contact σ_c. However, the constraint prevents both of these conditions being true at the same time,* and so ensures that the solution is minimal.

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

The model is solved with a branch-and-bound procedure. The efficiency of the algorithm is heavily dependent on the branching variables that are selected and on the quality of the bounds computed at each node of the search tree. For this reason we propose a branching heuristic in the following section. Lower bounds of the problem will be introduced in Section 4.

3.2. 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 significantly reduce the size of enumerations (*cf.* Sections 4.1 and 4.2).

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. In this way it creates one branch for each datum 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, one extra branch is created for the null transfer. In this case, the whole set of $x_{k,c}$ variables ($\forall k \in \mathcal{D}$) is set to 0.

The order in which these branches are visited is heuristic.

1. First of all, we seek to identify the most critical transfers in terms of feasibility. In particular, the fewer the remaining opportunities for node r_c to receive a datum unit $k \in \mathcal{D}$, the more urgent the transfer of datum unit k becomes. Thus, we give priority to branches for which criterion $|\{\, t \in \{c, c+1, \ldots, m\}$ such that $r_t = r_c$ and $x_{k,t}$ is not set to 0 $\}|$ is the lowest.
2. In case of a tie we seek to balance the dissemination of the datum units, and so we first explore the branches with the fewest disseminated units. We prioritize branches whose criterion $|\{\, t \in \{1, \ldots, m\}$ such that $r_t \neq r_c$ and $x_{k,t}$ is not set to 0 $\}|$ is the highest. This approach is based on the heuristic proposed by Belblidia *et al.* in [2].
3. If there is one, the branch associated with the null transfer is 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.

[1] : The behaviour of this algorithm is not described in the present paper. For more details see the online documentation of the IBM-Ilog CP Optimizer.

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.

4. Additional Features

In this section we propose additional features that can easily be integrated into the branching algorithm proposed in Section 3.2. We first propose two lower bounds for the problem, then describe three techniques for breaking the symmetries that are inherent in our approach.

4.1. 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 let $\alpha = u - |\mathcal{O}_i|$ be the number of units that node i has to receive during the transfer plan. Let σ_x 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 transmitted to node i (i.e. such that $x_{k,c}$ 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 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 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	$x_{1,c}$	$x_{2,c}$	$x_{3,c}$	
$\sigma_3 = (s_3, i)$	$\{0, 1\}$	$\{0, 1\}$	$\{0, 1\}$	$\rightarrow \phi(3) \in \{\emptyset, \{1\}, \{2\}, \{3\}\}$
$\sigma_5 = (s_5, i)$	$\{0, 1\}$	$\{0\}$	$\{0\}$	$\rightarrow \phi(5) \in \{\emptyset, \{1\}\}$
$\sigma_8 = (s_8, i)$	$\{0, 1\}$	$\{0\}$	$\{0\}$	$\rightarrow \phi(8) \in \{\emptyset, \{1\}\}$
$\sigma_9 = (s_9, i)$	$\{0\}$	$\{0, 1\}$	$\{0, 1\}$	$\rightarrow \phi(9) \in \{\emptyset, \{2\}, \{3\}\}$

Each contact σ_c, $c \in \{3, 5, 8, 9\}$ is such that there exists a datum unit $k \in \mathcal{D}\backslash\mathcal{O}_i$ for which variable $x_{k,c}$ is not set to 0. 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. 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. Delivery length $\lambda_i(\phi)$ is necessarily greater or equal to 9, because there is no assignment of transfers $\phi(3)$, $\phi(5)$ and $\phi(8)$ enabling node i to receive datum units $\{1, 2, 3\}$, *cf.* Figure 2. From this observation, we propose tightening the lower bound as follows.

Proposition 4.2. *Let $i \in \mathcal{R}$ be a recipient node, and let $\sigma^i \subseteq \sigma$ be the subsequence of contacts built from σ by considering only contacts $\sigma_c = (s_c, i) \in \sigma$ such that there exists a datum unit $k \in \mathcal{D}\backslash\mathcal{O}_i$ for which variable $x_{k,c}$ is not set to 0. Let $\sigma_x \in \sigma^i$ denote the first contact from which –*

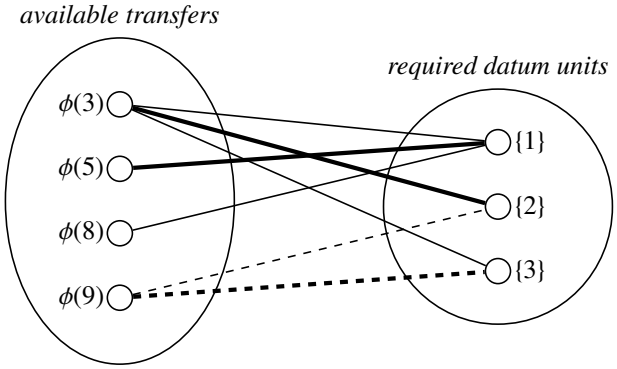

Figure 2: The assignment problem that is solved to compute the lower bound.

by adding the contacts of σ^i one after the other – there exists a valid assignment of the transfers which leads to the delivery of all the datum units to node i. If such an index 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 (see Section 5 for a more comprehensive comparison between these two lower bounds).

Another lower bound can be derived from the proof of polynomiality proposed in [8] for the case where there is only one recipient node, *i.e.* $r = |\mathcal{R}| = 1$. 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.

Solving this problem is based on the observation that where there is only one recipient node, datum units never need to be duplicated, and the trajectory of each datum unit is an elementary journey. Here, the dissemination problem can be reduced to a flow problem where u units of flow are to be routed with the minimum number of contacts through a time-independent version of the evolving graph. The contacts can therefore be added one by one, until a sufficient amount of flow can traverse the graph.

The bound is unfortunately weak in practice. Transforming the problem into a flow problem implies relaxing the identity of the datum units, in the sense that a unit of flow is not associated with a particular datum unit. This prevents us using information collected during the search (*e.g.* the value of the x-variables) to constrain the admissible flows.

4.2. Symmetry-breaking techniques

The overall performance of the search can be improved by breaking the symmetries that are inherent in our approach. The search space contains a large number of equivalent transfer plans that may be ignored by applying dominance rules that we describe below.

Let us first consider the example on the left of Figure 3. At time $t = 3$, datum units 1 and 2 share the same sources (nodes 1, 2, 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 (see the right half of the figure). That is to say, the sub-branchings (in the evolving graph) corresponding to datum units 1 and 2, and with contacts occurring after contact σ_3, can be swapped. The dissemination length is not affected by this operation. As a result, it is sufficient to consider one option ($\phi(4) = \{1\}$ or

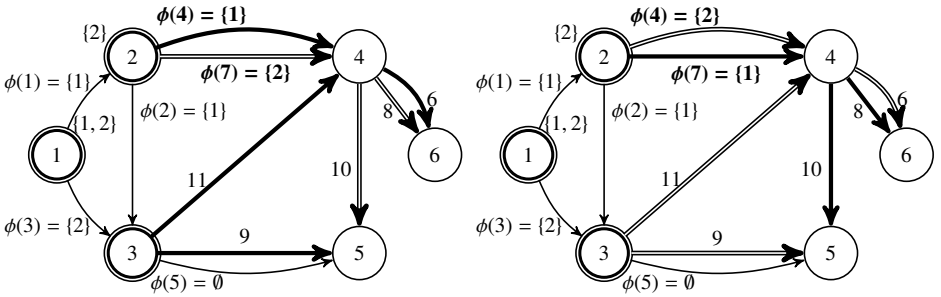

Figure 3: A symmetry appears between units 1 and 2 when we try to fix $\phi(4)$.

$\phi(4) = \{2\}$) 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. This can be formalized as follows.

Definition 4.1. *Let $t \in \{0, 1, \dots, m\}$ denote a time index. A* partial transfer plan ϕ^t *of* length t *is a partial function from $\{1, 2, \dots, m\}$ to $T_\phi = \{\emptyset, \{1\}, \dots, \{u\}\}$, whose domain X is at least $\{1, 2, \dots, t\}$ (i.e. $\{1, \dots, t\} \subseteq X$). Note that domain X can be empty when $t = 0$. The states O_i^x (see equation* (1) *on page 2) are still defined for all the nodes $i \in \mathcal{N}$ and any $x \in \{0, \dots, t\}$. The delivery length $\lambda_i(\phi^t)$ of a node $i \in \mathcal{N}$ becomes $\lambda_i(\phi^t) = \min\{x \in \{0, 1, \dots, t\} \,|\, O_i^x = \mathcal{D}\}$, or $\lambda_i(\phi^t) = \infty$ if such an index does exist. The dissemination length remains $\lambda(\phi^t) = \max_{i \in \mathcal{R}}\{\lambda_i(\phi^t)\}$.*

Definition 4.2. *Let ϕ^t be a partial transfer plan of length $t \in \{0, \dots, m\}$, and let X be its domain. The* extension $\Lambda(\phi^t)$ *of ϕ^t is the set of valid transfer plans ϕ such that $\forall c \in X$, $\phi(c) = \phi^t(c)$.*

Proposition 4.3. *Let $t \in \{0, 1, \dots, m\}$ be a time index, and let 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 share the same source nodes at time t – i.e. $k_1 \in O_i^t(\phi^t)$ if and only if $k_2 \in O_i^t(\phi^t)$ for all the nodes $i \in \mathcal{N}$ – then the subset Φ_1 of transfer plans $\phi_1 \in \Lambda(\phi^t)$ such that $\phi_1(t+1) = \{k_1\}$* dominates *the subset 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 set Φ_1. To this end, we consider the copy ϕ_1 of ϕ_2, where we set $\phi_1(c) = \{k_1\}$ for all $c \in \{t, \dots, m\} \,|\, \phi_2(c) = \{k_2\}$, and $\phi_1(c) = \{k_2\}$ for all $c \in \{t, \dots, m\} \,|\, \phi_2(c) = \{k_1\}$. Transfer plan ϕ_1 therefore belongs to Φ_1 (the roles of datum units k_1 and k_2 have been swapped after contact 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 equivalent for any *other* datum unit $k \in \mathcal{D} \backslash \{k_1, k_2\}$, any node $i \in \mathcal{N}$, and any time index $c \in \{0, 1, \dots, m\}$. 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 $k_1 \in O_i^{c+1}(\phi_2) \Rightarrow k_1 \in O_i^c(\phi_2) \cup \phi_2(c+1)$
   (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. In addition, we can also prove that $k_2 \in O_i^c(\phi_2)$ is equivalent to $k_1 \in O_i^c(\phi_1)$. Thus, we have shown that $O_i^\lambda(\phi_1) = O_i^\lambda(\phi_2)$ at time $\lambda = \lambda(\phi_2)$, *i.e.* $\lambda(\phi_1) = \lambda(\phi_2)$. □

In practice, Proposition 4.3 can be applied at each node of the search tree. Given a sequential branching algorithm such as that described in Section 3.2, 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 $T = \{\, k \in \mathcal{D} \,|\, x_{k,t+1} \text{ is not set to } 0 \,\}$ denotes the set of units that could be transmitted 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 sources. If so, k_2 can be removed from T, that is to say variable $x_{k_2,t+1}$ can be set to 0. 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, because the branch where $x_{k_1,t+1} = 1$ (the set of transfer plans $\Phi_1 \subseteq \Lambda(\phi^t)$ such that $\phi(t+1) = \{k_1\}$) will be explored.

Remark 4.1. *In the trivial cases where only one source node $s \in \mathcal{N}$ has the whole datum at the outset, i.e. where $O_s = \{1, 2, \ldots, u\}$ and $\forall i \in \mathcal{N}\backslash\{s\}$, we have $O_i = \emptyset$. This proposition means we only need to consider one transfer plan out of $u!$ during the search. In these trivial cases, 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 the units to the branchings. Proposition 4.3 ensures that the branchings are assigned to the units according to their indexes (other permutations are ignored).*

Let us now turn to the example shown on the left of Figure 4. Node 3 is the recipient in two contacts σ_1 and σ_2, without having the 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 $\phi(1) = \{1\} \wedge \phi(2) = \{2\}$, or if $\phi(1) = \{2\} \wedge \phi(2) = \{1\}$. Thus, 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 and let k_1, $k_2 \in \mathcal{D}$ be two datum units with $k_1 < k_2$. Let us also assume that 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. $\nexists y \in \{x1, \ldots, x2\} \,|\, s_y = i$. Among the transfer plans such that $k_1 \in O_{s_{x1}}^{x1-1} \cap O_{s_{x2}}^{x2-1} \backslash O_{r_{x2}}^{x2-1}$ and $k_2 \in O_{s_{x1}}^{x1-1} \cap O_{s_{x2}}^{x2-1}$, the set Φ_1 of transfer plans ϕ_1 such that $\phi_1(x1) = \{k_1\}$ and $\phi_1(x2) = \{k_2\}$* dominates *the set Φ_2 of transfer plans ϕ_2 such that $\phi_2(x1) = \{k_2\}$ and $\phi_2(x2) = \{k_1\}$ (the datum unit with the lowest index has to be transferred first).*

Proof. Let $\phi_2 \in \Phi_2$ be a valid 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 can build an equivalent transfer plan $\phi_1 \in \Phi_1$ by copying ϕ_2 and by setting $\phi_1(x1) = \{k_1\}$; $\phi_1(x2) = \{k_2\}$. 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 is applied every time we have to decide the value of a transfer $\phi(x1)$ for which there exists a contact $\sigma_{x2} \in \sigma$ such that $x1 < x2$, $r_{x1} = r_{x2}$, and $\nexists y \in \{x1, \ldots, x2\} \,|\, s_y = r_{x1}$ (in the branching algorithm proposed in section 3.2). Note that the first $x1 - 1$ transfers are then fixed, and the set $T = \{\, k \in \mathcal{D} \,|\, x_{k,x1} \text{ is not set to } 0 \,\}$ of units that can be transmitted during contact σ_{x1} is given. In this way, in every branch $k_2 \in T$ (associated with decision $\phi(x1) = \{k_2\}$), transfer $\phi(x2) = \{k_1\}$ can be blocked if $k_2 \in O_{s_{x2}}^{x2-1}$. 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.

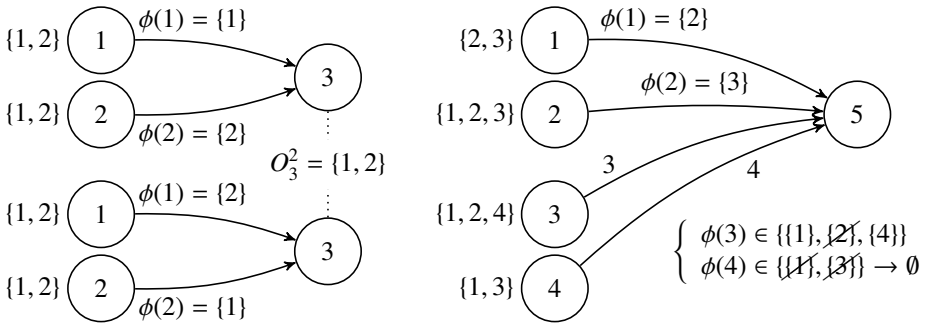

Figure 4: When consecutive contacts occur, datum units can be sent in the order of their indexes.

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

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

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 add the constraints for each possibility. For example, if we look at the instance shown on the right of Figure 4, $x2 = 3$ and $x2 = 4$ both need to be considered when we set $\phi(2) = \{3\}$. Finally, note also that the case where $\phi(3) = \{1\}$ cannot be ignored, because node 3 will not be in possession of datum unit 3 when σ_3 occurs, *i.e.* no swap will be possible between transfers $\phi(2)$ and $\phi(3)$.

Further research could include trying to detect other symmetry patterns. For example, we could attempt to detect that 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 during these four contacts, while ensuring that as far as possible the datum units are transmitted in the order of their indexes. In this way, more transfers could be arbitrarily set, and even fewer transfer plans would need to be considered during the search.

The final technique for breaking symmetries that we propose in this paper involves registering the state of the nodes visited in the search tree. This enables branches to be pruned by detecting dominance relationships among the transfer plans *under construction*. Different sub-sequences of transfers can give rise to a similar dissemination – *i.e.* there may exist two partial transfer plans ϕ_1^t and ϕ_2^t of size $t \in \{1, \ldots, m\}$ 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 partial transfer plans of size $t \in \{1, 2, \ldots, \min\{m, \lambda(\phi_1^t)\}\}$. If $O_i^t(\phi_2^t) \subseteq O_i^t(\phi_1^t)$ 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 $\Lambda(\phi_2^t)$. Let $\phi_1 \in \Lambda(\phi_1^t)$ be the transfer plan built with the first t transfers of ϕ_1^t and completed 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 $\forall c \in \{t + 1, \ldots, m\}$, $\phi_1(c) = \phi_2(c)$. We know that $O_i^t(\phi_2) \subseteq O_i^t(\phi_1^t) = O_i^t(\phi_1)$ for all the nodes $i \in \mathcal{N}$. Let us then assume 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 $c + 1$. If $i \neq r_{c+1}$, then $O_i^{c+1}(\phi_2) = O_i^c(\phi_2) \subseteq O_i^c(\phi_1) = O_i^{c+1}(\phi_1)$. If $i = r_{c+1}$, $O_i^{c+1}(\phi_2) = O_i^c(\phi_2) \cup \phi_2(c + 1) \subseteq O_i^c(\phi_1) \cup \phi_1(c + 1) = O_i^{c+1}(\phi_1)$. Consequently, we

have proved by recurrence that $O_i^c(\phi_2) \subseteq O_i^c(\phi_1)$ holds for any index $c \in \{t, \ldots, m\}$. 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 $\mathrm{S}_t^{curr} = \{O_i^t \mid i \in \{1, 2, \ldots, q\}\}$ 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 a *visited* partial transfer plan of size $t \in \{1, 2, \ldots, m\}$. From Proposition 4.5 we have 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 $\forall i \in \mathcal{N}$, $O_i^t(\mathrm{S}_t^{curr}) \subseteq O_i^t(\mathrm{S}_t^{dom})$.

This kind of symmetry-breaking technique falls within the well-known *nogood recording* framework that was first introduced by Verfaillie and Schiex [10]. The major drawback of such a method is the huge number of global states which can be generated and thus recorded. An easy way of managing this is to implement each list L_t as a truncated heap 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 4.2. *If the global states are represented with bit vectors, testing whether two given 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 thousands of global states per list* L_t*,* $t \in \{1, 2, \ldots, m\}$ *and thereby, to improve the performances of the solver significantly, cf. Section 5 for more details.*

In this section, 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, while the third is more *reactive* insofar as it aims to detect and prune symmetric branches after they have been generated.

In the following section, we assess the constraint-programming-based approach proposed in Sections 3 and 4, *i.e.* the model, the branching algorithm, the lower bounds, and the three symmetry-breaking techniques.

5. Computational Results

In this section computational results are reported and discussed. First we describe the benchmark used to perform these tests. The best results achieved in [7] for these instances are recalled, and these serve as a reference for subsequent comparisons. The model described in Section 3 is assessed in Section 5.2. The features proposed in Section 4 are discussed in Section 5.3.

5.1. About the benchmark

Below we consider 184 *instances*, organized into eight *classes*. The classes are listed in terms of the number *nbinst* of instances they include, the number u of datum units and the number q of nodes characterizing those instances, *cf.* Table 1. The average number of receivers $\overline{rec} = |\mathcal{R}|$, the average number of sources $\overline{src} = |\{i \in \mathcal{N} \mid O_i \neq \emptyset\}|$ and the average number of contacts $\overline{m}$ in the instances are also reported. Moreover, the eight *classes* are *grouped* by difficulty. The first group (3u10n and 4u20n) comprises the easiest instances. The second and third group comprise respectively the instances with many systems but few units, and the instances with few systems but many units.

	name	*nbinst*	u	q	$\overline{rec}$	$\overline{src}$	$\overline{m}$
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

Table 1: Some aggregates describing the benchmark.

Like in [7], the computations were all performed on a server equipped with 16×6 cores (running at 2.67Ghz) and 1To RAM. Algorithms were all implemented with `C++`. Constraint-programming models were solved with *CP Optimizer*, a commercial solver developed by IBM-ILOG. Note that the multithreading features proposed by the library were deactivated, because the use of concurrent optimizers led to unstable results (*e.g.* different executions of the same computations generally led to very different results in terms of cpu time), which prevented reliable comparisons between our methods. Each instance was therefore solved using a *deterministic* sequential algorithm with a one-hour time limit.

Our tables of results include the same metrics in each case, namely:

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

   It follows that $100\% - solved - feas$ (-%) indicates the ratio of instances for which the solver neither found a feasible solution nor showed the instance to be infeasible.
3. cpu (-s) indicates the average time taken by the given solver to solve the problem, over all instances and thus all time limits. cpu therefore tends to 3600s if *solved* tends to 0.0%.

As a reference, we first report (in Table 2) the best results we previously achieved in [7] using integer linear programming.

Remark 5.1. *In the following paragraphs different sets of parameters are compared and the best strategies selected. In our opinion, strategies should be selected for the three groups rather than for the eight classes. In this way, the benchmark cannot be learned by heart. All the instances are available on [11].*

5.2. About constraint programming

Let us first discuss the computational results reported in Table 3. These results were obtained using three different algorithms. **min** consists in solving the constraint programming model defined by constraints (4) through (9) (the solutions are only required to be minimal). **min+st/act** follows the same approach, but by also considering constraint (10) (the solutions therefore need to be minimal and strictly-active). **prep+min+st/act** consists in calling the preprocessing procedure defined in [7], followed by **min+st/act**.

	name	*solved*	*feas*	*cpu*
1	**3u10s**	100	-	0.47
	4u20s	100	-	2.0
2	**4u50s**	100	-	4.1
	4u100s	100	-	20.1
	5u50s	100	-	19.4
3	**10u10s**	93.8	6.3	464
	50u10s	68.8	6.3	1691
	100u10s	66.7	0.00	3305

Table 2: The best results achieved with integer linear programming in paper [7].

		min			**min+st/act**			**prep+min+st/act**		
	name	*solved*	*feas*	*cpu*	*solved*	*feas*	*cpu*	*solved*	*feas*	*cpu*
1	**3u10s**	100	-	3.1	100	-	4.1	100	-	0.84
	4u20s	100	-	12.3	100	-	10.2	100	-	5.5
2	**4u50s**	100	-	339	100	-	60.2	100	-	18.5
	4u100s	80.0	20.0	818	100	-	196	95.0	5.0	333
	5u50s	69.6	30.4	1316	91.3	8.7	555	100	-	173
3	**10u10s**	18.8	50.0	3027	18.8	50.0	2964	37.5	37.5	2269
	50u10s	0.00	31.3	3602	0.00	0.00	3601	0.00	0.00	3601
	100u10s	0.00	33.3	3604	0.00	0.00	3603	0.00	0.00	3603

Table 3: The computational results achieved with CP Optimizer and different models.

The computational results reported in Table 3 might appear disappointing at first glance, especially compared to the results achieved using linear programming that are reported in Table 2. Nevertheless, it should be remarked that there is a significant gap between **min** and **min+st/act**. This shows that we were successful in implementing the dynamic deduction procedures that were discussed in [7], *i.e.* we were able to integrate fully into the solver the concepts of minimal and strictly-active transfer plans. This improvement was made possible by the introduction of a-variables, and through non-linear constraints (10).

There is also a clear gap between algorithms **min+st/act** and **prep+min+st/act**, showing that the preprocessing procedures we proposed in [7] outperform the generic consistency algorithms implemented in CP Optimizer.

5.3. Additional features

To get the most out of constraint programming, a custom branching algorithm often needs to be implemented. The strategy proposed in Section 3.2 is quite easy to implement with a constraint programming tool. Transfers need to be set and the nodes visited according to 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 consequently that they should be given priority during the branching. Our computational results are reported in Table 4, columns **none**. The proportion of instances

solved is significantly better, but is still not comparable to results obtained using integer linear programming.

To obtain competitive results, we must consider the *ad hoc* tools we described in Section 4, namely the weak (**wlb**) and the strong (**slb**) lower bounds, the two proactive symmetry-breaking techniques (**sym**), and the nogood-recording (**ngr**) procedure, *cf.* Tables 4 through 6.

wlb / slb :. The lower bounds have positive effects on the performance of our method. They enable the solver to prune some branches earlier. The weak lower bound provides better results for the first two groups, *i.e.* on the smallest instances, but not for larger instances. Conversely, the strong lower bound provides better results for the third group (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 // tight bounds, and on the other light computations // weak bounds.

ngr / sym :. The symmetry-breaking techniques enable the search space to be significantly reduced at a low computational cost. The results show **sym** and **ngr** operate well across all classes.

In summary, algorithm **sym+ngr+wlb** should be applied on the first two groups, while algorithm **sym+ngr+slb** is more suitable for the third group, *cf.* Table 7.

6. Conclusions and Perspectives

In this paper we address the problem of transferring data in a deterministic delay-tolerant network (system of systems). We propose an algorithm based on constraint programming for solving the so-called *dissemination problem*. This supplements our previous algorithm [7], with a better heuristic branching algorithm and some extensions (*e.g.* some lower bounds and symmetry breaking techniques).

The table below shows the best computational results achieved so far, for each class and across the two papers (we refer the reader to [7] for more details about column *preprocessing*).

	algorithm		***ad hoc* procedures**		**results**		
name	*solver*	*model*	*preprocessing*	*features*	*solved*	*feas*	*cpu*
3u10s	CPLEX*	min+st/act	minimal	-	100	-	0.26
4u20s	CPO	min+st/act	light	sym+wlb	100	-	1.3
4u50s	CPO	min+st/act	light	sym+wlb	100	-	2.6
4u100s	CPLEX*	min	light	-	100	-	20.1
5u50s	CPO	min+st/act	light	sym+wlb	100	-	12.6
10u10s	CPO	min+st/act	aggressive	sym+ngr+slb	100	-	36.1
50u10s	CPO	min+st/act	normal	sym+ngr+slb	87.5	12.5	540
100u10s	CPO	min+st/act	normal	sym+wlb	100	-	129

* : an Integer Linear Programming solver developped by IBM-ILOG ; results from [7]

In the future 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 contacts are successful.

		none		sym		sym+ngr		sym+ngr+wlb	
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*
1	**3u10s**	100	0.46	100	0.47	100	0.46	100	0.39
	4u20s	100	1.8	100	1.7	100	1.7	100	1.3
2	**4u50s**	100	6.3	100	3.2	100	3.2	100	2.7
	4u100s	100	198	100	160	100	58.5	100	88.5
	5u50s	100	51.2	100	33.7	100	39.4	100	20.7
3	**10u10s**	43.8	2236	50.0	1807	81.3	812	87.5	462
	50u10s	43.8	2106	43.8	2105	43.8	2102	75.0	1029
	100u10s	83.3	1184	83.3	1184	83.3	1189	100	702

Table 4: Computational results obtained by different algorithms with CP Optimizer – *part*-1.

		sym+ngr+slb		sym+wlb		sym+slb		ngr	
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*
1	**3u10s**	100	0.59	100	0.38	100	0.59	100	0.46
	4u20s	100	1.8	100	1.3	100	1.8	100	1.7
2	**4u50s**	100	3.6	100	2.6	100	3.7	100	5.2
	4u100s	100	216	100	181	100	174	100	107
	5u50s	100	73.1	100	12.6	100	66.0	100	54.2
3	**10u10s**	100	36.1	68.8	1135	93.8	379	43.8	2143
	50u10s	87.5	640	75.0	1032	75.0	1061	43.8	2107
	100u10s	100	1220	100	731	100	1291	83.3	1183

Table 5: Computational results obtained by different algorithms with CP Optimizer – *part*-2.

		ngr+wlb		ngr+slb		wlb		slb	
	name	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*	*solved*	*cpu*
1	**3u10s**	100	0.38	100	0.59	100	0.38	100	0.60
	4u20s	100	1.3	100	2.0	100	1.5	100	2.2
2	**4u50s**	100	4.3	100	7.7	100	5.6	100	11.9
	4u100s	100	84.6	100	279	100	154	100	395
	5u50s	100	46.7	100	241	100	29.9	95.7	245
3	**10u10s**	50.0	1807	68.8	1425	43.8	2091	62.5	1725
	50u10s	62.5	1689	62.5	1497	62.5	1662	62.5	1495
	100u10s	83.3	1187	83.3	1431	83.3	1187	83.3	1473

Table 6: Computational results obtained by different algorithms with CP Optimizer – *part*-3.

	name	*algorithm*	*solved*	$feas$	*cpu*
1	**3u10s**	sym+ngr+wlb	100	-	0.39
	4u20s	sym+ngr+wlb	100	-	1.3
2	**4u50s**	sym+ngr+wlb	100	-	2.7
	4u100s	sym+ngr+wlb	100	-	88.5
	5u50s	sym+ngr+wlb	100	-	20.7
3	**10u10s**	sym+ngr+slb	100	-	36.1
	50u10s	sym+ngr+slb	87.5	12.0	640
	100u10s	sym+ngr+slb	100	-	1220

Table 7: The algorithms that obtained the best results with CP Optimizer in every group.

Acknowledgement

These works are financed by the *Conseil Régional de Picardie* and carried out in the framework of *Labex MS2T*, funded by the French government through the *Investments for the Future* program, and managed by the *National Agency for Research* (ANR-11-IDEX-0004-02).

References

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

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

[3] 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 '03, ACM Press, New York, USA, 2003, p. 27.

[4] A. Pentland, R. Fletcher, A. Hasson, DakNet: Rethinking Connectivity in Developing Nations, Computer 37 (1) (2004) 78–83.

[5] S. Merugu, M. Ammar, E. Zegura, Routing in Space and Time in Networks with Predictable Mobility, Tech. rep., Georgia Institute of Technology (2004).

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

[7] R. Bocquillon, A. Jouglet, Modeling elements and solving techniques for the data dissemination problem, preprint extended version available at `https://hal.archives-ouvertes.fr/hal-01178611`, submitted.

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

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

[10] G. Verfaillie, T. Schiex, Solution reuse in dynamic constraint satisfaction problems, in: Proceedings of the 12th National Conference on Artificial Intelligence, 1994, pp. 307–312.

[11] Instances – `https://www.hds.utc.fr/~rbocquil/dokuwiki/_media/dp_instances.zip`