A GLOBAL CONSTRAINT MODEL FOR INTEGRATED ROUTEING AND SCHEDULING ON A TRANSMISSION NETWORK ♦

Lluís Ros*, Tom Creemers*, Evgueni Tourouta† and Jordi Riera*

* Institut de Robòtica i Informàtica Industrial. Llorens i Artigas, 4-6, 2^a planta. 08028 Barcelona, Spain. e-mails: *llros@iri.upc.es, creemers@iri.upc.es, riera@iri.upc.es* .

† Faculty of Automatic Telecommunication, Informatics and Computer Science.
Moscow Technical University of Communication and Informatics.
Aviamotornaya St., 8a. 111024 Moscow, Russia. e-mail: *turuta_en@mail.ru* .

ABSTRACT

This paper presents a novel approach to planning the sending of messages along a meshed transmission network with limited bandwidth paths. The approach allows determining the topological routeing of messages and a feasible time schedule satisfying three basic constraints: maximum bandwidth capacity per path, integrity of packages and maximum receiving time for each message. Constraint Logic Programming (CLP) is used to solve the combined problem of routeing and scheduling in an integrated manner. To this end, a finite-domain model with global constraints has been derived for a generic transmission network. The model implementation, in the CLP language CHIP, is explained in detail. An implementation has been run on extensive test cases, showing the efficiency of this approach.

Keywords: Constraint Logic Programming, global constraints, transmission networks, bandwidth packing, path assignment, routeing, scheduling, integrated model.

1 INTRODUCTION

Given a generic transmission network, the number of messages that can be simultaneously transmitted along any one of its lines is constrained by the line's capacity or *bandwidth*. We will assume that, when a message is transmitted from one node to another, the sent information occupies a fixed portion of the bandwidth of all the lines along its route. Hence, if on a given line the whole of the available bandwidth is being used - we say the line is saturated - it will be impossible to transmit any additional message through it. This aspect of the problem statement is referred to as *bandwidth packing*.

In this context, one of the posed problems consists of determining an optimal *routeing* for a set of messages between pairs of nodes of the network [8]. More precisely, given a set of requests for messages where each one is to be sent from an *origin* to a *destination* node, one wishes to determine a corresponding set of feasible routes along which messages can be simultaneously transmitted, while guaranteeing that no bandwidth constraint is violated.

The problem gets more interesting if one cares taking into account the temporal dimension. In the latter case, the transmission of each message must be additionally scheduled in time, and each message lasts for a certain duration and should reach its destination node before a given *due date*. Hence, we do not only want to know the topological routeing of messages, but also obtain a feasible *time schedule*. In this work, we tackle this combined problem of scheduling and routeing.

In [8] routeing and scheduling are solved separately; the generation of feasible routes is done by the Constraint Logic Programming system ECL^iPS^e [6], while several schedulers based on random search, simulated annealing, classifier systems or hill climbing, are compared. In [9], Guided Local Search [10] was applied to this problem, but the inclusion of the temporal dimension was abandoned and the problem was viewed as a pure path assignment problem [2].

Our approach is different in that the entire combined routeing and scheduling problem is represented as one single constraint model and solved

♦ This work has been partially funded by the spanish CICYT project TAP-99-1086-C03-01.

together efficiently by constraint propagation, always ensuring global optimality. Routes are constructed simultaneously with the construction of the time schedule. Hence, the notion of path "assignment" is not strictly applicable any more. The approach is based on the use of *global constraints* [3] in the Constraint Logic Programming system CHIP++ v5 [4,5].

The paper is organised as follows. Section 2 starts with necessary notation and definitions, including the notions of time schedule and message routeing. The problem statement is given in Section 3, by listing the constraints that define a feasible schedule and message routeing. This constraint-satisfaction problem will be tackled via the Constraint Logic Programming paradigm, for which Section 4 provides an introduction. Using this paradigm, a high-level constraint model solving the problem is then given in Section 5. How the model can be implemented with the global constraint predicates of the CHIP system is then analysed in Sections 6 and 7. Finally, Section 8 concludes and points out possible extensions.

2 DEFINITIONS

We model a transmission network X as a pair $X = (G, bw)$, where:

- $G = (V, A)$ is a directed graph with a set V of n nodes, $V = \{v_1, v_2, \ldots, v_n\}$, and a set A of m arcs, $A = \{a_1, a_2, \ldots, a_m\} \subset V \times V$. All arcs are oriented. Each arc models a line of the network, while each node models the linking device between three or more of these lines.
- bw is a function that assigns a non-negative integer capacity $bw(a)$ to each arc a, which models its bandwidth.

A *path* of X (or G) is an alternating sequence of nodes and arcs, $(v_0, a_1, v_1, a_2, \ldots, v_{l-1}, a_l, v_l)$, such that, $a_i = (v_{i-1}, v_i)$, $i = 1, \ldots, l$, and none of the nodes $v_1, \ldots, v_{l-1}$ appears twice. An $o-d$ *path* is a path with $v_0 = o$ and $v_l = d$. A *cycle* of X (or G) is a path with $v_0 = v_l$.

On the network X we can consider a set M of *messages* to be transmitted. A message $m_i \in M$ is represented as a tuple $m_i = (o_i, d_i, b_i, t_i, l_i, f_i, P_i)$, where:

- $o_i \in V$ and $d_i \in V$ are, respectively, the *origin* and *destination* nodes between which the communication of message m_i will take place.
- $b_i \in Z^+$ is the required *bandwidth* for m_i.
- $t_i \in T \cup \{0\}$ is the time instant when m_i should begin to be emitted.
- $l_i \in T$ is the duration of the message, that is, the total number of time intervals that the i-th message is occupying the bandwidth b_i, on all lines which it flows along. The message begins at time t_i and ends at time $t_i + l_i$.
- $f_i \in T$ is the due date of message m_i.
- P_i is the *path* of X along which the message will flow to go from node o_i to node d_i.

We assume that all message transmissions take place within a time horizon discretised in t_f time intervals, delimited by the time instants $t = 0, 1, 2, \ldots, t_f$. We index these intervals so that the i-th interval is the one ranging from $t = i-1$ to $t = i$, and $T = \{1, 2, \ldots, t_f\}$ will denote the set of these indices.

A *time schedule* is an assignment of values to the variables t_i within the set $T \cup \{0\}$. A *message routeing* is an assignment of a path P_i to each message.

3 PROBLEM STATEMENT

Our goal is to find both a time schedule and a message routeing that satisfy the following constraints:

- *Saturation* (or *bandwidth packing*). No arc can transport more messages than the maximum allowed by its bandwidth capacity. Thus, if O_j^t denotes the set of messages flowing along arc a_j in the time interval t, then, $\forall t \in T, \forall a_j \in A$ it must hold:

$$\sum_{m_i \in O_j^t} b_i \le bw(a_j). \qquad (1)$$

- *Continuity* and *unbranching*. Every message m_i arrives at a node along only one of the arcs incident to it, and leaves it through only one of the arcs emanating from it. Thus, the information contained in a message cannot be partitioned into smaller packages to send them along different paths.
- *Due date*. Message m_i must arrive at node d_i before time instant f_i.

We could optionally add other constraints to the problem, as for example impose that specific messages never flow along intersecting paths, or assign priorities to the messages in order to obtain a schedule where the important messages are scheduled earlier than lower-priority ones. Although the constraint-solving paradigm we use is general enough to deal with these constraints, we do not consider these possibilities in this work.

4 Constraint Logic Programming

The idea behind *Constraint Programming* is the declarative expression of problems as constraints and the use of constraint solvers specialised in restricted domains (like real numbers, or finite sets) to find their solutions. A problem is represented by an underlying network of variables and constraints. As soon as information becomes available anywhere in the network, constraint solvers are woken up to propagate this information through the whole network, prunning the variable domains as much as possible by eliminating unfeasible values. This in turn may wake up other constraint-solvers and the process iterates until no further prunning is achieved. As it would be out of the scope of this paper to give a detailed discussion of Constraint Programming methods, we refer the interested reader to surveys such as [11] and [7] for an introduction.

Consistency techniques form an important subclass of constraint handling methods, and particularly popular here are the so-called *finite-domain* constraint models, where decision variables range over finite sets of possible values. Such variables are called *finite-domain* variables. Techniques like arc- and path-consistency are then used to remove inconsistent values from the domains until feasible solutions are found. Consistency techniques have been succesfully integrated within Logic Programming, giving rise to the so-called *Constraint Logic Programming* paradigm [5], or CLP, for short.

Most finite-domain constraints are based on syntactic, domain-independent propagation methods. However, in [1] and [3] a new type of finite-domain constraints based on semantic methods was first introduced in the CLP language CHIP. These constraints use domain-specific knowledge to derive better propagation properties. Constraints of this type are called *global constraints* and combine the following important properties:

- They model a complex condition on larger sets of variables.
- The constraint reasoning often detects inconsistency rapidly and prunes the search space significantly.
- They can be applied to large problem instances and their semantics fits in multiple contexts.

In this paper we develop such a finite-domain model with global constraints for the combined scheduling/routeing problem presented above. Our implementation is entirely based on CHIP++ v5, mainly making use of the global constraints `cycle/10` and `diffn/1` of this language.

5 Constraint model

Solving a problem in the CLP paradigm boils down to defining a proper constraint model, which involves defining a set of proper finite-domain variables modeling the decisions, and proper predicates modeling the constraints. The constraint model for the problem in Section 3 is next developed.

We represent each path P_i in the message routeing with n finite-domain variables $S_{i,1}$, $S_{i,2}$, ..., $S_{i,n}$, hereafter referred to as *topological variables*, defined as follows:

- We let $S_{i,j} = k$, $1 \le k \le n$, to mean that node v_k has been chosen as the successor of v_j in the path P_i.
- If $S_{i,j} > n$, then node v_j does not belong to the path P_i.

For the message scheduling, we represent the initial time instant t_i of message m_i with an additional finite-domain variable T_i. Its initial domain is set to $\{0, 1, \ldots, f_i - l_i\}$, which implicitly ensures satisfying the due date constraint for this message. These variables are hereafter referred to as *temporal variables*.

The following predicates will allow an easy top-level description of the constraint model:

- `path(o,d,Ss)`

  `Ss=[`$S_1, S_2, \ldots, S_n$`]` is a list with n domain variables. o and d are nodes. The list `Ss` stores a path between o and d. We say that `Ss` is a *path-list*. We use the same convention used for the topological variables: if $S_i = k$, $1 \le k \le n$, then node v_k is the successor of v_i in the *o-d* path, otherwise, when $S_i > n$ this will mean that v_i does not belong to this path.

  The `path/3` predicate[1] is true if the path-list `Ss` defines an *o-d* path of G. Its implementation is given in Section 6.

1. The notation `predicate/n` is common in Logic Programming. The number n indicates the amount of arguments of the predicate.

- `non_saturation(Ts,Ls,Cs)`

  `Ts=[`$T_1, T_2, \ldots, T_{|M|}$`]` is a list of $|M|$ finite-domain temporal variables. `Ls=[`$l_1, \ldots, l_{|M|}$`]` is a list of integers storing the durations of the messages. `Cs=` `[`$C_1, \ldots, C_n$`]` is a list of path-lists. Each C_i is a path-list of the form $C_i = [S_{i,1}, \ldots, S_{i,n}]$ and defines the path along which the message m_i will flow. The `non_saturation/3` predicate is true if during the time intervals between time instant T_i and time instant $T_i + l_i$, $i = 1, \ldots, |M|$, there is no line saturation at any of the lines in the paths contained in `Cs`. This constraint corresponds to Equation (1) and its implementation is given in Section 7.

With these definitions, and using PROLOG-like syntax, the logical program modeling our problem can be written as follows:

```
routeing_scheduling(S_{1,1}, ..., S_{|M|,n},
                    T_1, T_2, ..., T_|M| ) :-

    /* Statement of constraints */
    path(o_1, d_1 , [ S_{1,1}, ..., S_{1,n} ]),
    ...
    path(o_|M|, d_|M| , [ S_{|M|,1}, ..., S_{|M|,n} ]),
    non_saturation([ T_1, T_2, ..., T_|M| ],
                   [ l_1, ..., l_|M| ],
                   [ S_{1,1}, ..., S_{|M|,n} ]),

    /* consistent labeling */
    labeling([ S_{1,1}, ..., S_{|M|,n} ]).
```

The declarative semantics of this program is as follows. The variables $S_{1,1}, \ldots, S_{|M|,n}$ and $T, \ldots, T_{|M|}$ define a feasible sequence of message transmissions and a feasible routeing of them if and only if all the following conditions hold:

- $S_{i,1}, \ldots, S_{i,n}$ define an o_i-d_i path, for $i = 1, \ldots, |M|$.
- $T_1, T_2, \ldots, T_{|M|}$ define a sequence of sendings such that no saturation is produced at any of the lines traversed by the messages.

6 IMPLEMENTING `path/3`

The predicate `cycle/10` has been added to the CHIP language so as to ease the modeling of several problems of assignment, routeing and graph partitioning and will be here used to implement the `path/3` predicate.

6.1 The Constraint Predicate `cycle/10`

Basically, given a graph $G = (V, A)$ and a natural number $N > 0$, `cycle/10` expresses a partitioning of G into exactly N cycles, $C_1, C_2, \ldots, C_N$, of G, in such a way that every node in V appears in exactly one of these cycles. `cycle/10` has the following structure:

```
cycle(N, Ss, Ws, Min, Max, Ds,
      Ls, Ms, Origin, Eq).
```

By using these 10 arguments, one can establish several constraints on the N cycles. For the purpose of our problem we will not use the `Ms` and `Origin` arguments, which will be set to the atom `unused`. The reader may find an exhaustive description of all the arguments in the CHIP reference information [4]. The remaining arguments are as follows:

- `N`: is a natural number different from zero. It is the number of cycles into which G is partitioned.
- `Ss`: is a non-empty list $[S_1, \ldots, S_n]$ of n finite-domain variables or integers, where n is the number of nodes in G. To use `cycle/10`, one has to provide in `Ss` the structure of G, by initializing the domain of each variable S_i to contain the identifiers of the adjacent nodes from v_i.
- `Ws`: is a non-empty list $[W_1, \ldots, W_n]$ of n integers or domain variables, representing the 'weights' of the nodes. To simplify, we will take all weights equal to 1. Hence `Ws` is a list of ones, and the notion of 'weight' corresponds to the 'length' of a cycle.
- `Min` and `Max`: are two integers indicating respectively the minimum and maximum weight of the nodes accumulated in a cycle. With the node weight equal to one we can set `Min` equal to `1` and `Max` equal to n.
- `Ds`: is a list $[D_1, \ldots, D_p]$ of p integers, indicating the nodes which cannot belong to a same cycle. It expresses the partitioning into different cycles, by identifying a representative node for each one.
- `Ls`: is a list $[L_1, \ldots, L_p]$ of p domain variables or integers, expressing a weight (a length) for each one of the cycles identified in `Ds`.
- `Eq`: is a list of c elements, $[E_1, \ldots, E_c]$. Each E_i is a list of integers and contains two or more identifiers of nodes of G. No identifier can appear in more than one list E_i. All nodes whose

identifier appears in the same list are said to belong to the same equivalence class and belong to the same cycle.

The declarative semantics of `cycle/10` is as follows. Given a graph G defined by the initial values of the domains of the variables in the list `Ss`, and given a natural number `N`, `cycle/10` is true when it is possible to find exactly `N` cycles of G, such that each node of G appears in exactly one cycle and, moreover, the nodes included in different equivalence classes appear also in different cycles. Moreover, each S_i in `Ss` will store the number of the successor node of v_i in the cycle traversing v_i.

We describe the constraints imposed on $S_1,...,S_n$ by the `cycle/10` predicate.

- $S_1,...,S_n$ must be initialised to take values in the set of possible node identifiers of nodes of G.
- No two nodes can have the same successor: $S_i \neq S_j$, $\forall i, j \in \{1, ..., n\}, i \neq j$.
- $\forall i \in \{1, ..., n\}$, let C_i be the set recursively defined as follows:

  $$v_i \in C_i,$$

  $$\text{if } v_j \in C_i \text{, then } S_j \in C_i$$

  Thus, C_i contains the identifiers of the nodes of the cycle traversing v_i. Each node must appear in exactly one cycle:

  $$V = \bigcup_{i=1}^{n} C_i$$

  $$C_i \cap C_j = \varnothing,$$

  $$\forall i, j \in \{1, ..., n\}, v_i \notin C_j, \ v_j \notin C_i$$
- Let e_{i1}, ..., e_{ik} be the integers occurring in C_i, then the accumulated weight of a cycle must satisfy $\text{Min} \le W_{e_{i1}} + \ldots + W_{e_{ik}} \le \text{Max}$.
- There must be exactly N different cycles $|\{C_1, C_2, ..., C_n\}| = N$.
- The nodes in `Ds` cannot belong to the same cycle. That is, $\forall k \in \{D_1, ..., D_p\}$, $\forall l \neq k \in \{D_1, ..., D_p\}$, $C_k \cap C_l = \varnothing$.
- Let $d = D_i$ and e_{d1}, ..., e_{dk} the integers occurring in C_d, then the accumulated weight (length) of its cycle must satisfy $L_i \le W_{e_{d1}} + \ldots + W_{e_{dk}}$.
- Finally, nodes belonging to different equivalence classes must belong to different cycles as well. That is, $\forall v_i \in E_l, v_j \in E_k$, $i \neq j$, it must be: $C_i = C_j$ if $l=k$, and $C_i \neq C_j$ if $l \neq k$.

6.2 `cycle/10` Constraints for the Routeing subproblem

Now consider our transmission network X and its associated graph $G = (V, A)$. We want to take advantage of the constraints imposed by `cycle/10` in order to implement the `path/3` predicate and implicitly define all the possible paths linking each pair of nodes o_i and d_i. Our use of `cycle/10` to implement `path/3` is motivated by the fact that it will allow us to combine the constraint in an elegant way with another global constraint and in this way achieve a true integration of the routeing, bandwidth packing and scheduling problems, as shown below. However, the `cycle/10` cannot be directly used due to the following limitations:

(1) It does not generate paths, but cycles.

(2) It requires the graph to be partitionable in exactly N cycles.

We can overcome these limitations by adding extra nodes and arcs to the graph $G = (V, A)$. For each message m_i we will construct a new graph G_i^a, in such a way that it is possible to use `cycle/10` to implicitly define all paths between any couple of nodes in G. G_i^a is called the *augmented graph*. Let $G_i^a = (V_i^a, A_i^a)$. The set of nodes is defined as $V_i^a = V + W + \{f_i\}$, where $V = \{v_1, v_2, ..., v_n\}$ are the nodes of the original graph G, $W = \{w_1, w_2, ..., w_n\}$ is a set of *auxiliary nodes*, and f_i is an additional node called *phantom*. We define the set of arcs A_i^a as

$$A_i^a = A + A' + A'' + \{(d_i, f_i), (f_i, o_i)\}$$

where:

$$A' = \{(v_1, w_1), (v_2, w_2), ..., (v_n, w_n)\}\text{, and}$$

$$A'' = \{(w_1, w_2), ..., (w_{n-1}, w_n), (w_n, w_1)\}$$
$$\cup \{(w_1, v_2), ..., (w_{n-1}, v_n), (w_n, v_1)\}.$$

The definition of G_i^a is not as artificial as it may seem. Its schematic view of is depicted in Figure 1. In order that all o_i - d_i paths are present in at least one cycle of G_i^a, we introduce a phantom node f_i and two arcs (d_i, f_i), (f_i, o_i). We hence solve the first limitation above. In order to partition G using a fixed number of cycles (limitation (2) above), we introduce the n auxiliary nodes in W, and the arcs in A' and A''. This augmented graph will be partitionable into exactly two cycles: the cycle C following a path from o_i to d_i and returning back to o_i via the single node f_i. All the remaining non-visited

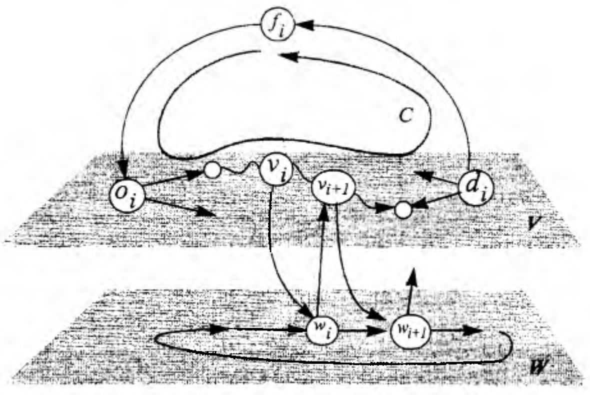

Figure 1. The structure of G_i^a.

nodes can be visited in one single second cycle. This can be seen easily: if all the nodes of the original graph belong to C then all the nodes of W can be collected trivially in their numbering order (and back from w_n to w_1). If a node $v_j \in V$ was not collected in C, then it suffices to substitute the arc (w_{j-1}, w_j) in the second cycle by the deviation formed by the two arcs (w_{j-1}, v_j) and (v_j, w_j).

For each message m_i, we must avoid the presence of auxiliary nodes in the cycle C traversing o_i and d_i. For this purpose we define two equivalence classes: $E_{i1} = \{v | v \in W\}$ and $E_{i2} = \{f_i\}$.

Now, the definition of the extended graph associated with each message, allows an easy implementation of the `path/3` predicate:

```
path(oi, di, Gai):-
    extended_graph(Gai, oi, di),
    cycle(2, Gai, [1, ..., 1],
          1, fi, Ds, Ls,
           unused, unused, [[fi],W]).
```

where the predicate `extended_graph/3` builds up the extended graph associated with the current message. The implementation of this predicate is straightforward and, hence, it is not given here. The argument `Ds` can be set to the list $[f_i, w_1]$, identifying the first cycle as the one containing the phantom node f_i and the second one containing node w_1. In this way the argument `Ls` can recollect or constrain the lengths of both cycles.

7 IMPLEMENTING `non_saturation/3`

The `diffn/1` predicate was introduced in CHIP to model constraints that frequently arise in scheduling, packing or geometric positioning problems, and will be used here to implement the predicate `non_saturation/3`.

7.1 The `diffn/1` Predicate

Suppose we want to fit n-dimensional rectangles inside an n-dimensional rectangular volume (Figure 2). Then by setting a `diffn/1` constraint one can force that no pair of rectangles intersect.

`diffn/1` has the simple structure `diffn(Rectangles)`, where `Rectangles` is a list of m n-dimensional rectangles. We define an n-dimensional rectangle as a tuple $(O_1, ..., O_n, L_1, ..., L_n)$, where O_i and L_i are either integers or domain variables and represent, respectively, the origin and the length of the rectangle in the i-th dimension. `Rectangles` has the form

$$[[O_{11}, ..., O_{1n}, L_{11}, ..., L_{1n}], ...$$
$$..., [O_{m1}, ..., O_{mn}, L_{m1}, ..., L_{mn}]]$$

`diffn/1` establishes the following constraints:

[1] $\forall i \in \{1, ..., m\}, \forall j \in \{1, ..., n\}, L_{ij} \neq 0$,

[2] $\forall i \in \{1, ..., m\}, \forall j \in \{1, ..., n\}, \quad j \neq i$, $\exists k \in \{1, ..., n\}$ such that either $O_{ik} \geq O_{jk} + L_{jk}$, or $O_{jk} \geq O_{ik} + L_{ik}$. In other words, no two any rectangles can intersect.

7.2 `diffn/1` Constraints for the Scheduling Problem

We will use `diffn/1` to implement the `non_saturation/3` predicate. For this purpose, we define a 4-dimensional space E, and a set of 4-dimensional rectangles, and we set the appropriate packing constraints on these rectangles within E using `diffn/1`. For each line in the transmission network we put a rectangle for each of the sent messages along it. In one of the directions E, the length of each of each rectangle will be equal to the bandwidth of the corresponding message. In a second dimension, the length will be the duration

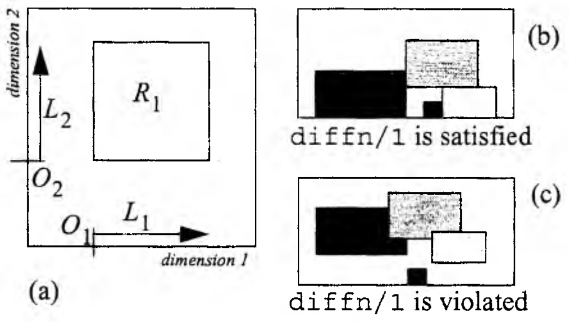

Figure 2. (a): Graphical interpretation of the parameters in *diffn/1*. (b) and (c): if there is a pair of rectangles that intersect, *diffn/1* is violated.

of the message. In the other dimensions the length will be set to 1. These elements are further described next.

Consider the transmission network $X = (G, c, w)$ at hand.

- We let b be the sum of all the bandwidths required by the messages $m_i \in M$:

$$\hat{b} = \sum_{m_i \in M} b_i$$

- We define $\hat{c}$ as the maximum capacity of any line:

$$\hat{c} = max_{a \in A}\{bw(a)\}.$$

- Let $B = max\{b, \hat{c}\}$.
- Let $V^a = \bigcup_{m_i \in M} V_i^a$

Then, E is defined as the cartesian product:

$$E = V^a \times V^a \times \{0, 1, \ldots, t_f\} \times \{0, 1, \ldots, B\}.$$

Here, the first and second dimensions are topological: we assign a point of $V^a \times V^a$ to each pair (u, v), $u, v \in V^a$. The third is a temporal dimension where each coordinate corresponds to an instant of our temporal horizon, and we represent the bandwidth in the fourth dimension.

Each message m_i is being transmitted along the path defined by $S_{i,1}, \ldots, S_{i,n}$, within the time instants T_i and $T_i + l_i$. Hence, to model the fact that each message m_i occupies a certain portion of the bandwidth of any line in X, each arc $a = (v_k, v_l)$, $v_k, v_l \in V^a$, $l = S_{i,k}$, is assigned a rectangle

$$(O_1^{ia}, \ldots, O_4^{ia}, L_1^{ia}, \ldots, L_4^{ia}),$$

for each one of the mesages flowing along it, defined as follows (Figure 3):

$$\left.\begin{array}{ll} O_1^{ia} = v_k & L_1^{ia} = 1 \\ O_2^{ia} = v_l & L_2^{ia} = 1 \\ O_3^{ia} = T_i & L_3^{ia} = l_i \\ O_4^{ia} = O_4^{ia} & L_4^{ia} = b_i \end{array}\right\}$$

Both the origin and length of the rectangle in the two first directions are fixed. The origin of the rectangle in the third dimension is equal to T_i and, thus, depends on the time instant in which message m_i begins to be transmitted. The length of the rectangle in this direction is fixed and equal to the duration l_i of the message. The origin of the fourth dimension is free, but not its length, which is equal to the bandwidth required by m_i.

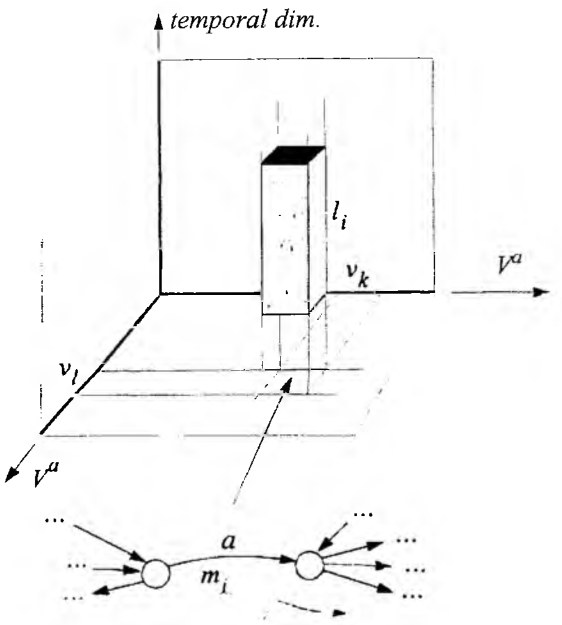

Figure 3. The space E. There is a hyper-rectangle associated with each arc a, for each of the messages flowing along it. The picture does not represent the bandwidth dimension.

In order to reflect the fact that each line disposes only a limited capacity, which can be different from one line to another, we introduce several auxiliary 4-dimensional "limiting" rectangles. We will fix their positions in such a way that the position of the rest of 4-dimensional rectangles (those corresponding to messages), be restricted in its fourth dimension. For each arc $a = (v_k, v_l)$, $v_k, v_l \in V^a$, $l = S_{i,k}$ we define a rectangle

$$(O_1^{0a}, \ldots, O_4^{0a}, L_1^{0a}, \ldots, L_4^{0a})$$

where:

$$\left.\begin{array}{ll} O_1^{0a} = v_k & L_1^{0a} = 1 \\ O_2^{0a} = v_l & L_2^{0a} = 1 \\ O_3^{0a} = 0 & L_3^{0a} = t_f \\ O_4^{0a} = 1 & \\ L_4^{0a} = \begin{cases} B - bw(b), \text{if } (v_l \in V^a) \\ B - \hat{b}, \text{otherwise} \end{cases} & \end{array}\right\}$$