RAIRO Operations Research
RAIRO Oper. Res. **37** (2003) 221-234
DOI: 10.1051/ro:2004002

MINIMUM CONVEX-COST TENSION PROBLEMS ON SERIES-PARALLEL GRAPHS

Bruno Bachelet[1] and Philippe Mahey[1]

Abstract. We present briefly some results we obtained with known methods to solve minimum cost tension problems, comparing their performance on non-specific graphs and on series-parallel graphs. These graphs are shown to be of interest to approximate many tension problems, like synchronization in hypermedia documents. We propose a new *aggregation* method to solve the minimum convex piecewise linear cost tension problem on series-parallel graphs in $O(m^3)$ operations.

Keywords. Minimum cost tension, convex piecewise linear costs, series-parallel graphs.

Introduction

The exploding use of Internet and of hypermedia documents have turned crucial the necessity to dispose of robust on-line algorithms to manage complexity and interactivity. One of the resulting problems which has emerged recently is the synchronization of hypermedia documents by considering that each object can be compressed or delayed like an elastic spring. The heterogeneity of the objects that compose a hypermedia document turns their presentation in time and space a hard problem. On the other hand, interactivity means that real-time updates of the schedule of the document should be possible, increasing the need for faster decision-making algorithms.

As explained in [9] and [17], such documents are composed of media objects (audio, video, text, image...), which duration of presentation must be adjusted to satisfy a set of temporal constraints that express the progress of the animation as

[1] LIMOS, UMR 6158-CNRS, Université Blaise-Pascal, BP 10125, 63173 Aubière, France; e-mail: bachelet@isima.fr, mahey@isima.fr

defined by the author. But for these constraints to be satisfied, the author must accept some flexibility on the duration (that we call *ideal*) of presentation of each object, pauses being totally forbidden if not explicitly wanted. To estimate the quality of an adjustment, a cost function, usually convex (*cf.* Fig. 1), is introduced for each object. To summarize, the problem we attempt to solve here is to find an adjustment of best quality, *i.e.* which minimizes the sum of the costs of the media objects.

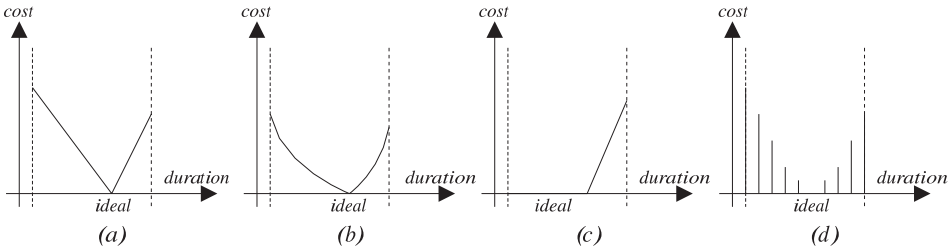

FIGURE 1. Examples of cost functions. (a) Piecewise linear with a single ideal value. (b) Non-linear, but convex and derivable. (c) Piecewise linear with several ideal values. (d) Discrete values.

The set of temporal constraints can be modeled as a directed graph $G = (X; U)$ (*cf.* [6]) where X is a set of nodes, U a set of arcs, $m = |U|$ and $n = |X|$. The nodes represent events (the start or the end of presentation of an object). The arcs express duration constraints between nodes. With each arc u is associated a duration interval $[a_u; b_u]$, an ideal duration o_u and a cost function c_u defined on the interval. An arc $u = (x; y)$ between two nodes x and y means the event x precedes y and they are separated by a duration θ_u between a_u and b_u, the ideal value being o_u. Figure 2 shows how to represent some of the main temporal relations used in hypermedia synchronization (introduced by [3]).

Let $\pi : X \mapsto \mathbb{R}$ be a potential function which assigns a date to each event node of the graph. Then the duration θ_u of an object associated with an arc $u = (x; y)$ can be seen as the difference of potentials $\theta_u = \pi_y - \pi_x$, in other words, $\theta = (\theta_u)_{u \in U}$ is a tension vector on the graph (*e.g.* [7]). Denoting by A the incidence matrix of the graph, *i.e.* matrix A of dimension $(m \times n)$ with the elements a_{xu} equal to -1 (if u leaves x), $+1$ (if u comes to x) or 0 (any other case), the problem is simply formulated as following:

$$(P) \begin{cases} \text{minimize } \displaystyle\sum_{u \in U} c_u(\theta_u) \\ \\ \text{with } \theta = A^T \pi,\ a \leq \theta \leq b. \end{cases}$$

And let T_G be the set of feasible tensions, *i.e.* $T_G = \{\theta \in \mathbb{R}^m \mid \theta = A^T \pi,\ a \leq \theta \leq b\}$. In this article, we only consider convex two-piecewise linear cost functions as shown in Figure 1a, the adaptation to more pieces of the method described here

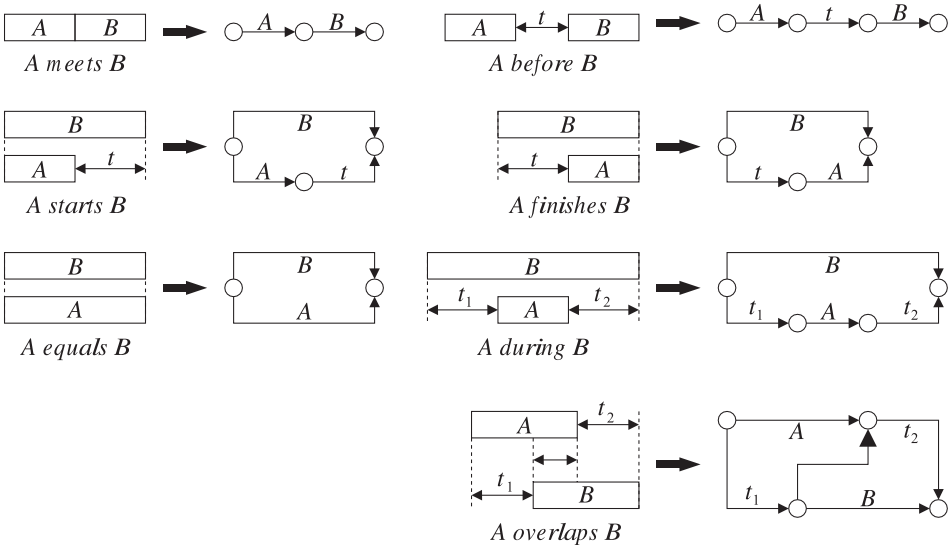

FIGURE 2. Graph representation of temporal constraints.

is straightforward (Sect. 3 shows that the core of the method manages piecewise linear costs with more than two pieces, so only the initialization phase of the method has to be adapted). Hence, from now on we consider the cost functions c_u as following:

$$c_u(\theta_u) = \begin{cases} c_u^1(o_u - \theta_u) \text{ , if } \theta_u < o_u \\ c_u^2(\theta_u - o_u) \text{ , if } \theta_u \geq o_u. \end{cases}$$

In Section 1, we present the results we obtained with known methods to solve the minimum cost tension problem on graphs with non-specific structure. Then in Section 2, we recall and introduce some properties of the series-parallel graphs related to tension. Section 3 explains the *aggregation* method. Numerical results of this method and comparisons with the previous methods are presented and discussed in Section 4. The last section ends this article with our first thoughts on how to exploit this method on non series-parallel graphs.

1. Minimum cost tension problem

With convex piecewise linear costs, it is possible to model the problem with linear programs. It is a solution widely used in practice for the synchronization problem (*e.g.* [10, 17]). Another way to solve the problem is the *out-of-kilter* algorithm first introduced for the minimum cost flow problem [14] and then for the minimum cost tension problem in [18]. We present an adaptation of that method to piecewise linear costs in [5]. That algorithm is pseudo-polynomial, $O(m^2(A+B))$ operations where $A = \max_{u \in U}\{a_u; b_u\}$ and $B = \max_{u \in U}\{c_u^1; c_u^2\}$. A polynomial method is presented in [15] but is only really efficient in practice for a special class

TABLE 1. Numerical results on non-specific graphs.

Nodes	Arcs	CPLEX	Kilter	Cost-Scaling
50	200	0.44	0.12	0.1
50	400	0.83	0.3	0.19
100	400	0.93	0.47	0.28
100	800	2	1.3	0.54
500	2000	12.5	15.4	3.5
500	4000	37.7	49.1	6,8
1000	4000	57.2	76.5	11.6
1000	8000	193.7	239.9	20.4

of graphs (*Penelope*'s graphs). More recently, [1] presents an algorithm to solve a more generic problem called the *convex cost integer dual network flow problem*, the algorithm consists in transforming the minimum cost tension problem into a minimum cost flow problem, solved with the well-known *cost-scaling* method (*e.g.* [2]). This algorithm is polynomial, $O(mn^2 \log nA)$ operations, and proves to be very efficient in practice.

Table 1 aims at a practical comparison of the methods, which is always tricky because of all kinds of biases. But the goal here is to get an idea of how the methods behave on graphs with non-specific structure. Later in this article, we show the performance of these very same implementations on series-parallel graphs. Results are expressed in seconds, obtained on a RISC 6000 / 160 MHz processor with an AIX Unix operating system. We use GNU C++ 2.95 compiler and its object-oriented features to implement the methods. For the linear programming, we use the simplex method provided in CPLEX 6.0 software. These results are the means of series of 10 tests on randomly generated graphs. Both A and B are fixed to 1000. The implementation of the methods and the generation of the graphs are available in [4].

2. SERIES-PARALLEL GRAPHS

A common definition of series-parallel graphs is based on a recursive construction of these graphs (*e.g.* [12, 13, 22]) that is very intuitive and close to the way synchronization constraints are built in a hypermedia document.

A graph is *series-parallel*, also called *SP-graph*, if it is obtained from a graph with only two nodes linked by an arc, applying recursively the two following operations:

- the *series composition*, applied upon an arc $u = (x; y)$, creates a new node z and replaces u by two arcs $u_1 = (x; z)$ and $u_2 = (z; y)$ (*cf.* Fig. 3a). We call *series* the relation that binds u_1 and u_2 and note it $u_1 + u_2$;
- the *parallel composition*, applied upon an arc $u = (x; y)$, duplicates u by creating a new one $v = (x; y)$ (*cf.* Fig. 3b). We call *parallel* the relation that binds u and v and note it $u//v$.

We regroup the series and parallel relations under the term *SP-relation*. During the construction process, a SP-relation that binds two arcs can become a relation

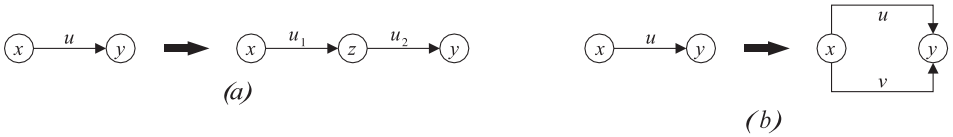

FIGURE 3. Series and parallel compositions.

between two series-parallel subgraphs. Hence, we introduce the term *single SP-relation* to identify a SP-relation between two arcs. From the recursive definition of a SP-graph, it is easy to verify that a SP-graph has always a single SP-relation (the SP-relation created from the last composition). Hence, it is easy to check if a graph is series-parallel: find a single SP-relation in the graph, apply a reduction reverse to the composition that produces the SP-relation and go on again until only one arc remains in the graph. This linear-time method is explained in [22] and [20]. Another efficient approach to recognize a SP-graph is proposed in [13], based on the fact that paths in SP-graphs are organized a certain way.

The SP-relations are binary operations, so we can represent a SP-graph by a binary tree called *decomposition binary tree* or *SP-tree* (*cf.* [11, 22]). Figure 4 shows a SP-tree of an SP-graph. All the algorithms cited earlier to recognize a SP-graph can be adapted, without any complexity loss, to build a SP-tree during their process. Hence we will use this representation to present our *aggregation* method.

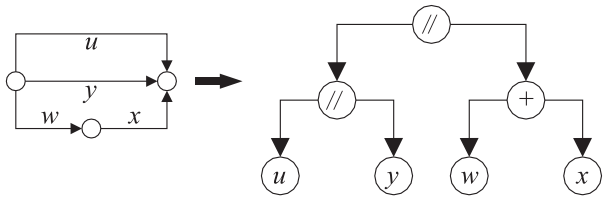

FIGURE 4. Example of SP-tree.

From the definition of a SP-graph, it is obvious that a SP-graph has only one *source* node (*i.e.* without any predecessor) and only one *target* node (*i.e.* without any successor). Hence we define the *main tension* $\overline{\theta}$ of a graph as the tension between its source s and target t, *i.e.* $\overline{\theta} = \pi_t - \pi_s$.

3. AGGREGATION METHOD

We present here the *aggregation* method to solve the minimum cost tension problem with convex piecewise linear cost functions (*cf.* Fig. 1a) on an SP-graph G. Note that the resolution of an optimization problem on this kind of graphs is usually easier than on non-specific graphs (*e.g.* [8, 11, 21]).

The aggregation method works on an SP-tree T of the SP-graph G. The method is recursive: considering an SP-relation in T, it supposes that the optimal tensions of the two subgraphs implied in the relation are known, and from them it is possible to quickly build the optimal tension of the whole SP-relation. Hence, starting from the leaves of T, the optimal tension of each SP-relation is built to finally reach the root of the tree T.

To get an efficient algorithm, we need what we call the *minimum cost function* C_G of a SP-graph G. This function represents the cost of the optimal tension (as defined by linear program (P) in the introduction) where the main tension is forced to a given value.

$$C_G(x) = \min \left\{ \sum_{u \in U} c_u(\theta_u) \mid \theta \in T_G,\, \bar{\theta} = x \right\}.$$

As each function c_u is convex, the minimum cost function is indeed convex (assuming that $C_G(x) = +\infty$ if no feasible tension exists such that the main tension is forced to x).

We consider two series-parallel subgraphs G_1 and G_2, and suppose that their minimum cost functions C_{G_1} and C_{G_2} are known. If we look at the SP-relation $G_1 + G_2$ (*cf.* Fig. 5a), G_1 and G_2 only share one node, hence there is no tension constraints between them. But if we add the constraint that the main tension of $G_1 + G_2$ must be equal to x, it imposes to x_1 and x_2, the main tensions of G_1 and G_2, that $x = x_1 + x_2$. Hence, the minimum cost function $C_{G_1+G_2}$ of the SP-relation $G_1 + G_2$ is:

$$C_{G_1+G_2}(x) = \min_{x=x_1+x_2} C_{G_1}(x_1) + C_{G_2}(x_2).$$

It means $C_{G_1+G_2}$ is the inf-convolution $C_{G_1} \square C_{G_2}$. It is well-known that this operation maintains convexity (*e.g.* [19]).

If we look now at the SP-relation $G_1//G_2$ (*cf.* Fig. 5b), G_1 and G_2 share their source and target nodes, hence the only tension constraint between them is that their main tensions x_1 and x_2 must be equal. If we add the constraint that the main tension of $G_1//G_2$ must be x, then it imposes $x = x_1 = x_2$. Hence, the minimum cost function $C_{G_1//G_2}$ of the SP-relation $G_1//G_2$ is:

$$C_{G_1//G_2}(x) = C_{G_1}(x) + C_{G_2}(x).$$

It means $C_{G_1//G_2}$ is simply the sum $C_{G_1} + C_{G_2}$, which is convex if C_{G_1} and C_{G_2} are convex. From our analysis of the two SP-relations, it is easy to write an algorithm that builds the minimum cost function C_G of a SP-graph G. But what interests us is to find the minimum cost tension of G. We propose now a specific way to represent the minimum cost functions so we know not only the cost of the optimal tension of a SP-relation, but also how to build it.

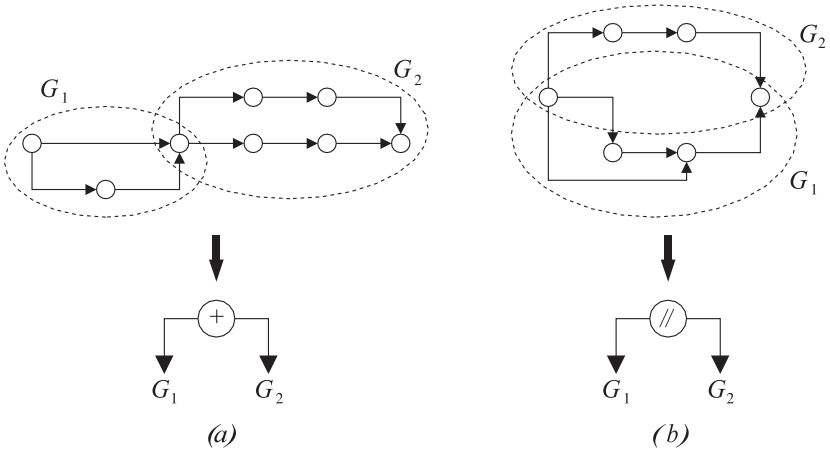

FIGURE 5. SP-relations between subgraphs.

For this purpose, we define the *t-centered minimum cost function* C_G^t of G as following:

$$C_G^t(x) = C_G(x+t) - C_G(t).$$

That means $C_G^t(0) = 0$ and the function represents the minimum cost to increase or decrease the main tension from the value t.

We choose to represent this piecewise function with two sets sh_G^t and st_G^t, sh_G^t, called the *shrinking* set, represents C_G^t on the interval $]-\infty; 0[$, and st_G^t, called the *stretching* set, represents C_G^t on the interval $]0; +\infty[$. These sets simply contain the definition of each piece of the function on the interval they represent. They contain triplets of the form $(c; e; L)$ where c represents the slope of the curve, e the length of the interval on which the piece is defined and L is a set of arcs (there can be several sets because more than one tension may be optimal, *i.e.* more than one way may exist to optimize the tension) that must be increased or decreased to adapt the tension on this piece. For efficiency reasons, the triplets are sorted from the smallest slope to the highest. Here are the sets st_G^t and sh_G^t of the example of Figure 6 (where a, b, c, d and e are arcs of G):

$$sh_G^t = \{(-2/5; 5; \{a; b\}); (3/5; 10; \{c\}); (3; 5; \{d; e\})\}$$
$$st_G^t = \{(2/5; 5; \{a; b\}); (2; 7; \{c; e\})\}.$$

For instance, if we want to decrease the main tension t, we consider the first triplet of the shrinking set sh_G^t, *i.e.* $(-2/5; 5; \{a; b\})$, which means the tension t can be decreased by a value between 0 and 5, with a unit cost 2/5, by decreasing the tension of the arcs a and b by the same value. If the tension t must be decreased by more than 5 units, then we consider the second triplet of sh_G^t, and so on until the whole decrease is performed.

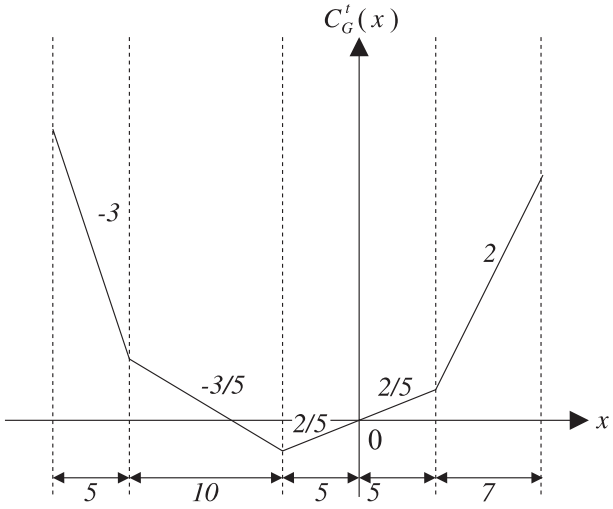

FIGURE 6. Example of t-centered minimum cost function.

Let us note θ_G^* the minimum cost tension of a graph G and $C_G^* = C_G^{\overline{\theta_G^*}}$. We explain here how to find θ_G^* and build the C_G^* function. First the C_u^* function of an arc u is represented by $sh_u^* = \{(c_u^1; o_u - a_u; \{u\})\}$ and $st_u^* = \{(c_u^2; b_u - o_u; \{u\})\}$ with the optimal tension $\theta_u^* = o_u$.

3.1. Series aggregation

Now consider the graph $G = G_1 + G_2$ and suppose that we know the optimal tensions θ_1^* and θ_2^* and the minimum cost functions C_1^* and C_2^* of the subgraphs G_1 and G_2. The tension θ_G^* of G made of the two tensions θ_1^* and θ_2^* is optimal, because there is no constraint between the two subgraphs after the series composition, which means the minimum cost function C_G^* is centered on $\overline{\theta_G^*}$. To increase $\overline{\theta_G^*}$ we can choose to increase either $\overline{\theta_1^*}$ or $\overline{\theta_2^*}$. If we look at $p_1 = (c_1; e_1; L_1)$ and $p_2 = (c_2; e_2; L_2)$, the first pieces of the stretching sets st_1^* and st_2^*, we decide to increase $\overline{\theta_1^*}$ if $c_1 < c_2$ or else $\overline{\theta_2^*}$. The same reasoning can be made to decrease $\overline{\theta_G^*}$. This way the whole C_G^* function is built.

We can conclude that to build the function $C_G^* = C_1^* \,\square\, C_2^*$ of the graph $G = G_1 + G_2$, we need to create $sh_G^* = sh_1^* \cup sh_2^*$ and $st_G^* = st_1^* \cup st_2^*$ sorted from the smallest slope to the highest. Figure 7 shows an example. If we note p_1 and p_2 the numbers of pieces of C_1^* and C_2^*, then C_G^* has $p = p_1 + p_2$ pieces, and the process of finding the optimal tension of a series composition needs $O(pm)$ operations ($O(p)$ operations to go through the p pieces and $O(m)$ to copy a set of at most m arcs for each piece).

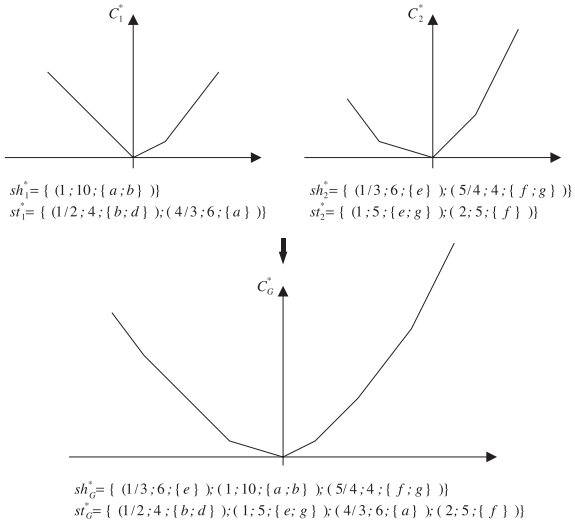

FIGURE 7. Example of minimum cost function of a series composition.

3.2. PARALLEL AGGREGATION

We consider now the graph $G = G_1//G_2$ and suppose that we know the optimal tensions θ_1^* and θ_2^* and the minimum cost functions C_1^* and C_2^* of the subgraphs G_1 and G_2. The parallel composition is possible only if $\overline{\theta_1^*} = \overline{\theta_2^*}$ (if we want to get a valid tension). As we need to find the optimal tension θ_G^* of the graph G, we need a method to equalize $\overline{\theta_1^*}$ and $\overline{\theta_2^*}$ optimally, *i.e.* such that the tension θ_G^* made of θ_1^* and θ_2^* is optimal. Suppose that $\overline{\theta_1^*} < \overline{\theta_2^*}$, to equalize $\overline{\theta_1^*}$ and $\overline{\theta_2^*}$ we can increase $\overline{\theta_1^*}$ and/or decrease $\overline{\theta_2^*}$, so we look at $p_1 = (c_1; e_1; L_1)$ and $p_2 = (c_2; e_2; L_2)$, the first pieces of st_1^* and sh_2^*. We decide then to increase $\overline{\theta_1^*}$ if $c_1 < c_2$ or else to decrease $\overline{\theta_2^*}$. This process is repeated until $\overline{\theta_1^*} = \overline{\theta_2^*}$ (*cf.* Algorithm 8 (Fig. 8) and Fig. 11).

The tension θ_G^* made of θ_1^* and θ_2^* obtained with Algorithm 8 (Fig. 8) is compatible, *i.e.* belongs to T_G, because modifications on G_1 or G_2 satisfy the constraints on the tension. θ_G^* is also optimal for $G = G_1//G_2$ so the function C_G^* will be centered on its main tension $\overline{\theta_G^*}$.

Then to build the function $C_G^* = C_1^* + C_2^*$ we use the procedure 10 (only detailed for the st_G^* part of the function) that is illustrated by Figure 9. If we note p_1 and p_2 the numbers of pieces of C_1^* and C_2^*, then C_G^* has at most $p = p_1 + p_2$ pieces, and the whole process of finding the optimal tension of a parallel composition needs $O(pm)$ operations (the equalization process go through at most p pieces and copies at most m arcs for each piece, the same for Algorithm 10 (Fig. 10) that creates at most p pieces and copies for each at most m arcs).

```
while θ*1 < θ*2 do
 if st*1 = ∅ and sh*2 = ∅ then /* no feasible tension */;
 let p1 = (c1; e1; L1) be the first piece of st*1 if st*1 ≠ ∅;
 let p2 = (c2; e2; L2) be the first piece of sh*2 if sh*2 ≠ ∅;

 if sh*2 = ∅ or c1 < c2 then /* tension increase */
  λ ← min{e1; θ*2 − θ*1};
  for each u ∈ L1 do θ*1u ← θ*1u + λ;
  st*1 ← st*1 − {p1};
  if λ < e1 then st*1 ← st*1 ∪ {(c1; e1 − λ; L1)};
  sh*1 ← sh*1 ∪ {(−c1; λ; L1)};
 else /* tension decrease */
  λ ← min{e2; θ*2 − θ*1};
  for each u ∈ L2 do θ*2u ← θ*2u − λ;
  sh*2 ← sh*2 − {p2};
  if λ < e2 then sh*2 ← sh*2 ∪ {(c2; e2 − λ; L2)};
  st*2 ← st*2 ∪ {(−c2; λ; L2)};
 end if;
end while;
```

FIGURE 8. Algorithm to equalize parallel tensions.

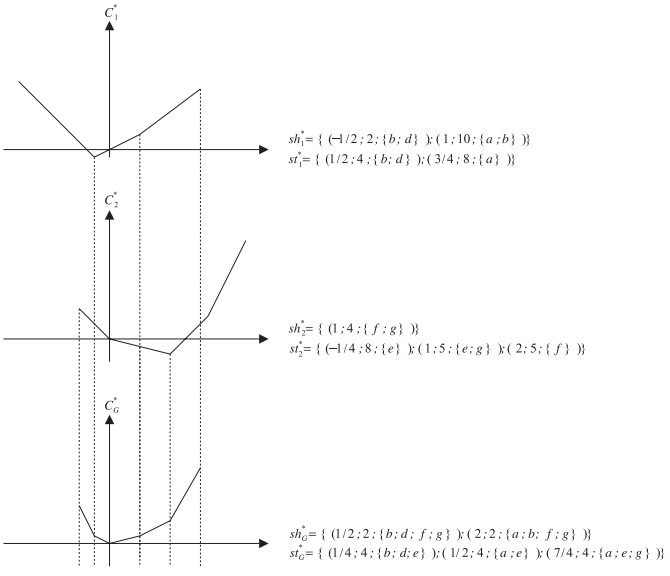

FIGURE 9. Example of minimum cost function of a parallel composition.

```
while st*1 ≠ ∅ and st*2 ≠ ∅ do
 let p1 = (c1; e1; L1) be the first piece of st*1;
 let p2 = (c2; e2; L2) be the first piece of st*2;
 λ ← min{e1; e2};
 st*G ← st*G ∪ {(c1 + c2; λ; L1 ∪ L2)};
  st*1 ← st*1 − {p1};
  st*2 ← st*2 − {p2};
  if e1 > λ then st*1 ← st*1 ∪ {(c1; e1 − λ; L1)};
  if e2 > λ then st*2 ← st*2 ∪ {(c2; e2 − λ; L2)};
end while;
```

FIGURE 10. Algorithm to build the optimal tension of a parallel composition.

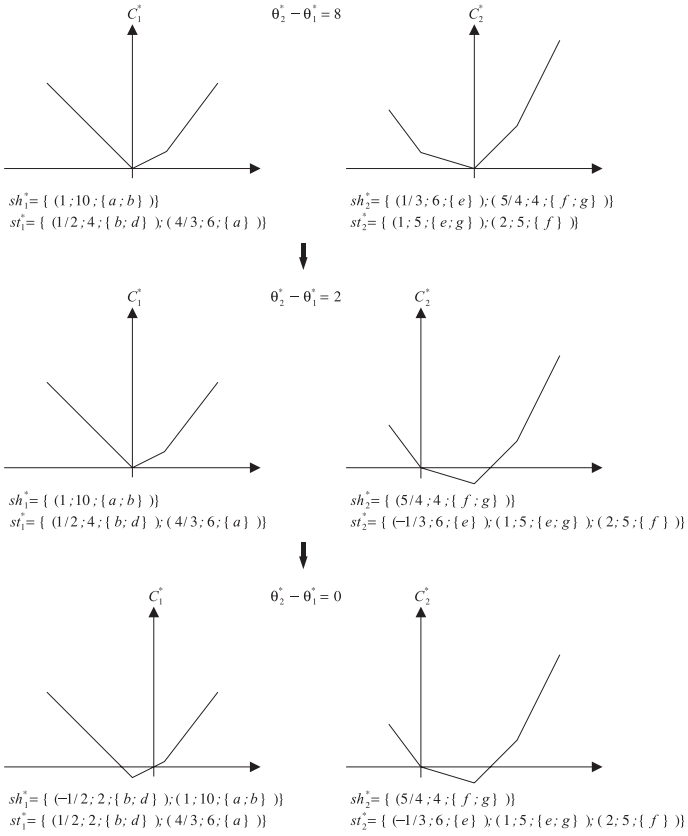

FIGURE 11. Example of equalization of parallel tensions.

```
algorithm aggregate(Tree T = (o, T_l, T_r),Tension θ*_T,Function C*_T)
 if T_l ≠ ∅ then aggregate(T_l,θ*_l,C*_l);
 if T_r ≠ ∅ then aggregate(T_r,θ*_r,C*_r);

 if o = + then build C*_T and θ*_T of the series composition T_l + T_r;
 else if o = // then build C*_T and θ*_T of the parallel composition T_l//T_r;
 else sh*_T ← {(c¹_u; o_u − a_u; {u})}; st*_T ← {(c²_u; b_u − o_u; {u})};
end algorithm;
```

FIGURE 12. Algorithm to build the optimal tension of a SP-graph.

3.3. Complexity

Finally Algorithm 12 (Fig. 12) resumes the whole aggregation method, which, from the leaves of the SP-tree of the graph, applies the series and parallel compositions with the construction of the optimal tension and the minimum cost function as we just explained in the previous paragraphs.

In this algorithm $T = (o; T_l; T_r)$ is the SP-tree with the root o, the left subtree T_l and the right subtree T_r. We show now that this recursive method has a polynomial complexity.

Theorem 3.1. *The aggregation algorithm performs $O(m^3)$ operations.*

We established that each composition needs $O(pm)$ operations. It is known that a SP-graph contains $m - 1$ SP-relations ($n - 2$ series relations because each one creates a node and there are only two nodes at the beginning of the construction process, and $m - n + 1$ parallel relations because any SP-relation creates an arc and there is only one arc at the beginning). So the aggregation needs $O(pm^2)$ operations. We explained earlier that for each composition, if p_1 and p_2 are the numbers of pieces for C_1^* and C_2^*, C_G^* has at most $p_1 + p_2$ pieces. That means if each arc has a two-piecewise cost function, the minimum cost function of the whole graph has at most $2m$ pieces, and the aggregation needs $O(m^3)$ operations.

4. Numerical results

Table 2 shows numerical results of the methods presented in Section 1 and the aggregation algorithm on series-parallel graphs. Details on how the tests were performed can be found in the comments of Table 1 (Sect. 1). These tests still focus on the size and density of the graphs, because the theoretical complexity of the aggregation method shows that the scale of the data (*i.e.* the tension boundaries and the unit costs) has no impact on this method, contrary to the out-of-kilter and the dual cost-scaling approaches that are affected (*cf.* Sect. 1).

The linear programming and the out-of-kilter methods take advantage of the particular structure of the SP-graphs and behave really better on this class of graphs. However the cost-scaling approach on the dual of the problem does not work that well on this kind of instances, even with an improvement technique like

TABLE 2. Numerical results on series-parallel graphs.

Nodes	Arcs	CPLEX	Kilter	Cost-Scaling	Aggregation
50	200	0.4	0.07	0.09	0.05
50	400	0.71	0.15	0.2	0.09
100	400	0.73	0.2	0.31	0.09
100	800	1.4	0.38	0.63	0.18
500	2000	4.4	3	4.5	0.5
500	4000	10.7	5.3	11.3	0,99
1000	4000	11.6	9	17.2	1
1000	8000	30.8	21.5	34.5	2.1

the *wave implementation* (*cf.* [2]). The aggregation method reveals quite efficient, and not very sensitive to the graph dimension.

CONCLUSION

We show here how to solve the minimum cost tension problem on series-parallel graphs with convex piecewise linear costs in $O(m^3)$ operations. But the real instances that interest us for the hypermedia synchronization are a bit more complex than the SP-graphs. They are indeed related to the *generalized series-parallel* graphs (*cf.* [16]). Through this article, we explained that, in the context of the minimum cost tension problem, a SP-graph can be reduced to a single arc with a convex piecewise linear cost function. One idea can be to identify series-parallel subgraphs of a non-specific graph, to aggregate these subgraphs and solve then the minimum cost tension problem on the reduced graph with any known method. To improve this simple idea, our future work will be to find algorithms to extract *series-parallel components* from a graph and to develop an efficient process (using the aggregation method) to find the minimum cost tension of the whole graph when assembling back these components.

REFERENCES

[1] R.K. Ahuja, D.S. Hochbaum and J.B. Orlin, Solving the Convex Cost Integer Dual Network Flow Problem. *Lect. Notes Comput. Sci.* **1610** (1999) 31-44.
[2] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows – Theory, Algorithms, and Applications*. Prentice Hall (1993).
[3] J.F. Allen, Maintaining Knowledge about Temporal Intervals. *Commun. ACM* **26** (1983) 832-843.
[4] B. Bachelet, B++ Library. `http://frog.isima.fr/bruno/?doc=bpp_library`
[5] B. Bachelet and P. Mahey, Optimisation de la présentation d'un document hypermédia. *Ann. Sci. Univ. Blaise Pascal* **110** (2001) 81-90.
[6] B. Bachelet, P. Mahey, R. Rodrigues and L.F. Soares, Elastic Time Computation for Hypermedia Documents. *SBMidìa* (2000) 47-62 .
[7] C. Berge and A. Ghoula-Houri, *Programmes, jeux et réseaux de transport*. Dunod (1962).
[8] M.W. Bern, E.L. Lawler and A.L. Wong, Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs. *J. Algorithms* **8** (1987) 216-235.
[9] M.C. Buchanan and P.T. Zellweger, Specifying Temporal Behavior in Hypermedia Documents. *Eur. Conference on Hypertext '92* (1992) 262-271.

[10] M.C. Buchanan and P.T. Zellweger, Automatically Generating Consistent Schedules for Multimedia Documents. *Multimedia Systems* (1993) 55-67.
[11] A.K. Datta and R.K. Sen, An Efficient Scheme to Solve Two Problems for Two-Terminal Series Parallel Graphs. *Inform. Proc. Lett.* **71** (1999) 9-15.
[12] R.J. Duffin, Topology of Series-Parallel Networks. *J. Math. Anal. Appl.* **10** (1965) 303-318.
[13] D. Eppstein, Parallel Recognition of Series-Parallel Graphs. *Inform. Comput.* **98** (1992) 41-55.
[14] D.R. Fulkerson, An Out-of-Kilter Method for Minimal Cost Flow Problems. *SIAM J. Appl. Math.* **9** (1961) 18-27.
[15] M. Hadjiat, Penelope's Graph: a Hard Minimum Cost Tension Instance. *Theoret. Comput. Sci.* **194** (1998) 207-218.
[16] C.W. Ho, S.Y. Hsieh and G.H. Chen, Parallel Decomposition of Generalized Series-Parallel Graphs. *J. Inform. Sci. Engineer.* **15** (1999) 407-417.
[17] M.Y. Kim and J. Song, Multimedia Documents with Elastic Time. *Multimedia '95* (1995) 143-154.
[18] J.M. Pla, An Out-of-Kilter Algorithm for Solving Minimum Cost Potential Problems. *Math. Programming* **1** (1971) 275-290.
[19] R.T. Rockefellar, *Convex Analysis*. Princeton University Press (1970).
[20] B. Schoenmakers, *A New Algorithm for the Recognition of Series Parallel Graphs*. Technical report, No CS-59504, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands (1995).
[21] K. Takamizawa, T. Nishizeki and N. Saito, Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs. *J. ACM* **29** (1982) 623-641.
[22] J. Valdes, R.E. Tarjan and E.L. Lawler, The Recognition of Series Parallel Digraphs. *SIAM J. Comput.* **11** (1982) 298-313.