Aggregation Approach for the Minimum Binary Cost Tension Problem

Bruno Bachelet, Christophe Duhamel

Aggregation Approach for the Minimum Binary Cost Tension Problem

Bruno Bachelet[1] and Christophe Duhamel[2]
LIMOS, UMR 6158-CNRS,
Université Blaise-Pascal, BP 10125, 63173 Aubière, France.

Research Report LIMOS/RR04-08

Revised December 2005

[1]bruno.bachelet@isima.fr - http://frog.isima.fr/bruno
[2]christophe.duhamel@isima.fr - http://www.isima.fr/~duhamel

Abstract

The aggregation technique, dedicated to two-terminal series-parallel graphs (or TTSP-graphs) and introduced lately to solve the minimum piecewise linear cost tension problem, is adapted here to solve the minimum binary cost tension problem (or *BCT* problem). Even on TTSP-graphs, the BCT problem has been proved to be NP-complete. As far as we know, the aggregation is the only algorithm, with mixed integer programming, proposed to solve exactly the BCT problem on TTSP-graphs. A comparison of the efficiency of both methods is presented here.

Keywords: minimum cost tension, binary costs, two-terminal series-parallel graphs.

Abstract

La technique d'agrégation, dédiée aux graphes série-parallèles et introduite récemment pour résoudre le problème de la tension minimum à coûts linéaires par morceaux, est adaptée ici pour résoudre le problème de la tension minimum à coûts binaires (ou problème *BCT*). Même sur des graphes série-parallèles, le problème BCT a été prouvé NP-complet. A notre connaissance, l'agrégation est le seul algorithme, avec la programmation linéaire mixte, proposé pour résoudre de manière exacte le problème BCT sur des graphes série-parallèles. Une comparaison de l'efficacité des deux méthodes est présentée ici.

Mots clés : tension de coût minimum, coûts binaires, graphes série-parallèles.

Abstract

The aggregation technique, dedicated to two-terminal series-parallel graphs (or TTSP-graphs) and introduced lately to solve the minimum piecewise linear cost tension problem, is adapted here to solve the minimum binary cost tension problem (or *BCT* problem). Even on TTSP-graphs, the BCT problem has been proved to be NP-complete. As far as we know, the aggregation is the only algorithm, with mixed integer programming, proposed to solve exactly the BCT problem on TTSP-graphs. A comparison of the efficiency of both methods is presented here.

1 Introduction

The study of tension problems in graphs is motivated here by synchronization problems in hypermedia documents [2]. These documents are composed of various media objects such as audio, video, text, image, applet... Authors need powerful tools to schedule automatically the temporal specifications of these objects in a document. Any media object u has an ideal duration o_u and an interval $[a_u; b_u]$ in which its effective (i.e. scheduled) duration can vary. The author also specifies temporal constraints in order to express the way the presentation of the document should happen. The problem is finally to schedule the duration of each media object so that it satisfies both the tolerance intervals and the temporal constraints.

This problem can be interpreted as a *minimum cost tension* problem (or *MCT* problem) in a graph [6]. Let $G = (X; U)$ be a graph, with X the set of nodes, U the set of arcs, $m = |U|$ and $n = |X|$. The nodes represent events in the hypermedia presentation (i.e. the start or end of presentation of a media object), and the arcs express temporal constraints between two events (i.e. precedence and duration between two events). Let $\pi : X \longmapsto \mathbb{R}$ be a function that assigns a potential to each node of the graph. It represents the date scheduled for each event. Hence, the tension θ_u of an arc $u = (x; y)$, which is the difference of potentials $\theta_u = \pi_y - \pi_x$, is the duration between events x and y, and is constrained to $\theta_u \in [a_u; b_u] \subset \mathbb{R}$. The minimum cost tension problem can finally be modeled as:

$$(P_{MCT}) \begin{cases} \text{minimize} & \displaystyle\sum_{u \in U} c_u(\theta_u) \\ \text{with} & \pi_y - \pi_x = \theta_{(x;y)},\ \forall (x; y) \in U \\ & a_u \leq \theta_u \leq b_u,\ \forall u \in U \end{cases}$$

To measure the quality of a document, many proposals were made for c_u. The first studies consider piecewise linear costs, with a minimum for o_u [7, 11, 2]. The problem is thus expressed as a linear program, and many polynomial algorithms were developed for this specific problem [10, 1]. [4] proposed an *aggregation* method to solve the *minimum convex piecewise linear cost tension* problem (or *CPLCT* problem) on *two-terminal series-parallel* graphs (or *TTSP-graphs*). It was shown to be competitive on this class of graphs with the best *dual cost-scaling* algorithms [1].

However, the number of objects that need to be modified (i.e. that are not scheduled at their ideal duration) is also relevant for hypermedia synchronization [12]. Altering the duration of a media object is CPU consuming, thus in a real time context, minimizing this operation is important. We propose here to develop an aggregation method for the *minimum binary cost tension* problem (or *BCT* problem) on TTSP-graphs, where the cost functions are defined as:

$$c_u(\theta_u) = \begin{cases} 0, \text{ if } \theta_u = o_u \\ 1, \text{ if } \theta_u \neq o_u \end{cases} \quad (1)$$

Due to its discrete nature, this problem is NP-complete [8]. Lately, it was also proved that it is NP-complete on TTSP-graphs [13]. Our interest in this class of graphs is explained by the specific structure of the temporal constraints used in hypermedia synchronization, which leads us to manipulate graphs that are very close to TTSP-graphs [4].

Section 2 proposes an overview of the TTSP-graphs and the aggregation approach. Section 3 explains how to adapt the method to the BCT problem. Then, comparative numerical results with mixed integer programming are presented in Section 4. Section 5 shows that the aggregation algorithm can be used to solve the BCT problem on any elementary cycle, a subproblem that can be significant to solve the BCT problem on any graph (i.e. without any specific structure) [3]. We conclude with a discussion on some leads to solve the BCT problem on *quasi-k series-parallel* graphs and maybe on any graph, using the aggregation technique.

2 Aggregation Method

2.1 Two-Terminal Series-Parallel Graphs

A common definition of TTSP-graphs is based on a recursive construction of the graphs (e.g. [9], or [15] with the *edge series-parallel* digraphs) that is very intuitive and close to the way synchronization constraints are built in a hypermedia document. A digraph G is *two-terminal series-parallel*, also called *TTSP-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. Figure 1a). We call *series* the relation that binds u_1 and u_2 and denote 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. Figure 1b). We call *parallel* the relation that binds u and v and denote it $u//v$.

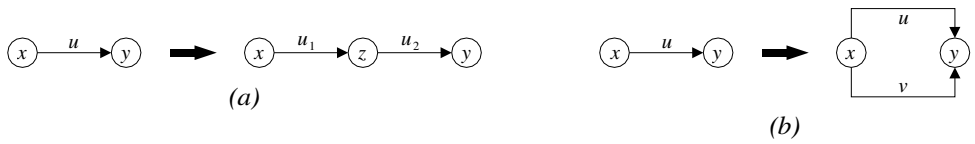


Figure 1: Series and parallel compositions.

The series and parallel relations are gathered under the term *SP-relations*. During the construction process, a SP-relation that binds two arcs can become a relation between two TTSP-subgraphs. The SP-relations are binary operations, so we can represent a TTSP-graph by a binary tree called *decomposition binary tree* [8] or *SP-tree*, as illustrated by Figure 2. [15, 14, 9, 5] propose different ways to find such a tree in linear time.

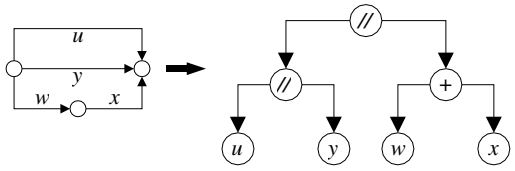


Figure 2: Example of SP-tree.

For the CPLCT problem, numerical results in [4] show that linear programming and the out-of-kilter method take advantage of the particular structure of the TTSP-graphs and behave really better on this class of graphs than on non-specific graphs. However, the dual cost-scaling approach does not work that well on these instances, whereas it proves to be the most efficient for non-specific graphs. Finally, the aggregation is the method that presents the best performance on TTSP-graphs.

2.2 Aggregation Method

The *aggregation* method, that allows to solve minimum cost tension problems only on TTSP-graphs, has been introduced in [4]. The algorithm works on a SP-tree T of the TTSP-graph G and is recursive: considering a SP-relation in T, it assumes 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.

From the definition of a TTSP-graph, it is obvious that a TTSP-graph has only one source node (i.e. without any predecessor) and only one target node (i.e. without any successor). Hence, the *main tension* $\overline{\theta}$ of a TTSP-graph is defined as the tension between its source s and target t, i.e. $\overline{\theta} = \pi_t - \pi_s$. To get an efficient algorithm, the *minimum cost function* C_G of a TTSP-graph G must be defined. This function represents the cost of the optimal tension where the main tension is forced to a given value:

$$C_G(x) = \min\{\sum_{u \in U} c_u(\theta_u) \mid \theta \text{ a tension}, \bar{\theta} = x\}$$

Let us consider two TTSP-subgraphs G_1 and G_2, and suppose that their minimum cost functions C_{G_1} and C_{G_2} are known. The minimum cost function $C_{G_1+G_2}$ of the SP-relation $G_1 + G_2$ is:

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

Thus, $C_{G_1+G_2}$ is the inf-convolution $C_{G_1} \square C_{G_2}$. The minimum cost function $C_{G_1//G_2}$ of the SP-relation $G_1//G_2$ is:

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

If all the functions c_u are convex, as in the CPLCT problem, the minimum cost function C_G is convex. From this assessment, a simple recursive algorithm [4] for the CPLCT problem has been proposed to build the minimum cost function C_G of a TTSP-graph G.

3 Aggregation for the BCT Problem

In the BCT problem, the minimum cost functions have no specific properties, they are neither convex, nor continuous. Moreover, for each aggregation, the minimum cost functions of the subgraphs must be entirely expressed, because any part may be of interest in the whole process. As we know that the problem is NP-complete, it should be expected that the size to store the detail of a minimum cost function will grow exponentially (in theory) with the size of the graph. To sum up, computing the minimum cost function can be done the rough way and can lead to an explosion in time and space of the algorithm. That justifies the modeling we choose here to represent a minimum cost function.

We consider a minimum cost function C_G as a set of cases, a case e being a set of arcs S_e of G scheduled at their ideal value. Let $I_G = [MIN_G; MAX_G]$ be the interval in which C_G is defined, and W_G the maximum value of C_G. With each case e is associated the feasibility interval $i_e = [min_e; max_e]$ (of the main tension $\overline{\theta}$) in which e can occur, and its cost c_e. As an example, we suppose a single arc u. Its minimum cost function C_u is defined as follows:

$$C_u \begin{cases} W_u = 1 \ ; \ I_u = [a_u; b_u] \\ \\ \text{Case 1: } S_1 = \{u\} \ ; \ c_1 = 0 \ ; \ i_1 = [o_u; o_u] \end{cases}$$

As shown in Figure 3a, the value of C_u is usually $W_u = 1$ (the worst case). If case 1 occurs, i.e. $\theta_u = o_u$, then the cost is 0.

3.1 Parallel Aggregation

In this section, we explain how to build the minimum cost function C_G of a graph G that is the parallel relation $G_1//G_2$ where G_1 and G_2 are two TTSP-subgraphs with their minimum cost functions C_1 and C_2 respectively. But first, let us consider G_1 and G_2 as single arcs u and v with their minimum cost functions C_u and C_v defined as above for u and as follows for v (cf. Figure 3b):

$$C_v \begin{cases} W_v = 1 \ ; \ I_v = [a_v; b_v] \\ \\ \text{Case 2: } S_2 = \{v\} \ ; \ c_2 = 0 \ ; \ i_2 = [o_v; o_v] \end{cases}$$

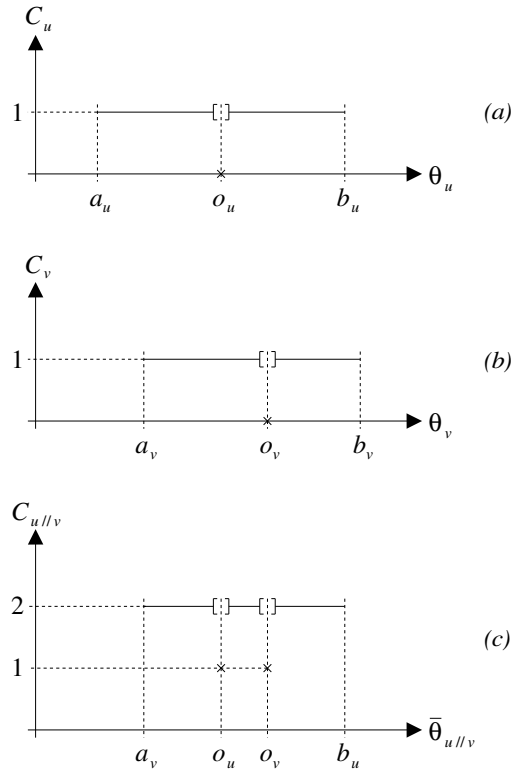


Figure 3: An example of parallel aggregation.

We define now the minimum cost function $C_{u//v}$ of the parallel aggregation of u and v. In the general case, the cost of the function is $W_u + W_v = 2$ (the sum of the general cases of both arcs). Then we must identify all the particular cases:

- Case 1': Case 1 for u and general case for v. The cost is $c_1 + W_v$ and the case is feasible only when both cases occur, i.e. when the main tension $\overline{\theta}_{u//v} \in i_1 \cap I_v$.
- Case 2': General case for u and case 2 for v. The cost is $W_u + c_2$ and the case is feasible when $\overline{\theta}_{u//v} \in I_u \cap i_2$.
- Case 3: Case 1 for u and case 2 for v. The cost is $c_1 + c_2$ and the case is feasible when $\overline{\theta}_{u//v} \in i_1 \cap i_2$.

To sum up, the minimum cost function $C_{u//v}$ is defined as follows (see also Figure 3c):

$$C_{u//v} \begin{cases} W_{u//v} = W_u + W_v \ ; \ I_{u//v} = I_u \cap I_v \\ \\ \text{Case 1': } S_{1'} = \{u\} \quad ; \ c_{1'} = c_1 + W_v \ ; \ i_{1'} = i_1 \cap I_v \\ \text{Case 2': } S_{2'} = \{v\} \quad ; \ c_{2'} = c_2 + W_u \ ; \ i_{2'} = i_2 \cap I_u \\ \text{Case 3 : } S_3 = \{u; v\} \ ; \ c_3 = c_1 + c_2 \quad ; \ i_3 = i_1 \cap i_2 \end{cases}$$

The idea here is to check all the possible cases and determine their feasibility and their cost. Algorithm 1 explains the general process to build the minimum cost function $C_{1//2}$ of the parallel relation $G = G_1//G_2$, from the minimum cost functions C_1 and C_2 of its TTSP-subgraphs G_1 and G_2.

Algorithm 1 *Parallel aggregation.*

```
if I1 ∩ I2 ≠ ∅ then
 W1//2 ← W1 + W2;
 I1//2 ← I1 ∩ I2;

 for all cases e in C1 such that ie ∩ I2 ≠ ∅ do  [All the cases of G1 combined with the general case of G2]
  let e' be a new case of C1//2;
  Se' ← Se;
  ce' ← ce + W2;
  ie' ← ie ∩ I2;
 end for;

 for all cases e in C2 such that ie ∩ I1 ≠ ∅ do  [All the cases of G2 combined with the general case of G1]
  let e' be a new case of C1//2;
  Se' ← Se;
  ce' ← ce + W1;
  ie' ← ie ∩ I1;
 end for;

 [All the cases of G1 combined with all the cases of G2]
 for all pairs of cases e in C1 and f in C2 such that ie ∩ if ≠ ∅ do
  let e' be a new case of C1//2;
  Se' ← Se ∪ Sf;
  ce' ← ce + cf;
  ie' ← ie ∩ if;
 end for;
end if;
```

Proposition 1 *Let n_1 and n_2 be the numbers of cases in C_1 and C_2 respectively. The number of cases of $C_{1//2}$ can not exceed $n_1 + n_2 + n_1 \times n_2$, and the parallel aggregation $G_1//G_2$ requires $O(n_1 \times n_2)$ operations.*

Proof. Algorithm 1 considers $n_1 + n_2 + n_1 \times n_2$ cases (the single cases + the combinations of the cases). □

However, just from the simple example above, it is obvious that for practical instances, lots of combinations of cases will not be possible in parallel (their feasibility interval will be empty). For instance, it seems reasonable to think that case 3 happens rarely in practice.

3.2 Cases Redundancy and Overlapping

Moreover, the number of cases might be virtually overestimated: it is possible, in the same minimum cost function, to get two cases with the same interval. In order to avoid this redundancy (and later useless combinations of cases), one of the cases must be removed: the one with the worst cost (if they are equal, it does not matter which one to eliminate). For instance, if we suppose in the example above that $o_u = o_v$, then cases 1' and 2' become redundant with case 3, so both of them must be removed.

We choose to detect redundancy each time a case e is added to the minimum cost function of a series or parallel relation. We oppose the case e with each case f already in the minimum cost function of the SP-relation. There are three possibilities:

- if $c_e \geq c_f$ and $i_e \subseteq i_f$, there is redundancy of the cases, then case e is not added;
- if $c_e \leq c_f$ and $i_e \supseteq i_f$, there is redundancy of the cases, then case f is removed and e is added;
- otherwise, there may be overlapping of the cases, but e is added.

Notice that we do not attempt to build a minimal set of cases for a SP-relation, i.e. a set of disjoint, i.e. non-overlapping, cases (remark: this is not the set with the minimal number of cases; if you look at Figure 3c, the minimal set would have 5 pieces, whereas $C_{u//v}$, as we build it, has only 3 pieces). As it seems difficult to maintain this minimal set (that is not even the set with the minimum number of pieces), we decide to deal with cases that are overlapping (e.g. cases like e and f where $i_e \cap i_f \neq \emptyset$, but neither $i_e \subseteq i_f$ nor $i_f \subseteq i_e$).

3.3 Series Aggregation

In this section, we explain how to build the minimum cost function C_G of a graph G that is the series relation $G_1 + G_2$ where G_1 and G_2 are two TTSP-subgraphs with their minimum cost functions C_1 and C_2 respectively.

But first, let us consider G_1 and G_2 as single arcs u and v with their minimum cost functions C_u and C_v defined as previously. We define then the minimum cost function C_{u+v} of the series aggregation of u and v. In the general case, the cost of the function is $W_u + W_v = 2$ (the sum of the general cases of both arcs). Then we must identify all the particular cases:

- Case 1': Case 1 for u and general case for v. The cost is $c_1 + W_v$ and the case is feasible on the interval of case 1 "plus" the interval of the general case for v, i.e. when the main tension $\overline{\theta}_{u+v} \in [min_1 + MIN_v; max_1 + MAX_v]$.
- Case 2': General case for u and case 2 for v. The cost is $W_u + c_2$ and the case is feasible when $\overline{\theta}_{u+v} \in [MIN_u + min_2; MAX_u + max_2]$.
- Case 3: Case 1 for u and case 2 for v. The cost is $c_1 + c_2$ and the case is feasible when $\overline{\theta}_{u+v} \in [min_1 + min_2; max_1 + max_2]$.

To sum up, the minimum cost function C_{u+v} is defined as follows (see also Figure 4):

$$C_{u+v} \begin{cases} W_{u+v} = W_u + W_v \;\; ; \;\; I_{u+v} = [MIN_u + MIN_v; MAX_u + MAX_v] \\ \qquad\qquad\qquad\qquad\qquad\quad = [a_u + a_v; b_u + b_v] \\ \text{Case 1':} \;\; \{u\} \;\; ; \;\; c_{1'} = c_1 + W_v \;\; ; \;\; i_{1'} = [min_1 + MIN_v; max_1 + MAX_v] \\ \qquad\qquad\qquad\qquad\qquad\quad = [o_u + a_v; o_u + b_v] \\ \text{Case 2':} \;\; \{v\} \;\; ; \;\; c_{2'} = c_2 + W_u \;\; ; \;\; i_{2'} = [min_2 + MIN_u; max_2 + MAX_u] \\ \qquad\qquad\qquad\qquad\qquad\quad = [a_u + o_v; b_u + o_v] \\ \text{Case 3:} \;\; \{u; v\} \;\; ; \;\; c_3 = c_1 + c_2 \;\; ; \;\; i_3 = [min_1 + min_2; max_1 + max_2] \\ \qquad\qquad\qquad\qquad\qquad\quad = [o_u + o_v; o_u + o_v] \end{cases}$$

The idea here is similar to the parallel aggregation. It is to enumerate all the cases and determine their cost (they are always feasible). Algorithm 2 explains the general process to build the minimum cost function C_{1+2} of the series relation $G = G_1 + G_2$, from the minimum cost functions C_1 and C_2 of its TTSP-subgraphs G_1 and G_2.

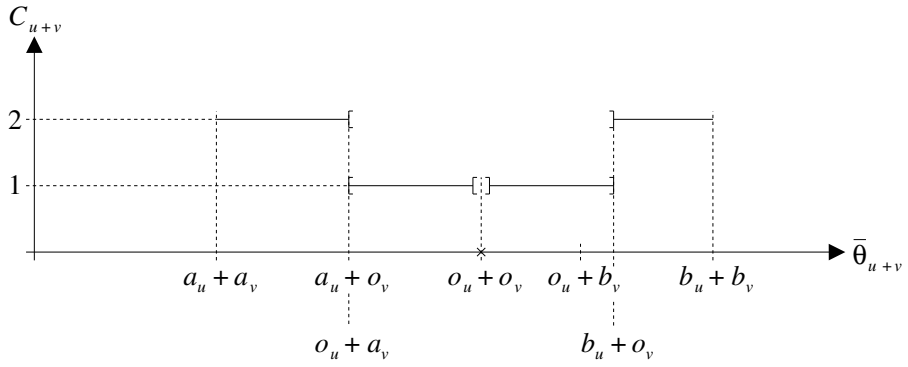


Figure 4: An example of series aggregation.

Proposition 2 *Let n_1 and n_2 be the numbers of cases in C_1 and C_2 respectively. The number of cases of C_{1+2} can not exceed $n_1 + n_2 + n_1 \times n_2$, and the series aggregation $G_1 + G_2$ requires $O(n_1 \times n_2)$ operations.*

Proof. Algorithm 2 considers $n_1 + n_2 + n_1 \times n_2$ cases (the single cases + the combinations of the cases). □

At the opposite of the parallel aggregation, all the cases are possible. However, it is still possible to detect redundant cases to remove. As an indication, on a practical instance of TTSP-graph with 20 nodes and 40 arcs, we observe more than 2 million cases in the final minimum cost function, if we keep the redundant cases. With the constant elimination of these cases, we observe less than a hundred cases in the final minimum cost function (cf. Table 1).

Algorithm 2 *Series aggregation.*

$W_{1+2} \leftarrow W_1 + W_2$;
$I_{1+2} \leftarrow [MIN_1 + MIN_2; MAX_1 + MAX_2]$;

for all cases e in C_1 do *[All the cases of G_1 combined with the general case of G_2]*
 let e' be a new case of C_{1+2};
 $S_{e'} \leftarrow S_e$;
 $c_{e'} \leftarrow c_e + W_2$;
 $i_{e'} \leftarrow [min_e + MIN_2; max_e + MAX_2]$;
end for;

```
for all cases e in C2 do  [All the cases of G2 combined with the general case of G1]
 let e' be a new case of C1//2;
 S_e' ← S_e;
 c_e' ← c_e + W1;
 i_e' ← [min_e + MIN1; max_e + MAX1];
end for;

for all pairs of cases e in C1 and f in C2 do  [All the cases of G1 combined with all the cases of G2]
 let e' be a new case of C1+2;
 S_e' ← S_e ∪ S_f;
 c_e' ← c_e + c_f;
 i_e' ← [min_e + min_f; max_e + max_f];
end for;
```

3.4 Conclusion

Proposition 3 *The aggregation method, for the BCT problem, generates a minimum cost function with at most $2^m - 1$ cases, and requires $O(2^m)$ operations.*

Proof. We can establish that for a TTSP-graph G_k with k arcs, at most $2^k - 1$ cases compose the minimum cost function C_k of G_k. For $k = 1$, it is obvious. Assume now that the property is true for any TTSP-graph with at most k arcs, and consider a TTSP-graph G_{k+1} with $k + 1$ arcs. By definition of TTSP-graphs, it is either a series or a parallel composition of two TTSP-graphs G_p and G_q, with p and q arcs respectively, such that $p + q = k + 1$. From propositions 1 and 2, the minimum cost function of G_{k+1} has $(2^p - 1) + (2^q - 1) + (2^p - 1)(2^q - 1) = 2^{p+q} - 1 = 2^{k+1} - 1$ cases.

Also from propositions 1 and 2, each composition i needs $O(2^{k_i})$ operations, where k_i is the number of arcs in the composition i. As there are $m - 1$ SP-relations in the TTSP-graph [4], the whole aggregation method finally requires $O(\sum_{i=1}^{m-1} 2^{k_i})$ operations (we suppose that the compositions are numbered in the same order they are performed in the aggregation process). It is known that $\sum_{i=1}^{m} 2^i < 2^{m+1}$, and it can be verified by recurrence that $\sum_{i=1}^{m-1} 2^{k_i} \leq \sum_{i=1}^{m} 2^i$, thus the whole aggregation method requires $O(2^m)$ operations. □

4 Numerical Results

As the complexity presented in the previous section represents worst case situations, we must study the practical behavior of the algorithm. As far as we know, the only other method that solves the BCT problem is mixed integer programming. As proposed in [12], the BCT problem can be modeled as a mixed integer program (P_{BCT}). It is based on the generic (P_{MCT}) program defined in Section 1, with the binary cost functions c_u defined by formula *(1)*. These c_u functions are modeled with binary variables y_u and additional constraints (a), (b) and (e).

$$(P_{BCT}) \begin{cases} \text{minimize} \quad \displaystyle\sum_{u \in U} y_u & \\ \\ \text{with} \quad -\theta_u - (o_u - a_u)y_u \leq -o_u, \ \forall u \in U & (a) \\ \qquad \theta_u - (b_u - o_u)y_u \leq o_u, \ \forall u \in U & (b) \\ \\ \qquad \pi_y - \pi_x = \theta_{(x;y)}, \ \forall (x;y) \in U & (c) \\ \\ \qquad a_u \leq \theta_u \leq b_u, \ \forall u \in U & (d) \\ \qquad y_u \in \{0,1\}, \ \forall u \in U & (e) \end{cases}$$

The following tables present a practical comparison of the aggregation technique with mixed integer programming, which is always difficult and questionable due to all kinds of biases. But the goal here is to get an idea of how the methods behave on TTSP-graphs. Results are expressed in seconds on an Intel Xeon 2.4 GHz processor under a Gentoo Linux operating system. We used GNU C++ 3.3 compiler and its object-oriented features to implement the methods, linked to the CPLEX 8.0 Callable Library for mixed integer programming.

Results are based on series of 30 tests on randomly generated graphs[3,4]. To generate an instance, the following parameters must be given: the number of nodes n, the number of arcs m and the tension bound $A = \max_{u \in U} b_u$. A TTSP-graph structure with n nodes and m arcs is randomly built, based on the constructive definition of TTSP-graphs in Section 2. Then, using the topological order of the nodes, a potential π_x is affected to each node x, such that $\theta_u = \pi_y - \pi_x < A$ for each arc $u = (x; y)$. The bounds a_u and b_u are randomly chosen to satisfy $0 \leq a_u \leq \theta_u \leq b_u \leq A$. Finally, the ideal tension o_u is randomly chosen in the $[a_u; b_u]$ interval.

Table 1 shows results where the size of the graphs varies, with $A = 1000$. Each value in columns "CPLEX" and "Aggregation" is the mean of a series of tests, and the number in parentheses is the associated standard deviation. Due to the combinatorial aspect of the problem, the time to solve an instance can vary significantly in a same series. However, these results allow to catch the general evolution of the computation time for each method.

Nodes	Arcs	CPLEX		Aggregation				Time Difference			
		Time		Time		Cases		Aggregation		CPLEX	
n	m										
10	20	0.01	(0.01)	0.01	(0.01)	32.3	(35.9)		(0/30)	0.000	(30/30)
20	40	0.03	(0.03)	0.01	(0.01)	86.4	(68.2)	0.637	(24/30)	0.000	(6/30)
30	60	0.09	(0.14)	0.01	(0.02)	143	(157)	0.749	(26/30)	0.396	(4/30)
40	80	0.43	(0.92)	0.03	(0.03)	231	(264)	0.861	(25/30)	0.290	(5/30)
50	100	4.2	(14.9)	0.08	(0.13)	342	(365)	0.841	(27/30)	0.447	(3/30)
60	120	9.1	(18.5)	0.19	(0.35)	498	(577)	0.884	(27/30)	0.571	(3/30)
70	140	276	(1049)	0.29	(0.58)	445	(574)	0.900	(27/30)	0.842	(3/30)
80	160	982	(5027)	0.22	(0.3)	464	(467)	0.957	(28/30)	0.175	(2/30)
90	180	3172	(6153)	0.9	(1.4)	844	(730)	0.963	(28/30)	0.158	(2/30)
100	200	8490	(19298)	1.1	(3.2)	783	(904)	0.976	(28/30)	0.492	(2/30)

Table 1: Numerical results, graph size influence.

In order to compare the methods, we propose two last columns in the table to present the relative time difference between the methods. We denote $t_{aggregation}$ and t_{CPLEX} the resolution times of the aggregation and linear programming respectively. The first column is the mean of $\frac{t_{CPLEX} - t_{aggregation}}{t_{CPLEX}}$ for the instances of a series of tests where the aggregation is faster than linear programming. At the opposite, the second column is the mean of $\frac{t_{aggregation} - t_{CPLEX}}{t_{aggregation}}$ where linear programming is faster than the aggregation. We choose to present the relative time difference instead of a time ratio like $\frac{time_{CPLEX}}{time_{aggregation}}$ to compare the methods, because extreme situations like a very short time for the aggregation and a long time for CPLEX on the same instance makes the mean estimation useless. If the relative difference is close to 0, that means the resolution times tend to be similar. At the opposite, if the difference is close to 1, that means the resolution times tend to be infinitely different. We also indicate into parentheses the number of times a method is faster than the other. Notice that when the resolution times are identical, it is considered that linear programming is faster.

With these informations, it appears that the aggregation is faster than mixed integer programming for most of the instances. The fact that linear programming is faster in some cases is due to the fact that the aggregation can not deal very efficiently with a chain of series compositions:

[3]Tool available at http://frog.isima.fr/bruno/?doc=bpp_library+ch=build_graph.

[4]Instances available at http://frog.isima.fr/bruno/retrieve.htm?archive=bct_instances.

in a parallel relation, there are cases that can immediately be detected infeasible, whereas in a series relation, all the cases are potentially feasible, but many of them will be eliminated later in the aggregation process. Thus, even if its theoretical complexity is exponential, the aggregation provides a quite efficient way to solve the BCT problem on TTSP-graphs with a size far above what is requested in hypermedia authoring: a 50 nodes and 100 arcs graph is already considered a large instance.

Even in the same series, some instances are easier to solve. If we look at the number of cases generated by the aggregation, there is also a significant variation among the instances of a same series. However, the number of cases is not clearly related to the resolution time. As the number of non-redundant cases is not significant to determine the complexity of an instance, we think that the aggregation explores more redundant cases in the difficult instances. Identifying precisely the reasons of this behavior is not that obvious. Intuitively, the ideal tension is crucial: for instance, there are situations where the tension can not be fixed to its ideal value. The aggregation and mixed integer programming have their own mechanism (the redundancy detection for the aggregation) to detect some situations more easily than others.

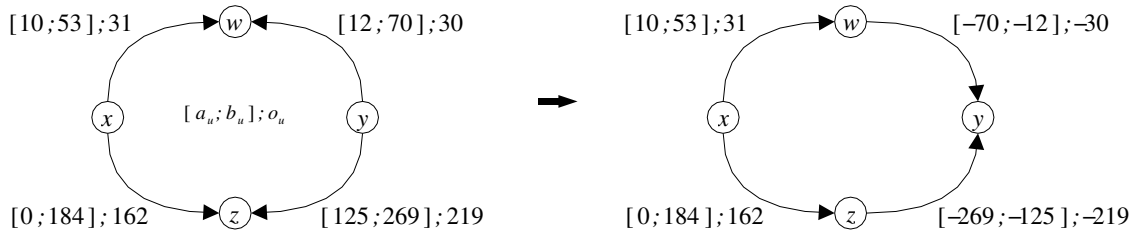
To show that the tension data are significant in the resolution time of both methods, Table 2 proposes results where the tension bound A varies. The aggregation method is still more efficient than mixed integer programming in most cases, but the difficulty to solve the instances, for both methods, increases with A.

Nodes	Arcs	Tension	CPLEX		Aggregation				Time Difference			
n	m	A	Time		Time		Cases		Aggregation		CPLEX	
20	40	1000	0.03	(0.03)	0.01	(0.01)	86.4	(68.2)	0.637	(24/30)	0.000	(6/30)
20	40	10000	0.03	(0.04)	0.01	(0.02)	151	(202)	0.643	(20/30)	0.125	(10/30)
20	40	100000	0.04	(0.03)	0.01	(0.01)	98.4	(80.5)	0.662	(28/30)	0.000	(2/30)
50	100	1000	4.2	(14.9)	0.08	(0.13)	342	(365)	0.841	(27/30)	0.447	(3/30)
50	100	10000	20.2	(50.3)	0.43	(1.2)	722	(836)	0.887	(24/30)	0.718	(6/30)
50	100	100000	16.1	(38.5)	1.5	(3.8)	1225	(1947)	0.921	(24/30)	0.824	(6/30)
80	160	1000	982	(5027)	0.22	(0.3)	464	(467)	0.957	(28/30)	0.175	(2/30)
80	160	10000	13953	(55727)	33.5	(165)	2069	(4284)	0.920	(26/30)	0.902	(4/30)
80	160	100000	15833	(73167)	170	(842)	2337	(4432)	0.968	(27/30)	0.795	(3/30)

Table 2: Numerical results, tension scale influence.

5 BCT Problem on an Elementary Cycle

[3] explains that solving the BCT problem on an elementary cycle C of a graph G can be useful to find cuts to add in the mixed integer program that models the BCT problem on the whole graph G. However, in this article, the problem was solved using mixed integer programming. We propose here to use the aggregation technique to solve exactly, and more efficiently than linear programming, the BCT problem on an elementary cycle.



(a) BCT problem on an elementary cycle *(b) BCT problem on a TTSP-graph*

Figure 5: Transformation of an elementary cycle into a TTSP-graph.

The idea is simply to transform the cycle C (Figure 5a) into a TTSP-graph H (Figure 5b), making the BCT problem (P') on H equivalent to the BCT problem (P) on C. To perform the transformation, the orientation of some arcs of C must be changed so the cycle becomes two directed paths P_1 and P_2 in parallel, e.g. $H = P_1//P_2$ (cf. Figure 5).

Thus, H is two-terminal series-parallel and in order to make (P') equivalent to (P), the tension data of any arc u that has been reversed must be modified as follows (cf. Figure 5): its interval $[a_u; b_u]$ becomes $[-b_u; -a_u]$ and its ideal tension o_u becomes $-o_u$.

It is easy to check that the constraints and the objective functions of both problems are equivalent. Numerical results for elementary cycles (not introduced in this paper because very similar to those presented for TTSP-graphs in the previous section) show that this method to solve the BCT problem on an elementary cycle is more efficient in practice than mixed integer programming.

6 Conclusion

This article provides a first alternative to mixed integer programming in solving the BCT problem on TTSP-graphs. Moreover, the practical efficiency of the aggregation method is far above mixed integer programming. It can thus be useful in the resolution of the BCT problem on non-specific graphs: as shown in Section 5, it provides a way to solve the problem on an elementary cycle, which is relevant to generate cuts for the linear program of the BCT problem [3]. We also hope to find better cuts with the aggregation algorithm: the BCT problem can be solved optimally and quite quickly for TTSP-subgraphs of any graph, which provides bounds for new constraints.

The aggregation method results in more than just an optimal tension. As presented in this article, it provides aggregated information, which allows to adapt optimally the tension of a TTSP-graph to any main tension in polynomial time[5]. It is the idea of the *reconstruction* technique that solves the CPLCT problem on *quasi-k series-parallel* graphs (or *k-QSP* graphs) [5]. A graph is k-QSP if the removal of a minimal subset of arcs (of size k) makes the remainder of the graph two-terminal series-parallel. For the CPLCT problem, the out-of-kilter technique can use efficiently the aggregated information provided by the aggregation technique on TTSP-subgraphs. For the BCT problem, no equivalent method actually exists. However, as the aggregation technique is really efficient for the BCT problem, the study of a reconstruction approach for k-QSP graphs seems interesting.

References

[1] Ravindra K. Ahuja, Dorit S. Hochbaum, and James B. Orlin. Solving the Convex Cost Integer Dual Network Flow Problem. In *Management Science*, volume 49, pages 950–964, 2003.

[2] Bruno Bachelet. *Modélisation et optimisation de problèmes de synchronisation dans les documents hypermédia*. PhD thesis, Université Blaise Pascal, Clermont-Ferrand, France, 2003. http://frog.isima.fr/retrieve.htm?archive=hypermedia_synchronization.

[5] If the minimum cost function C_G of a graph G contains disjoint cases, they can be ordered to use dichotomy to find a case; if there are p cases, it requires $O(\log p)$ operations. If the minimum cost function C_G of a graph G contains overlapping cases (as proposed with the aggregation), they need to be regrouped by cost (at most m sets), and ordered in each set, so that a case can be searched by dichotomy in each set (there are less than 2^m cases), starting from the lowest cost set; thus it requires $O(m^2)$ operations.

[3] Bruno Bachelet, Christophe Duhamel, Philippe Mahey, and Luiz Fernando Soares. Hypermedia Synchronization: Modeling and Optimization with Graphs. In *Information Processing: Recent Mathematical Advances in Optimization and Control*, pages 49–62. Presses de l'Ecole des Mines de Paris, 2004.

[4] Bruno Bachelet and Philippe Mahey. Minimum Convex-Cost Tension Problems on Series-Parallel Graphs. In *RAIRO Operations Research*, volume 37-4, pages 221–234. EDP Sciences, 2003.

[5] Bruno Bachelet and Philippe Mahey. Minimum Convex Piecewise Linear Cost Tension Problem on Quasi-k Series-Parallel Graphs. In *4OR: Quarterly Journal of European Operations Research Societies*, volume 2-4, pages 275–291. Springer-Verlag, 2004.

[6] C. Berge and A. Ghoula-Houri. *Programmes, jeux et réseaux de transport*. Dunod, 1962.

[7] M. Cecelia Buchanan and Polle T. Zellweger. Specifying Temporal Behavior in Hypermedia Documents. In *European Conference on Hypertext '92*, pages 262–271, 1992.

[8] Alak Kumar Datta and Ranjan Kumar Sen. An Efficient Scheme to Solve Two Problems for Two-Terminal Series Parallel Graphs. In *Information Processing Letters*, volume 71, pages 9–15. Elsevier Science, 1999.

[9] David Eppstein. Parallel Recognition of Series-Parallel Graphs. In *Information and Computation*, volume 98-1, pages 41–55, 1992.

[10] Malika Hadjiat and Jean François Maurras. A Strongly Polynomial Algorithm for the Minimum Cost Tension Problem. In *Discrete Mathematics*, volume 165/166, pages 377–394. Elsevier Science, 1997.

[11] Michelle Y. Kim and Junehwa Song. Multimedia Documents with Elastic Time. In *Multimedia '95*, pages 143–154, 1995.

[12] Maira T. Medina, Celso C. Ribeiro, and Luiz F.G. Soares. Automatic Scheduling of Hypermedia Documents with Elastic Times. In *Parallel Processing Letters*, volume 14-1, pages 45–59, 2004.

[13] Celso C. Ribeiro and Eric Sanlaville. On the Complexity of Scheduling with Elastic Times. Technical report, Computer Science Department, PUC-Rio, Rio de Janeiro, Brazil, 2002. http://www-di.inf.puc-rio.br/~celso/artigos/SchedulingElasticTimes.ps.

[14] Berry 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. http://www.cwi.nl/ftp/CWIreports/AA/CS-R9504.ps.Z.

[15] Jacobo Valdes, Robert E. Tarjan, and Eugène L. Lawler. The Recognition of Series Parallel Digraphs. In *SIAM Journal on Computing*, volume 11-2, pages 298–313, 1982.